XRootD

Authentication & Access Control Configuration Reference

9-October-2016

Gerardo Ganis, CERN

Andreas-Joachim Peters, CERN

Andrew Hanushevsky, SLAC

1 Introduction

This document describes the configuration of the security and the default access control components of the extended root daemon (**xrootd)**. Configuration directives use a special prefix for each component that allows you to use a single configuration file. The prefixes are shown in the following table.

Component	Purpose
sec	**Sec**urity authentication
acc	**Ac**cess control (i.e., authorization)

Configuration directives for each component come from a configuration file specified when **xrootd** is started (see the **–c** option **xrootd** option).

Records that do not start with a recognized identifier are ignored. This includes blank record and comment lines (i.e., lines starting with a pound sign, #). This guide documents the **acc** and **sec** configuration directives. Other directives are documented in supplemental guide specific to the component they deal with.

Refer to the manual "**Configuration File Syntax**" on how to specify and use conditional directives and set variables. These features are indispensable for complex configuration files usually encountered in large installations.

By default, security and access control features are disabled. These features can be enabled with the following **xrootd** and **ofs** directives:

Directive	Purpose
xrootd.fslib	Load the shared library implementing the **ofs** and **acc** components.
xrootd.seclib	Load the shared library implementing the **sec** (authentication) component.
ofs.authlib	Load the shared library implementing a special **acc** component.
ofs.authorize	Enables access control, **acc** component.

1.1 Client/Server Authentication

The authentication component is structured as a highly versatile multi-protocol suite. In order to accomplish this task, it is organized into a set of shared libraries:

Shared Library	Purpose
libXrdSec.so	Protocol manager and host-based authentication.
libXrdSecgsi.so	Dynamically loadable GSI authentication.
libXrdSeckrb5.so	Dynamically loadable Kerberos V authentication.
libXrdSecpwd.so	Dynamically loadable password-based authentication.
libXrdSecsss.so	Dynamically loadable simple shared secret authentication.
libXrdSecunix.so	Dynamically loadable unix-based authentication.
LibXrdSec*xxxx*.so	Dynamically loadable xxxx authentication protocol.

This means that **libXrdSec.so** must be available since it is needed to boot-strap additional protocols. The corresponding shared library must be available for each requested protocol (e.g., **krb5**).

For ***servers***, the location of **libXrdSec.so** is specified using the **xroot seclib** directive (see the "**xrd** & **xrootd** Configuration Guide"). Additional libraries are specified using the **sec.protocol** directive documented in this guide.

For ***clients***, the task of deploying shared libraries is more problematic because library placement and location is not immediately obvious. The same rules apply; **libXrdSec.so** and any additional protocol libraries must be available. Typically, these libraries should be placed in one of the directories listed in the client's **LD_LIBRARY_PATH** environmental variable. Alternatively, they can be placed in a well-known linker/loader location (e.g., /usr/local/lib).

The client will load libraries, as available, compatible with the security configuration defined for the server. Thus, the server controls what protocols the client will use, if any. While this potentially simplifies security administration, it does complicate the client-side environment. This is because the client may be potentially running multiple protocols at the same time, depending on what set of servers the client wants to use. Generally, however, this is transparent to the client application.

1.2 Server Authorization

The default authorization component is already built into **xrootd** and needs only to be activated using the **ofs.authorize** directive. If you use the default authorization scheme, you must also create an authorization file that lists client capabilities. The file is specified by the **acc.authdb** directive, Procedures must be developed to properly share this file with all of the servers that rely on it to provide cohesive access control. Fortunately, authorization is only a server-side issue.

Other authorization schemes may be used with **xrootd**. A specific scheme is implemented as a plug-in. The shared library containing the implementation is then specified using the **ofs.authlib** directive.

In a clustered environment, authorization should be enabled on all actual data servers since clients might bypass a redirector and communicate directly with a data server. Consider enabling authorization at the redirector level only if you need to control file requests. Since a request for a file does not implicitly allow actual access to file data; authorization at the redirector level generally does not enhance security but may add significant overhead.

2 Authentication Configuration

2.1 level

```
sec.level {all | local | remote] [relaxed] level [force]

level:   none | compatible | standard | intense | pedantic
```

Function

Specify the request verification level.

Parameters

all The verification level applies to all clients. This is the default.

local The verification level applies only to clients in the server's DNS domain.

remote

The verification level applies only to clients outside the server's DNS domain.

relaxed

applies the specified level of verification to clients that support request verification. Old clients that don't support verification are not included. This option is meant to allow a non-disruptive client software upgrade path.

level Is the verification level:

none requests are not to be verified. This is the default unless the **level** directive specifies otherwise.

compatible

verifies only potentially destructive requests (i.e. those that modify file data or metadata. This provides backward compatibility for old clients that only require read-only access to data.

standard

includes compatible verification plus key requests that access data.

intense

includes standard verification plus additional requests that access metadata.

pedantic

verifies all requests.

force requires verification even for authentication protocols that do not support encryption. Normally, when a client authenticates with a protocol that does not support generic encryption, verification is not employed. This option is meant for debugging purposes as it does not provide enhanced security.

Defaults

sec.level all none

Notes

1) Request verification uses cryptographic signing to ensure that a request has been sent by the client that the server has previously authenticated.
2) Request verification requires that the authentication protocol used to authenticate a client supports generic encryption. Currently, only **gsi** authentication protocol supports generic encryption.
3) A client is considered to be in the same DNS domain when all DNS name components, other than the first one, match corresponding name components of the server.
4) The following section specifies the verification used for each request by specified level other than none.

Example

```
sec.level local none
sec.level remote standard
```

2.1.1 Verification performed by level

Operation	Compatible	Standard	Intense	Pedantic
admin	verified	verified	verified	verified
auth	---	---	---	---
bind	---	---	verified	verified
chmod	verified	verified	verified	verified
close	---	---	verified	verified
decrypt	---	---	---	---
dirlist	---	---	---	verified
endsess	---	---	verified	verified
getfile	verified	verified	verified	verified
locate	---	---	---	verified
login	---	---	---	---
mkdir	---	verified	verified	verified
mv	verified	verified	verified	verified
open read	---	verified	verified	verified
open Write	verified	verified	verified	verified
ping	---	---	---	---
prepare	---	---	---	verified
protocol	---	---	---	---
putfile	verified	verified	verified	verified
query	---	---	---	verified
query special	---	---	verified	verified
read	---	---	---	verified
readv	---	---	---	verified
rm	verified	verified	verified	verified
rmdir	verified	verified	verified	verified
set	---	---	verified	verified
set special	verified	verified	verified	verified
sigver	---	---	---	---
stat	---	---	---	verified
statx	---	---	---	verified
sync	---	---	---	verified
truncate	verified	verified	verified	verified
verifyw	---	---	verified	verified
write	---	---	verified	verified

2.2 protbind

```
sec.protbind hostpat { none | [ only ] protocols }

hostpat: prefix[*][suffix] | [prefix][*]suffix | localhost
```

Function

Bind a set of protocols to one or more hosts.

Parameters

hostpat

The hostname pattern to be used for matching host names. A pattern is a standard DNS name with an optional single asterisk somewhere in the specification. All of the characters prior to the asterisk (i.e., *prefix*) must match the left-most characters of the host name and all of the characters after the asterisk (i.e., *suffix*) must match the right-most characters of the host name. If the *hostpat* does not contain an asterisk, the all of the characters must match.

none Indicates that incoming clients from hosts matching *hostpat* need not supply any credentials.

only Indicates that incoming clients from hosts matching *hostpat* must supply credentials using one of the *protocols* that follow.

protocols

One or more blank-separated protocol ids that are to be bound to the host. Each protocol id must have been previously defined with the **protocol** directive.

localhost

substitutes the DNS registered name of the current host.

Defaults

All of the defined protocols are presented to each connecting client as acceptable authentication protocols. See the notes on how to change the default.

Notes

5) The **protbind** directive allows you to determine which authentication protocols are valid from which host. Alternatively, the **protbind** directive can be used to lessen authentication requirements from certain hosts (e.g., those behind a firewall vs. the ones outside a firewall).
6) Incoming clients from hosts bound to **`none`** are not asked to supply credentials.
7) Order is important. Host matching occurs in reverse order of specification. Specify the most general *hostpat* first and the least general, last.
8) If the *hostpat* is a single asterisk, then this defines the actual default for all unbound hosts.
9) The **protbind** directive is meant to lessen the security requirements on certain hosts. Unless **`only`** is specified, it does not restrict a host from using any defined security protocol, even if that protocol is not presented to the host as an option.
10) Because **host** protocol is the least restrictive authentication mechanism, binding the built-in **host** protocol to a host makes any other bindings to the host superfluous.

Example

```
sec.protbind bronco*slac.stanford.edu host
```

2.4 protocol

```
sec.protocol [ libpath ] protid [ parms ]
```

Function

Define the characteristics of an authentication protocol.

Parameters

libpath

The absolute path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

protid The unique 1- to 7-character protocol identifier.

parms The parameters required by the protocol to operate successfully. The parameters are protocol dependent. The notes and subsequent sections describe the parameters needed for various protocols. The *parms* can also be specified with the **protparm** directive.

Defaults

There are no defaults. Each protocol must be appropriately defined in order for it to be used.

Notes

1) Each supported protocol has a 1- to 4-characterunique identifier, the *protid*. The **sec** component currently comes with support for these protocols:
   - **host** authenticates a user by originating host name only,
   - **gsi** authenticates a user using GSI protocol,
   - **krb5** authenticates a user using Kerberos V protocol, and
   - **pwd** authenticates a user using a password-based protocol
   - **sss** authenticates a user using a simple shared secret protocol
   - **unix** authenticates using the Unix login name and group name

   Other protocols may be supported by an installation. Refer to the "xrootd Developer's Reference" on how to add new protocol support.
2) Even though the host protocol is built-in, it will not be used unless specified with a **protocol** directive.

3) Because **host** protocol is the least restrictive authentication mechanism; allowing its unbound use (see the **protbind** directive) makes all other protocols superfluous. A warning message is issued if you define the host protocol but do not restrict its use to certain hosts.
4) The following sections describe the required parameters, *parms*, for each protocol requiring configuration (**host** protocol does not need any parameters).
5) ***Warning***: **host** and **unix** protocols do not provide any significant level of security and should only be used in instances where security violations do not matter.

Example

```
sec.protocol host
```

2.4.1 **protocol** (gsi)

```
sec.protocol [ libpath ] gsi [-d:level] [-certdir:dir]
                [-cert:file] [-key:file] [-crl:opt]
                [-crldir:dir] [-crlext:extension]
                [-crlrefresh:period]
                [-cipher:ciphers] [-md:mds] [-ca:opt]
                [-dlgpxy:opt] [-exppxy:template]
                [-gmapopt:opt] [-gridmap:file] [-gmapto:to]
                [-gmapfun:file] [-gmapfunparms:parms]
                [-authzfun:file] [-authzfunparms:parms]
                [-authzto:to] [-authzpxy:opt]
                [-vomsat:opt]
                [-vomsfun:file] [-vomsfunparms:parms]
```

Function

Define the characteristics of the **gsi** authentication protocol.

Parameters

libpath

The absolute directory path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

d:*level*

Sets the verbosity level for this module to *level*; the level can be set to 1 (low), 2 (medium) or 3 (high or dump). Invoking **xrootd** with the verbose option *–d* sets the internal verbosity for this module to 1. Default is **0**.

certdir:*dir*

Specifies an alternative directory path for trusted Certificate Authority certificates; the path indicated by *dir* can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`.
Default: **/etc/grid-security/certificates**

cert:*file*

Specifies an alternative path for the file containing the certificate to be used by the server; the path leading to *file* can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`.

Default: **/etc/grid-security/xrd/xrdcert.pem**

key:*file*

Specifies an alternative path for the file containing the private key associated with the server certificate (server must have read access to the file) ; the path leading to *file* can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`.
Default: **/etc/grid-security/xrd/xrdkey.pem**

crl:*opt*

Defines the type of check to be performed on CRLs:

- 0 *do not care*; ignore any CRL information for the CA being used for certificate chain verification;
- 1 *use CRL if available*; if the CRL certificate is missing for a given CA, the related CRL is assumed to be empty;
- 2 *require CRL* for any trusted CA, but do not stop if the CRL certificate is not up-to-date;
- 12 *require CRL* for any trusted CA, and attempt to download the CRL certificate if the file is not found or is not up-to-date;
- 3 *require* an *up-to-date CRL* for each CA;
- 13 *require* an *up-to-date CRL* for each CA, and attempt to download the CRL certificate if the file is not found or is not up-to-date.

Default is **1**.

crldir:*dir*

Specifies an alternative directory path for CRL certificates; the path indicated by *dir* can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. By the default CRLs are searched for in the *same path as for CA certificates*.

crlext:*extension*

Specifies an alternative default extension for CRL files. Default is "**.r0**".

crlrefresh:*period*

Controls *period* for refreshing the CRL information; value in seconds.
Negative values disable the automatic refresh. Default one day (86400 secs).

cipher:*ciphers*

Specifies a colon-separated list of ciphers to be used for the session symmetric key. Default is "**aes-128-cbc:bf-cbc:des-ede3-cbc**" (*OpenSSL* naming convention).

md:*mds*

Specifies a colon-separated list of message digests to be used for integrity checks and signatures. Default is "**sha1:md5**" (*OpenSSL* naming convention).

ca:*opt*

Defines the CA verification level:

- 0 do not verify;
- 1 verify if self-signed, issuing a warning if not;
- 2 always verify the CA in the chain, failing when not possible

Default is **1**.

dlgpxy:*opt*

Specify the server-side requests about the client delegated proxy; the global result depends also on the client settings (see the XrdSecGSIDELEGPXY client environment variable below).

- 0 no specification from the server;
- 1 server will ask the client for a delegated proxy (client settings may be such that such a request is not honored);
- 2 server will save (export) the delegated proxy to a file (see the *exppxy* switch for the location and format);
- 3 server will ask the client for a delegated proxy and save it to a file.

Default is **0**.

exppxy:*template*

Specify the location and format of the delegated proxy file saved when the dlgpxy switch requires it. The template can contain one or more of the following place-holders which are resolved dynamically:

- <user> client username;
- <uid> client user ID;
- <host> client host name;
- <vorg> client virtual organization;
- <group> client group.

Default is */tmp/x509up_u<uid>*.

gmapopt:*opt*

Specify how to handle the grid-map file.

- 0 do not use; hash of the client DN will be used as user identifier (username);
- 1 use if available; otherwise as if 0;
- 2 require (fail at initialization if file is missing or not readable; fail if a DN mapping cannot be found);
- 10 do not use; client DN will be used as user identifier
- 11 use is available; otherwise as if 10.

Default is **1**.

gridmap:*file*

Specify an alternative location for the grid-map to be searched for if *gmapopt* is non null.
Default is */etc/grid-security/grid-mapfile*.

gmapto:*to*

Expiration time in seconds for entries in the cache associated with the grid-map file or the mapping function defined by *gmapfun*. Default is -1, i.e. no expiration.

gmapfun:*file*

Defines the full path to a file containing a plug-in function to be used to map DNs to usernames in addition to the grid-map file. See the dedicated session below for details about the function signature and return codes.

gmapfunparms:*parms*

Defines the parameters to be used to initialize the plug-in mapping function defined by *gmapfun*; multiple parameters are '|'-separated. See the dedicated session below for details.

authzfun:*file*

Defines the full path to a file containing a plug-in function to be called after a successful authentication handshake to complete the authorization information in the XrdSecEntity structure. See the dedicated session below for details about the functions signatures and return codes.

authzfunparms:*parms*

Defines the parameters to be used to initialize the plug-in mapping function defined by *authzfun*; multiple parameters are '|'-separated. See the dedicated session below for details.

authzto:*to*

Expiration time in seconds for entries in the cache associated with the authorization function defined by *authzfun*. Default is 43200 seconds, i.e. 12 hours.

authzpxy:*opt*

Defines if and how the user proxy information is exported in the XrdSecEntity for authorization. Options are entered in the form

*opt = what * 10 + where*

with *what* possibly taking the following values

- 0 full proxy chain (CA, certificate, proxies)
- 1 last user proxy only

and *where*

- 1 in the XrdSecEntity.creds field
- 2 in the XrdSecEntity.endorsements field

Default is **0**, i.e. no export.

vomsat:*opt*

Specify how to handle VOMS attributes

- 0 ignore, i.e. do not look for;
- 1 extract, if any; this will fill *XrdSecEntity.vorg*, *XrdSecEntity.role* and save the full attributes lists in *XrdSecEntity.endorsements* .
- 2 like 1 but generate a handshake failure if the attributes are missing.

Default is **1**.

vomsfun:*file*

Defines the full path to a file containing a plug-in function to be called to extract the VOMS in the XrdSecEntity structure. See the dedicated session below for details about the functions signatures and return codes.

vomsfunparms:*parms*

Defines the parameters to be used to initialize the plug-in VOMS attributes extraction function defined by *vomsfun*; multiple parameters are '|'-separated. See the dedicated session below for details.

Defaults

See description of each single parameter.

Notes

1) Servers usually use a dedicated service-certificate whose CN is of the form

   CN = *service/Fully.Qualified.Hostname*

   e.g. CN = xrd/pcepsft43.cern.ch

   To make servers to use standard user certificates specify the coordinates of the certificate and its key with the **cert** and **key** keywords above. In such a case, the *pass-phrase* for the private key will be prompted for at server start-up.
2) The possibility to specify the extension for CRL certificate files is provided to speed-up loading of certificates; if the extension or the corresponding file are missing, the whole set of files in the directory is tested and the relevant file eventually found if present.

Example

```
sec.protocol gsi -crl:3
```

2.4.1.2 Grid-map function

This function can be used to integrate or replace the grid-map file. The function is loaded as a plug-in from the file specified by the *gmapfun* switch, must e declared as extern “C” and must have the following name and signature

```
extern “C” {
char *XrdSecgsiGMAPFun(const char *DN, int now)
{
     // DN is the user DN
     // now is the result of time(0)
     …
     char *name = new char[length];
     …
     return name;
}}
```

The function is called once after loading with the parameters defined by *gmapfunparms* in the first argument (any ‘useglobals’ is removed; see below) and *now = 0* for initialization. At this level the parameters are separated by a ‘ ‘.
Working examples can be found in the distribution in the files

XrdSecgsiGMAPFunDN.cc and *XrdSecgsiGMAPFunLDAP.cc*

under *src/XrdSecgsi* . See also the makefiles in the module for examples of how to build these plug-ins.

2.4.1.3 Authorization function

This function can be used to complete or modify the content of the XrdSecEntity structure for authorization purposes. The function is part of a set of three functions loaded from the file specified by the *authzfun* switch. In addition to the main function, the plug-in should contain a function defining the string to be used to key the result of the call, and a function to initialize the plug-in.
The three functions must all declared as 'extern “C”'.
The main function has the following signature and name:

```
int XrdSecgsiAuthzFun(XrdSecEntity &entity)
```

where *entity* is the XrdSecEntity object associated with the handshake on the server side. On input *entity* contains:

- in *name* the username, DN or DN hash according to the GMAP option;
- in *host* the client hostname;
- in *creds* the proxy chain .

The proxy chain can be either in *raw opaque* or *PEM base64* format (see below).

This function returns

0	on success
<0	on error (implies authentication failure)

The initialization function has name and signature

```
int XrdSecgsiAuthzInit(const char *parms)
```

where *parms* is a string containing a ' '-separated list of parameters.
This function return <0 in case of failure or the format type of the proxy chain expected by the main function:

0	raw, to be used with XrdCrypto tools
1	PEM base64 standard string

The key function has name and signature:

```
int XrdSecgsiAuthzKey(XrdSecEntity &entity, char **key)
```

where *entity* is the XrdSecEntity object associated with the handshake on the server side. On input *entity.creds* contains the proxy chain, with the same convention for the format as above. The function is expected to fill in **key* the key to be used to cache the result of the main function and to return the length of the key. The key will be destroyed with *delete [],* so it must be allocated internally with *new char[].*

A working example can be found in the distribution in the file

XrdSecgsiAuthzFunDN.cc

under *src/XrdSecgsi* . See also the makefiles in the module for an example of how to build the plug-in.

2.4.1.3.1 The AuthzVO Plug-in

The GSI package comes with a general-purpose AuthzVO plug-in in the form of **libXrdSecgsiAuthzVO.so** and can be used for simple mapping of virtual organization (**VO**) names to usernames or groups. It also, by default, trims the un-mapped usernames to the base distinguished name contained in the certificate. These actions are controlled by the **authzfunparms** parameter. The parameter is specified in the form of a cgi string (i.e., keyword=value separated by an ampersand (&)). Valid keyword value pairs are explained below

```
-authzfunparms:keyword=value[&keyword=value[. . .]]

keyword:  debug=1 validо=vlist

          vo2grp=gspec vo2usr=uspec
```

where:

debug=1

Prints additional information involved in the mapping ans should only be used for debugging purposes.

valido=*vlist*

vlist is a comma-separated list of vo names that are acceptable. If not specified, all vo's are accepted. Otherwise, failure is returned if the vo is not in the list of vo's.

vo2grp=*gspec*

specifies how the vo name is to be converted into a group name. Specify for *gspec* a printf-like format string with a single %s. The vo name is inserted where the %s occurs. To make the vo name equal to the group name, specify only "%s" (i.e. **vo2grp=%s**). If **vo2grp** is not specified, the group name is unchanged.

vo2usr=*uspec*

specifies how the vo name is to be converted into a user name. Specify for *uspec* a printf-like format string with a single %s. The vo name is inserted where the %s occurs. To make the vo name equal to the user name, specify only "%s" (i.e. **vo2usr=%s**). If **vo2usr** is not specified, then the user name comes from distinguished name in the certificate (i.e. text after '/CN=') with spaces turned into underscores and the vo name is not used. Specifying **vo2usr=*** returns the user name as set by the gsi plug-in.

Notes

1) The AuthzVO plug-in is best used when gsi internal mapping is turned off. Normally, this requires that you also specify '**-gmapopt:10 -gmapto:0**' options.

Example

```
-authzfun:libXrdAuthzVO.so \
-authzfunparms:valido=atlas,cms,vo2grp=us%s \
-gmapopt:10 -gmapto:0
```

The above example loads the AuthzVO plug-in. The parameters indicate that only vo names of atlas and cms are valid. The cms vo name will be converted to a group name of **uscms** and atlas vo will be converted to a group name of **usatlas**. The user name comes from the distinguished name in the certificate.

2.4.1.4 VOMS attributes extraction function

This function can be used to fill the VOMS-related content of the XrdSecEntity structure for authorization purposes. The function is part of a set of two functions loaded from the file specified by the *vomsfun* switch. In addition to the main function, the plug-in should contain a function to initialize the plug-in.
The two functions must all declared as 'extern "C"'.
The main function has the following signature and name:

```
int XrdSecgsiVOMSFun(XrdSecEntity &entity)
```

where *entity* is the XrdSecEntity object associated with the handshake on the server side. On input *entity* contains:

- in *name* the username, DN or DN hash according to the GMAP option;
- in *host* the client hostname;

- in *creds* the proxy chain .

The proxy chain can be either in *raw opaque* or *PEM base64* format (see below).
This function returns

0	on success
<0	on error (implies authentication failure)

The initialization function has name and signature

```
int XrdSecgsiVOMSInit(const char *parms)
```

where *parms* is a string containing a ' '-separated list of parameters.
This function return <0 in case of failure or the format type of the proxy chain expected by the main function:

0	raw, to be used with XrdCrypto tools
1	PEM base64 standard string

A working example can be found in the distribution in the file

XrdSecgsiVOMSFunLite.cc

under *src/XrdSecgsi* . See the top of the file for instruction about modifying the relevant cmake files to build the plug-in.

2.4.1.4.1 The libXrdSecgsiVOMS.so plug-in

A general-purpose VOMS extraction plug-in using the official VOMS client libraries is available at https://github.com/gganis/voms.git . This extraction plug-in verifies the signature of the VOMS attributes and is able to parse correctly VOMS proxy certificates created with different versions. To build the plugin library VOMS version 2.x is required.
Please refer to the reference above for configuration details.

2.4.1.5 The *useglobals* parameter

Both the grid-map and authorization functions plug-ins support the special parameter *useglobals*. This parameter is detected and used *before* the plug-in is actually loaded and controls the way the symbols in the plug-in library are made available to the subsequent libraries, e.g. the ones loaded by the plug-in itself. If *useglobals* is added to the parameters list the symbols are made globally available, which means that *dlopen* is called with the flag *RTLD_GLOBAL* set. The useglobals parameter is removed from the list passed for the initialization call.

2.4.1.6 Configuring GSI Security

2.4.1.6.1 Server side

Follow these steps to configure GSI protocol for **xrootd**:

1. Locate a valid certificate to be used by the server. This may be a *service* certificates have the *Common Name* CN in the form

   CN = *service/Fully.Qualified.Hostname*

   and private key files protected only by the file system permissions (typical permission mask is 0400). The CN is the final part of the subject name which can be displayed using, for example, the command

   openssl x509 –in Certificate.pem -subject

   (use '*-text*' in place of '*-subject*' to display the full certificate content).
   Hosts have usually service certificates with CN of the type **host/*hostname.dom.ain*** ; however, the associated private key files typically require `root` privileges to be read. If a valid certificate cannot be located, follow CA specific instructions for submission of a certificate request.
2. Locate the directory path with the certificates for the trusted CAs. File names holding CA certificates are of the type *<subject_hash>.0*; for example, for the CERN CA, the certificate file is "**1d879c6c.0**".
3. If strong requirements about the CRLs have to be applied, locate the CRL certificate files; these are usually located in the same directory as the CA certificates, have the same name but extension "*.r0*"; for the example above, the CRL file is "**1d879c6c.r0**".
4. If some non-default values are found at points 1-3, add the relevant parameters in the *sec.protocol* directive of the configuration file.

2.4.1.6.2 Client side

The default settings should be adequate for most of the use cases. Exceptions to the rule may be the location of the CA certificates and related CRLs, and the strength of the requirement about CRLs.

The following **environment variables** are provided to change the defaults on the client side:

XrdSecDEBUG — verbose level; the level can be set to 1 (low), 2 (medium) or 3 (high or dump). Default is **0**.

XrdSecGSIUSERCERT or X509_USER_CERT
: alternative full path to the file containing the certificate to be used by the client; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default: **$HOME/.globus/usercert.pem**

XrdSecGSIUSERKEY or X509_USER_KEY
: alternative full path to the file containing the private key associated with client certificate; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default: **$HOME/.globus/userkey.pem**

XrdSecGSIUSERPROXY or X509_USER_PROXY
: alternative full path to the file containing the user proxy certificate to be used by the client; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default **/tmp/x509up_u*uid***

XrdSecGSIPROXYVALID — validity of the proxy certificate; formst is in the form "hh:mm". Default is **"12:00"**, i.e. 12 hours.

XrdSecGSIPROXYKEYBITS — bit strength of the proxy PKI. Default is **512**.

XrdSecGSIPROXYDEPLEN number of children generations which can originate from this proxy. Controls delegation. Use -1 for infinite. Default is **0**.

XrdSecGSICADIR or X509_CERT_DIR
alternative full path to the directory containing the CA certificates; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default: **/etc/grid-security/certificates**

XrdSecGSICRLDIR or X509_CERT_DIR
alternative full path to the directory containing the files with CRL information for CAs; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default: **/etc/grid-security/certificates**

XrdSecGSICACHECK defines CA verification level:
0 do not verify;
1 verify if self-signed, issuing a warning if not;
2 always verify the CAs in the chain, failing if not possible.
Default is **1**.

XrdSecGSICRLCHECK type of check to be performed on CRLs:
0 do not care; ignore any CRL information for the CA being used for certificate chain verification;
1 use CRL if available (if the CRL certificate is missing for a given CA, the related CRL is assumed to be empty);
2 *require* CRL for any trusted CA, but do not stop if the CRL certificate is not up-to-date;
12 *require CRL* for any trusted CA, and attempt to download the CRL certificate if the file is not found or is not up-to-date;
3 *require* an *up-to-date* CRL for each CA;
3 *require* an *up-to-date CRL* for each CA, and attempt to download the CRL certificate if the file is not found or is not up-to-date.
Default is **1**.

XrdSecGSICRLDIR	alternative full path to the directory containing the CRL certificates; the path can be absolute or relative to the place where the daemon is started; '~/' is expanded to `$HOME`. Default is the *same as for CA certificates*.
XrdSecGSICRLEXT	alternative default extension for CRL certificate files. Default is **".r0"**.
XrdSecGSIDELEGPROXY	defines behavior with respect to proxy delegation: 0 do nothing; 1 ask the server to send back a proxy request to be signed; this setting automatically enables the permission to sign requests (see below); 2 send over the local proxy file to the server for further handshakes (this includes proxy private key); Default is **0**.
XrdSecGSISIGNPROXY	controls permissions to sign requests proxy issued by the server: 0 deny; 1 allow (automatically enabled if client is in ask-for-proxy mode). Default is **1**.
XrdSecGSISRVNAMES	Define valid CNs for the server certificates; default is null, which means that the server CN must be in the form *"*/<hostname>"*. The string may contain multiple format specifications separated by a '|'. Each specifications can contain the *<host>* or *<fqdn>* placeholders which are replaced by *XrdSecEntity.host*; they can also contain the '*' wildcard. A '-' before the specification will deny the matching CN's; the last matching wins.

2.4.1.7 xrdgsiproxy

```
xrdgsiproxy [mode] [-debug] [-f file] [-out file]
                [-certdir dir] [-cert file] [-key file]
                [-bits bits] [-valid valid]
                [-path-length len]
```

Function

Stand-alone application to browse, create or destroy a user proxy certificate.

Parameters

mode Specifies the operation mode:
- *info* display content of the proxy certificate;
- *init* create a proxy certificate;
- *destroy* destroy existing proxy certificate.

Default is *info*.

debug

Run in verbose mode.

f *file*, **out** *file*

Alternate location of the proxy file. Default: **/tmp/x509up_u*uid***

certdir *dir*

Alternate directory path for trusted Certificate Authority certificates. Default: **/etc/grid-security/certificates**

cert *file*

Alternate path for the user certificate file.
Default: **$HOME/.globus/usercert.pem**

key *file*

Alternate path for the file containing the private key associated with the user certificate. Default: **$HOME/.globus/userkey.pem**

bits *bits*

Bit strength of the proxy PKI. Default is **512**.

valid *valid*

Validity of the proxy certificate. Default is **12:00**, i.e. 12 hours.

path-length *len*

Number of child generations which can originate from this proxy (i.e., control delegation). Use -1 for infinite. Default is **0**.

2.4.2 **protocol** (host)

```
sec.protocol host
```

Function

Enable host protocol to be used.

Defaults

Even though the host protocol is built-in; it will not be used unless specified with a **protocol** directive.

Notes

1) Because **host** protocol is the least restrictive authentication mechanism; allowing its unbound use (see the **protbind** directive) makes all other protocols superfluous. A warning message is issued if you define the host protocol but do not restrict its use to certain hosts.
2) ***Warning***: **host** protocol does not provide any significant level of security and should only be used in instances where security violations do not matter.

Example

```
sec.protocol host
```

2.4.3 **protocol** (krb5)

```
sec.protocol [ libpath ] krb5 [ kfn ] [ -ipchk ] sid
```

Function

Define the characteristics of the **krb5** authentication protocol.

Parameters

libpath

The absolute directory path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

kfn The full pathname to the file that contains encryption and decryption keys for the protocol. The default **keyfile** name is protocol dependent.

-ipchk

Verifies that credentials are always presented from the same host that actually obtained the credentials.

sid The service principal name used for authentication.

Defaults

The keyfile location is defined by Kerberos V to be /etc/v5srvtab. IP checking is disabled (i.e., the host that obtained credentials need not be the same as the one that supplies the credentials).

Notes

1) The **noipcheck** option is provided for **AFS** Kerberos support (i.e., you must specify **noipcheck** when using **AFS** kerberos) as well as support for installations that wish to forward tickets from host to host.

Example

```
sec.protocol krb5 /etc/krb5keys xrdserv
```

2.4.3.1 Configuring Kerberos V Security

Follow these steps to configure Kerberos V protocol for **xrootd**:

1. Create a principal in the Kerberos authentication database. This principal will be the **xrootd** "service name". While the default is to create a different instance[1] of the principal for every machine on which **xrootd** runs, it is much easier to create a single instance for all machines, especially if you have many machines.

2. Install a keytab file containing the principal's key string. The keytab file may be generated by using **kadmin**[2] command in Unix and the **Ktpass** command in Windows, as follows:

   Unix:

   ```
   kadmin
   ktadd -k filename principal
   ```

   Windows:

   ```
   Ktpass –princ principal –pass pswd -out filename
   ```

   Substitute for *filename* the name of the keytab file you wish to create or the name of an existing keytab file to which you wish to add a key. For *principal*, substitute the name of the principal you created in the previous step. Consult the man pages on **kadmin** and **Ktpass** for more information.

   1. Place the srvtab file in a secure location on each server.

   2. In the **xrootd** configuration file code the protocol directive using the location of the srvtab file and the service name principal, as previously described.

   3. If you are using Kerberos V with **AFS**, make sure to *not* specify the **-ipchk** protocol directive option since **AFS** does not handle ticket **IP** addresses.

[1] Kerberos V service principals are of the form "name/hostname".

[2] The command requires the "inquire" administrative privilege.

2.4.4 **protocol** (pwd)

```
sec.protocol [ libpath ] pwd [-d:level] [-dir:dir]
                [-vc:level] [-syspwd] [-maxfail:num]
                [-lf:lifetime] [-a:option] [-c:list]
                [-upwd:option] [-udir:dir]
                [-cryptfile:file]
```

Function

Define the characteristics of the **pwd** authentication protocol.

Parameters

libpath

The absolute directory path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

d:*level*

Sets the verbosity level for this module to *level*; the level can be set to 1 (low), 2 (medium) or 3 (high or dump). Invoking **xrootd** with the verbose option *–d* sets the internal verbosity for this module to 1. Default is **0**.

dir:*dir*

Specifies the directory path where to look for the password file; if this directive is missing, the password file is searched for under **$HOME/.xrd**; the alternative directory path can be either absolute (begins with '/'), relative to `$HOME` (begins with '~'), or relative to directory where the daemon is started. Examples:

-dir:/etc/xrd
    make use of /etc/xrd/pwdadmin as password file

-dir:~/local/xrd
    make use of $HOME/local/xrd/pwdadmin as password file

-dir:xrd
    make use of $PWD/xrd/pwdadmin as password file

vc:*level*

Specifies the level of verification of client identity:

0 no additional check is done; the exchanged information packet could potentially be re-used for a reply attack.

1 verify timestamp signature; this limits the time window for reply attacks to 5 min.

2 verify random nonce signature; this choice eliminates the risk of reply attacks; it requires an additional exchange.

Default is **2**.

syspwd

Instructs the server to check also the system password file; the right privileges must be owned by the server to be able to do this operation.

maxfail:*num*

Specifies the maximum number of unsuccessful attempts allowed before the related user tag is blocked. Option *disabled* by default.

lf:*lifetime*

Specifies the lifetime of the current password; when a time interval longer than this value is elapsed from the last the password change, the user is asked to change its password at next login. The format is

"<years>y:<days>d:<hours>h:<minutes>m:<seconds>:s"

e.g. "1y:182d:12h" for one year and a half. Lifetime is *infinite* by default.

a:*option*

Specify the set of users allowed to auto-register. Possible choices

0 none

1 users with an account on the machine (according to getpwnam) or with an enabled entry in the password file;

2 everybody.

Default is **0**.

c:*list*

Specifies the list of supported cryptographic modules; the default is 'ssl|local', with 'ssl' indicating the module based on OpenSSL, and 'local' an implementation of cipher-related functionality written by A. Pukall (http:/membres.lycos.fr/pc1/) and provided for backup in the case OpenSSL is not available.

upwd:*option*

Specify whether the server should also consider password files provided by users having an account on the server node; possible choices:

- 0 ignore user password files
- 1 use user password file $USERHOME/.xrd/pwduser, where $USERHOME is the user home directory as returned by getpwnam; the default subdirectory .xrd can be changed with –udir (see below).
- 2 check also password files with crypt-like password hashes; the default name for the file is $USERHOME/.xrdpass and can be changed with –cryptfile (see below). This option is provided mostly for backward compatibility with ROOT daemons.

Default is **0**.

udir:*dir*

Specify alternative user sub-directory for the user password file; if existing, the file read is "`$HOMEUSER`/*dir*/pwduser". Default: **.xrd**.

cryptfile:*file*

Specify alternative name for file with crypt-like password hash when option *-upwd:2* has been specified. Default: **.xrdpass**

Defaults

See description of each single parameter.

Example

```
sec.protocol pwd -a:1
```

2.4.4.1 Configuring pwd Security

2.4.4.1.1 Server side

Follow these steps to configure the password-based protocol for **xrootd**:

1. Create a password file. To create the file in the default location ($HOME/.xrd/pwdadmin) just run **xrdpwdadmin** (see below). Add a contact e-mail and the host name with

   xrdpwdadmin add –host *FQDN* -email *e.mail@my.domain*

2. Add entries for the users, e.g.

   xrdpwdadmin add usertag

   The file *$HOME/.xrd/genpwd/usertag* is created: it contains information about the temporary password and the public cipher initiators of the server. This file should be sent in a secure way to the user for which *usertag* has been created.
3. Users with an account on the system may be allowed by the server administrator to define their own password file, a sort of auto-registration. Option *'-upwd:1'* enables this feature. A user-password file (default coordinates *$HOME/.xrd/pwduser*) can be created in the same way as the main password file, changing the mode with *'-m user'*; only entries tagged with the file owner username are processed.

2.4.4.1.2 Client side

There are two files relevant for the client: the auto-login file (default *$HOME/.xrd/pwdnetrc*) and the file with the server cipher public initiators (default *$HOME/.xrd/pwdsrvpuk*). These files are created automatically by the code initializing the client. However, the user can browse and modify them with **xrdpwdadmin**. In particular, when the client receives the file *pwdfile* with password and cipher information, it can import the content as follows

xrdpwdadmin –m srvpuk –import *pwdfile*
xrdpwdadmin –m netrc update –import *pwdfile*

By default **auto-login** is switched-off. It can be switched using the appropriate environment variable (see below). The auto-login system understands any '*' as a wild character. A valid entry applying to a set of host of similar name can be copied into an entry with a wild char using *copy*; for example, after

xrdpwdadmin –m netrc copy usertag@lxplus076.cern.ch usertag@lxplus*

the password associated with *usertag@lxplus** will be used for a first login attempt to any machine of the LXPLUS cluster (depending on the shell, the '*' may need to be escaped in the above *copy* command).

Clients can require at any moment a password change by prefixing the password with the string $changepwd$, e.g. if the current password for usertag is curpwd, and the string "$changepwd$curpwd" is entered at password prompt, the client will be prompted again for the new password.

The following **environment variables** are provided to change the defaults on the client side:

XrdSecDEBUG	verbose level; the level can be set to 1 (low), 2 (medium) or 3 (high or dump). Default is **0**.
XrdSecPWDAUTOLOG	switch ON (=1) or OFF (=0) use of autologin information; default is **0** (OFF).
XrdSecPWDALOGFILE	full path to the file with autologin information. Default: **$HOME/.xrd/pwdnetrc**
XrdSecPWDVERIFYSRV	switch ON (=1) or OFF (=0) verification of server identity; verification requires the signature of a random nonce, which implies an additional exchange. Default is **1** (ON).
XrdSecPWDSRVPUK	full path to the file with server cipher initiators. Default: **$HOME/.xrd/pwdsrvpuk**

2.4.4.2 xrdpwdadmin

```
xrdpwdadmin [-m mode] action [tag] [newtag] [-f file]
            [-force] [-crypto list]
            [[-[no]passwd] [-[no]random] [-[no]change]
            [-import file]
            [-host name] [-email mail]
            [-iternum num] [-changepwd]
```

Function

Stand-alone application to browse, create or modify password and auto-login files.

Parameters

mode specifies the operation mode:

admin manage a general password file (server side);
user manage user-specific password file (server side);
netrc manage an auto-login file (client side);
srvpuk manage a file with public cipher initiators (client side).
Default is **admin**.

action specifies the action to be performed on the file:

add add new entry; requires specification of *tag*
update update information about an entry; requires specification of *tag*
remove remove all information about a tag; requires specification of *tag*
disable disable logins for a given tag; requires specification of *tag*
copy create a new tag as exact copy of an existing tag; requires specification of *tag* and *newtag*
browse browse the content of the file
trim eliminate obsolete / not-reachable information after a remove
Default is **browse**.

tag, newtag

String of characters of any length identifying a set of entries in the password file.

f *file* specifies an alternative path for the file to manage/create.

[no]passwd

[**add**, **update** *only*] Controls whether a password should be assigned to the entry or not; by default a password is assigned; if *–nopasswd* is specified, the entry is just activated and, if auto-registration is allowed, the user will be asked to set a password the first time she logs in.

[no]random

[**add**, **update** *only*] Controls randomness of a password; by default the new password is a random string of 8 printable chars; if *–norandom* is specified, the caller is prompted for a password; in *update* mode, a check is done to ensure that the new password is different from the current one.

[no]change

[**add**, **update** *only*] Controls type of password; by default the new password is a *one-time only* password to be changed upon first use; if *–nochange* is specified, the password type is directly set to *normal*.

import *file*

[**netrc**, **srvpuk** *only*] Import information from the file received by the administrator of the password file.

changepwd

[**netrc** *only*] Set password status to *one-time-only* so that at next login a password change handshake is triggered.

host *hostname*

[**admin**, **user** *only*] Add a special entry with the host name to the password file.

email *e-mail*

[**admin**, **user** *only*] Add a special entry with a contact e-mail to the password file.

iternum *num*

[**admin**, **user** *only*] Number of iterations to be used in hashing within the key derivation function. This is the main factor limiting time performance. Default is 10000 (J. Viega, M. Messier, *Secure Programming Cookbook*, p. 141).

crypto *list*

List of cryptographic modules (separated by a vertical bar) to be used if available. Default is 'ssl|local'.

force

Forces execution of the requested action even if it modifies existing information. When used together with *add*, it is equivalent to *update*.

2.4.5 **protocol** (sss)

```
sec.protocol [ libpath ] sss [options]

options: [-c cktpath] [-e etype] [-l lifetime]

         [-r refresh] [-s sktpath]
```

Function

Define the characteristics of the **s**imple **s**hared **s**ecret (**sss**) authentication protocol.

Parameters

libpath

is the absolute directory path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

-c *cktpath*

is the absolute file-path to the client's key table. The key table file must only be readable by the username under which the client is running, or group readable if the file name ends with ".grp". This is provided as a hint to the client and may be over-ridden (see the notes).

-e *etype*

is the encryption type to be used when transmitting secret information. Valid types are:

bf32 Blowfish encryption with CRC32 message validation (the default).

-l *lifetime*

is the maximum lifetime of any encrypted message sent between the client and server. Messages older than the specified number of seconds are rejected. The default is 13 seconds.

-r *refresh*

is the key table file refresh interval. The server checks every *refresh* minutes if minimum *refresh* value is 10 minutes. The default is 60 minutes.

-s *sktpath*

is the absolute file-path to the server's key table. The key table file must only be readable by the username under which the server is running or group readable if the file name ends with ".grp". The default is "$HOME/.xrd/sss.keytab".

Defaults

See description of each single parameter.

Notes

1) Default locations exist for client- and server-side key table files. This is "$HOME/.xrd/sss.keytab". Use the **–c** option to over-ride the default for the client and the **–s** option to over-ride the default for the server. Note that the client may specify a fixed location for the key table file irrespective of any explicit or implicit default.
2) The **–l** option controls how long an encrypted message may remain valid. This minimizes replay attacks. However, if server or client response times are slower than the lifetime of the message, the parties will not be able to authenticate. In this case, you should specify a larger time.
3) The **xrdsssadmin** command is used to create and maintain keytables that contain simple shared secrets. The command also is used to specify how shared secrets map to authenticated user and group names.
4) An "**sss**" authenticated client can set its identity using an instance of an object in the **XrdSecsssID** class. See the comments in **XrdSecsssID.hh** source file for detailed information.

Example

```
sec.protocol sss –r 30 –s /opt/xrootd/.xrd/sss.keytab
```

2.4.5.1 Configuring sss Security

2.4.5.1.1 Server side

For very simple[3] installations, follow these steps to configure the **sss** protocol for **xrootd**:

1. Create a key table file. To create the file in the default location ($HOME/.xrd/sss.keytab) just run **xrdsssadmin** (see below[4]) under the *same* username that will be used for **xrootd**:

   xrdsssadmin add

2. Distribute the key table file to the hosts that need to share the secret in the key table file. You can use a combination of **ssh** and **xrdsssadmin** to securely do this. For example, executing the following under the *same* username that will be used for **xrootd**

   cat $HOME/.xrd/sss.keytab | ssh *user@host* xrdsssadmin install

   where *user* is the username that will be used to run the client application on node *host*. This will create the file $HOME/.xrd/sss.keytab on the machine named *host* only readable by *user*.

2.4.5.1.2 Client side

The only file that the client needs is the key table file. This should have been distributed using the procedure outlined in the previous section. Two environmental variables control client-side execution:

XrdSecDEBUG verbose level; the level can be set to 1 (on). Default is **0**.

XrdSecSSSKT holds the location of the key table file. When set, the key table *must* exist as specified in the environmental variable. When not set, the default is provided by the server initiating the authentication protocol. If the server does not provide a default or if the key table is not found there, $HOME/.xrd/sss.keytab becomes the default.

[3] For more complicated installations refer to the **xrdsssadmin** command.

[4] For more complicated installations, see the description of the **xrdsssadmin** command.

2.4.5.2 xrdsssadmin

```
xrdsssadmin [options] action [ keyfn[.grp] ]

action:     add | del | install | list

options:    [-d] [-g group] [-h hold] [-k keyname[+]]

            [-l keylen] [-n keynum] [-s {c|g|k|n|u|x}]

            [-u user] [-x {days | mm/dd/yy} ]
```

Function

Stand-alone application to browse, create or modify the key table for the **sss** protocol.

Parameters

keyfn[**.grp**]

is the name of the key table file which is the target of the *action*. If the name ends with **.grp** then the file is allowed to have group access read-mode bits set. Otherwise, the file may only be read and written by the owner. If *keyfn* is not specified, $HOME/.xrd/sss.keytab is used. See the notes for details.

add adds a new key to the key table. The new key may be assigned an optional *keyname*, *group*, and *user*; with defaults of **anywhere**, **nogroup**, and **nobody**, respectively.

del deletes an existing key from the key table. Only keys matching **-g**, **-k**, **-n**, and **-u** options, as specified, are deleted.

install creates or replaces an existing key table. Only keys matching **-g**, **-k**, **-n**, and **-u** options, as specified, are installed. The key table is read from standard in; making the command suitable for use with **ssh** to provide a secure in-place update of a key table on a remote host.

list displays the contents of the key table. Only keys matching **-g**, **-k**, **-n**, and **-u** options, as specified, are displayed.

-d turns on debugging output.

-g *group*
specifies an optional group name. The resulting effect is dependent on the *action*. See the notes for a detailed explanation.

-h *hold* is the maximum number of keys with the same *keyname* that are to be held in the key table file. The default *hold* value is 3. See the notes for more information.

-k *keyname*[**+**]
specifies an optional key name. The resulting effect is dependent on the *action*. If the *keyname* ends with a plus sign (**+**), sss tokens may be forwarded when encrypted by the associated key. *Warning: forward-able sss tokens are inherently less secure.* See the notes for a detailed explanation.

-l *keylen*
is the byte-length of key to be added to the key table file via the **add** action. The default key length is 32 bytes (i.e., 256 bits). The *keylen* value should be between 4 and 128, inclusive. Otherwise, it is set to the minimum (maximum) value, as appropriate.

-n *keynum*
specifies an optional key number. The resulting effect is dependent on the *action*. See the notes for a detailed explanation.

-s {**c**|**g**|**k**|**n**|**u**|**x**}
sorts the output of the **list** action in ascending order as follows:

c - by creation date	**k** - by key name	**u** - by user name
g - by group name	**n** - by key number	**x** - by expiration date

-u *user* specifies an optional user name. The resulting effect is dependent on the *action*. See the notes for a detailed explanation.

-x *days* is the number of days that the key is to be valid. Specifying 0, the default, will never expire the key.

-x *mm/dd/yy*
is the date at which the key is to expire.

Notes

1) The **–g**, **–k**, -**n** and **–u** options modify the effects of the specified action. For **del**, **install**, and **list** actions, these options are used to narrow the set of keys to which the action applies. That is, only keys matching the specified options are deleted, installed, or listed.
2) For the **add** action, the **–g**, **–k**, and **–u** options are used to associated a group name, key name, and user name with the key. The following table lists the default names and special names.

Option	Default Name	Special Names
-g	nogroup	anygroup, usrgroup
-u	nobody	anybody

anygroup
allows the client using the key to specify[5] the actual group name. If the client omits the specification, the default name is used.

usrgroup
always sets the group list to null. This defers setting the user's groups until authorization time[6].

anybody
allows the client using the key to specify[7] the actual user name. If the client omits the specification, the default name is used.

3) Specifying any name other than a special name, prohibits the client from over-riding the name. The authenticated name is set to the specified name when the client successfully uses the associated key.
4) Warning: using special names *require* that you trust the client to specify a proper name. Special names do not confer any additional security.
5) When the *keyname* ends with a plus sign, an sss token encrypted by the associated key may be forwarded (i.e. used by a host different from the one that encrypted the sss token). Warning: *Allowing forwarded tokens makes it impossible to detect man-in-the-middle attacks or stolen sss tokens.* To maintain a high level of security, you should avoid making sss tokens forward-able whenever possible.

[5] The client does this as specified in the **XrdSecsssID.hh** file.
[6] The default authorization described in this reference uses the "userid" to look-up the associated groups as defined by Unix.
[7] The client does this as specified in the **XrdSecsssID.hh** file.

6) You need to make sss tokens forward-able for certain clients. For instance, clients who reside on a private network and tunnel through a Network Address Translation (NAT) device cannot use non-forward-able sss tokens. This is because the NAT device appears as a man-in-the-middle attacker to the sss protocol and the client's sss token will be rejected. Forward-able sss tokens avoid this problem.
7) For simple installations, you need not assign keys names. For more secure installations, you can use the *keyname* to easily install designated keys on designated hosts; limiting your exposure to intrusions. For instance,

```
xrdsssadmin –k io.slac.stanford.edu add /opt/xrootd/mytab
xrdsssadmin –k foo.hardvard.edu add /opt/xrootd/mytab
```

adds two keys to file **/opt/xrootd/mytab**, each with a unique name that corresponds to the host that should receive that key. You can now easily distribute the desired keys to each host by executing one of the following

```
grep io.slac.stanford.edu /opt/xrootd/mytab  \
   | ssh u1@io.slac.stanford.edu \
         xrdsssadmin install /opt/xrootd/mytab

egrep 'foo.harvard.edu|io.slac.stanford.edu' /opt/xrootd/mytab  \
   | ssh u1@border.slac.stanford.edu \
         xrdsssadmin install /opt/xrootd/mytab
```

2.4.6 **protocol** (unix)

```
sec.protocol [ libpath ] unix
```

Function

Define the characteristics of the **unix** authentication protocol.

Parameters

libpath

The absolute directory path where the protocol shared library exists. If an absolute path is not specified, the library must be on the search path defined by the **LD_LIBRARY_PATH** environmental variable.

Defaults

None.

Notes

1) ***Warning***: **unix** protocol does not provide any significant level of security and should only be used in instances where security violations do not matter.

Example

```
sec.protocol /usr/lib/xrd unix
```

2.5 protparm

```
sec.protparm pid parms
```

Function

Specify protocol parameters.

Parameters

pid The identifier of a yet-to-be-defined protocol with the **protocol** directive.

parms The parameters required by the protocol to operate successfully. The parameters are protocol dependent.

Defaults

See each protocol for its specific set of default parameters.

Notes

1) The **protparm** directive allows you to conveniently specify very long parameter strings. The final parameters are constructed in the order specified; with each specification (i.e., *parms*) separated by a new-line (\n) character. The first parameter string comes from the *parms* specification on the **protocol** directive.
2) The specified *pid* must not have been yet defined using the protocol directive.
3) Each specified *pid* must have a matching **protocol** directive following the last occurrence of **protparm** *pid*.

Example

```
sec.protparm krb5 /etc/krb5keys xrdserv
```

3 Default Authorization Configuration

3.1 audit

```
acc.audit parm [ parm ]

parm:    deny | grant | none
```

Function

Control the level of auditing.

Parameters

parm The level of auditing specify one or more of:

deny - audit access denials
grant - audit access approvals
none - turn off auditing

Defaults

```
acc.audit none
```

Notes

1) Audit parameters are cumulative. To enabled auditing of denials and approvals you must specify both **deny** and **grant** parameters.
2) The **none** parameter turns off auditing regardless of what was specified prior to the **none** parameter.
3) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.audit deny
```

3.2 authdb

acc.authdb *path*

Function

Specify the location of the authorization database.

Parameters

path The absolute path of the authorization database.

Defaults

acc.authdb /opt/xrd/etc/Authfile

Notes

1) You *must* specify the **authdb** directive together with the **ofs.authorize** directive in order to enable authorization.
2) The meaning of the specification depends on the mechanism used to obtain authorization information. The default implementation uses a flat file. You specify the name of this flat file using the **authdb** directive.
3) In order to maintain proper security, the authorization file must only be writable by the user running as **xrootd**. Some installation may also wish to restrict read access to the same user.
4) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

acc.authdb /opt/xrd/etc/AuthDB

3.3 authrefresh

```
acc.authrefresh seconds
```

Function

Control how frequently the authorization database is to be checked for modifications.

Parameters

seconds

The number seconds between checks for modifications.

Defaults

```
acc.authrefresh 43200
```

Notes

1) The **acc** component periodically checks to see if the authorization database has changed, If it has changed, **xrootd** rebuilds internal authorization information from the database. This means that the **authrefresh** directive determines the information currency.
2) Lower **authrefresh** *seconds* values increase **xrootd** overhead because internal information may need to be rebuilt more frequently.
3) Higher **authrefresh** *seconds* values decrease **xrootd** overhead. However, authorization information may be out of date for an excessively long period of time.
4) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.authrefresh 10800
```

3.4 gidlifetime

```
acc.gidlifetime seconds
```

Function

Controls how long group membership information may be cached in memory.

Parameters

seconds

The number seconds group membership information may be cached before it must be discarded and subsequently re-determined, as needed. The value may be suffixed by **s**, **h**, or **m** (the case is immaterial) to indicate that seconds (the default), minutes, or hours are being specified, respectively.

Defaults

```
acc.gidlifetime 12h
```

Notes

1) Because it is relatively expensive to determine a user's groups, **xrootd** caches the information in memory for a limited time so that subsequent requests for the user's group memberships can be quickly satisfied. Since the user's groups may change, the information is cached for a limited time. After the time interval expires, the membership information is discarded so that the next request for the user's groups will cause the information to be recomputed.
2) Lower **gidlifetime** *seconds* values increase **xrootd** overhead because group membership information may need to be rebuilt more frequently.
3) Higher **gidlifetime** *seconds* values decrease **xrootd** overhead. However, group membership information may be out of date for an excessively long period of time.
4) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.gidlifetime 4h
```

3.5 gidretran

```
acc.gidretran gid [ gid [ . . . ] ]
```

Function

Specify the Unix group ids that are aliases. See the notes on when to use this directive.

Parameters

gid A Unix group number that is an alias for some other group number.

Defaults

None.

Notes

1) The **gidretran** directive may be used by large installations running **NIS**[8] to disambiguate group names. Large installations may have a group membership list that is too large for a single group name entry. The typical solution is to break-up the membership list and assign it to two or more group names, each having the same group id number (**gid**). This leads to ambiguity as to which name should be used. The **gidretran** directive can record all such **gid**'s so that the authorization system knows when to retranslate a group name to its base name (e.g., the one shown via the **ls** command), as determined by the **getgrgid()** function.
2) Up to 128 group ids may be specified.
3) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.gidretran 10200 10201 10202
```

[8] **NIS+** does not have the problem described in this paragraph.

3.6 nisdomain

```
acc.nisdomain name
```

Function

Specify the domain name to use when determining membership in a **NIS** netgroup.

Parameters

name The **NIS** domain name to use.

Defaults

None. By default, a null domain name is used so that membership is solely determined based on the user's name and the originating host name.

Notes

1) Membership in a **NIS** netgroup is determine by three factors:
   a. User's originating host name,
   b. The user's actual name, and
   c. An arbitrary domain name.

   The factors are commonly referred to as the (machine,user,domain) triple. The **innetgr(**) function determines whether a triple is a member of a particular **netgroup**. When the domain name is not specified, then only the machine and user fields are used to determine membership in the a group, regardless of the domain specified in the **NIS netgroup** file.
2) Netgroup membership is only meaningful when the authorization database contains privileges based on **netgroup** name (i.e., **n** record type was specified).
3) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.nisdomain slacxrd
```

3.7 pgo

```
acc.pgo
```

Function

Specify how Unix group membership affects authorization.

Parameters

None.

Defaults

All of the groups in which a user is a member determine the user's privileges.

Notes

1) By default, all of a user's Unix groups determine the user's access privileges. This is the **POSIX** standard. Installations wishing **SVR5** behavior should specify the **pgo** directive. When **pgo** is specified, only the user's primary group determines the user's privileges.
2) The **pgo** directive can substantially reduce **xrootd** overhead when many users have very long group membership lists.
3) This directive is used by the default authorization scheme. Other authorization schemes may or may not honor this directive.

Example

```
acc.pgo
```

4 Authorization Database File

Authorization information may come from various sources, depending on the installation. The supplied implementation uses a flat Unix file to record authorization information. Regardless of the information source, the following semantics prevail.
An authorization database contains one or more authorization records.
Each authorization record is considered to be a capability.
Each capability is tied to a unique identifier within its name class.
An identifier may be a

- host name,
- domain name,
- netgroup name,
- template name,
- unix group name,
- user name

All template identifiers are logically processed in the order specified. The processing order of other identifiers is immaterial.
Each identifier is associated with an arbitrary list of path prefix-privilege pairs.
The list is always searched from left to right (i.e., in the order that it was specified).
The privileges associated with first prefix that matches an incoming path name are considered to be the applicable privileges.

Additionally, the flat file implementation allows for

- comments (i.e., any record whose first character is a pound sign, #),
- blank records, and
- continuations designated by a back slash (\) as the last non-blank character on the continued record.

4.1 Default Authorization Database Record

idtype id { {*path* | *objectid*} *privs* | *tname* } [{ {*path* | *objectid*} *privs* | *tname* } [•••]]

idtype: **g** | **h** | **n** | **t** | **u**

privs: *plets* | **-***plets* | *plets***-***plets*

plets: { **a** | **d** | **i** | **k** | **l** | **n** | **r** | **w** } [*plets*]

Where:

idtype
is a single letter indicating the type of identifier that follows. Valid type letters are:
g - group name **n** - netgroup name **u** - user name
h - host/domain name **t** - template name

id The actual identifier. Identifier must be consistent with their type. Additionally, host names must be fully qualified and specified in lower case. Should a host name start with a period, it is treated as a domain name. A domain name must match the right-most characters of a host name in order for the associated capability to be used.

objectid
The object identifier prefix to be used for matching purposes.

path The path prefix to be used for matching purposes.

privs The privileges associated with the preceding path. Privilege letters preceding a minus sign represent the privileges being granted. Privilege letters following the minus sign represent the privileges being denied. Privilege letters stand for:

a – all privileges
d – delete (i.e., remove)a file
i – insert (i.e., create) a file
k – lock a file (not used)
l – lookup a file (i.e., search directory)
n – rename a file
r – read a file
w – write a file

tname A previously defined (i.e., occurring earlier in the file) template name. The template's associated path-privs list is logically substituted for the template name.

Notes

1) Any number of database records (i.e., lines terminated by a new line, `\n`, character) may be specified.
2) A database record may be continued to another record by placing a back slash (`\`) character at the end of the record (i.e., last non-blank character). Continuations are useful for long specifications since you are not allowed to specify the same id within a type more than once.
3) Data records whose first character is a pound sign (#) are treated as comments.
4) Blank database records are ignored.
5) Only one particular *idtype-id* combination may appear in the authorization files. Therefore, you must specific all capabilities for an *idtype-id* in one record. Use continuation syntax to improve readability.
6) The path-privs and objected-privs entries are always matched from left to right. Therefore, specify paths and objectids from most significant to least significant order (i.e., all exceptions or longest paths first)
7) The authorization system does not perform multiple slash removal. Therefore a path prefix of `/foo//bar` and `/foo/bar`, while logically the same, are treated as two distinct specifications.
8) While prefix matching does not differentiate file system objects, paths ending with a slash logically indicate a directory. This type of specification works consistently with all operations except "stat" and "list". The reason is that these operations specify directory names without a trailing slash.
9) In order to maintain a file system object view in a path prefix matching model, the look-up privilege should be granted to all users for all path prefixes.
10) Certain privileges should be granted together. For instance, insert privileges should be granted along with delete privileges, and write with read.
11) The user record type is also used to convey default privileges and specified referential privileges. Refer to the following sections for more information.

Example

```
t base /fie l
u abh  /fie/foo/fum/ a /fie/foo/ rw base
```

A template named `base` has been defined. It provides for look-up privileges to any file system object whose path starts with `/fie`. In this examples, it also implies that the contents of the `fie` directory can be listed. Subsequently, user `abh` is granted all permissions for file system objects that start with `/fie/foo/fum/` (i.e., in directory `fum`), read-write permissions for all file system objects that start with `/fie/foo/` (i.e., in directory `foo`) and whatever privileges that are afforded by template `base`.

4.1.1 Default Privileges

Default privileges may be specified using the user record type, as follows.

u * { *path privs* | *tname* } [{ *path privs* | *tname* } [•••]]

The use of an asterisk as the user name indicates that the specified privileges are to apply to all users, regardless of their user name and location (subject to other applicable negative privileges). The default specification is useful for granting all users look-up privileges on the complete file space in order to maintain a file system view of authorization.

4.1.2 User Fungible Capabilities

Privileges may be granted to specific paths that encode the user's name without having to specify each such path for every user as follows:

u = { *path privs* | *tname* } [{ *path privs* | *tname* } [•••]]

Each path in **UFC** record should contain the character sequence "@=". The first such sequence indicates where the user's name should be substituted before a path prefix match is attempted. This allows you to provide for file system areas that are effectively "owned" by a user without needing to specify the actual user's name.

Example

```
u * /xrd lr
u = /xrd/users/@=/ a
```

All users, by default, have read and look-up access fo any file system object prefixed by `/xrd` (i.e., in directory `xrd`). However, users have all privileges for any file system object that is prefixed by `/xrd/users/`, followed by their user name, and ending with a slash (i.e., in a directory that corresponds to their user name).

5 Document Change History

1 June 2005

- Add the generalized **if** facility explanation.

18 July 2005

- Include GIS and PWD protocol information. Supplied by Gerri Ganis, CERN.

22 March 2006

- Add **exec** condition to **if/else/fi**.

23 May 2006

- Discuss the authorization plug-in and the **ofs.authlib** directive.

2 April 2007

- Move explanation of conditional directives to another manual.
- Some minor clean-up.

2 August 2007

- Add **unix** protocol description.
- Some minor clean-up.

8 January 2008

- General clean-up.

3 October 2008

- Add **sss** protocol description.

29 January 2009

- Add **usrgroup** option to the **sss** protocol.

7 April 2009

- Remove **loginid** option from the **sss** protocol.

25 November 2009

- Add **ssl** protocol description.

6 June 2011

- Make **xrdsssadmin** example for distributing keys more obvious.

27 September 2011

- Remove description of the **krb4** and **ssl** protocols.
- Fully describe all of the gsi security options.
- Describe the generic AuthzVO plug-in.

---------- Release 3.1.0

22 February 2012

- Add options 12 and 13 to "–crl:" to gsi security. Mentions these for envar XrdSecGSICRLCHECK.
- Add the "–crlrefresh:" option to gsi security.

---------- Release 3.1.1

6 April 2012

- Correct type: change "validvo" to "valido" as the VOMS way of screening valid organizations in x.509 certificates.

---------- Release 3.2.0
---------- Release 3.2.1
---------- Release 3.2.2
---------- Release 3.2.3
---------- Release 3.2.4

26 September 2012

- Document the **sslhashold** gsi security option.
- Document **vomsfun** and **vomsfunargs** gsi switches

---------- Release 3.2.5 to 3.2.7
---------- Release 3.3.0 to 3.3.2

7 February 2013

- Add link to external plug-in for VOMS extraction7 February 201315 October 20167 February 2013
- Remove description of **sslhashold** which was suppressed because an automatic mechanism to handle hash algorithms has been implemented
- Regenerate index and fix some typos

---------- Release 3.3.3 to 3.3.6
---------- Release 4.0.1 to 4.0.3

15 October 2014

- Explain how to create forward-able sss tokens using **xrdsssadmin**.

---------- Release 4.1.0 to 4.3.0
---------- Release 4.4.0

19 June 2016

- Explain how to do authorization based on objected.

---------- Release 4.5.0

9 October 2016

- Explain the **sec.level** directive.