CEPH-AUTHTOOL – CEPH KEYRING MANIPULATION TOOL

SYNOPSIS

ceph-authtool *keyringfile* [-l | –list] [-p | –print-key] [-C | –create-keyring] [-g | –gen-key] [–gen-print-key] [–import-keyring *otherkeyringfile*] [-n | –name *entityname*] [-u | –set-uid *auid*] [-a | –add-key *base64_key*] [–cap *subsystem capability*] [–caps *capfile*]

DESCRIPTION

ceph-authtool is a utility to create, view, and modify a Ceph keyring file. A keyring file stores one or more Ceph authentication keys and possibly an associated capability specification. Each key is associated with an entity name, of the form `{client,mon,mds,osd}.name`.

WARNING Ceph provides authentication and protection against man-in-the-middle attacks once secret keys are in place. However, data over the wire is not encrypted, which may include the messages used to configure said keys. The system is primarily intended to be used in trusted environments.

OPTIONS

`-l`, **`--list`**
will list all keys and capabilities present in the keyring

`-p`, **`--print-key`**
will print an encoded key for the specified entityname. This is suitable for the `mount -o secret=` argument

`-C`, **`--create-keyring`**
will create a new keyring, overwriting any existing keyringfile

`-g`, **`--gen-key`**
will generate a new secret key for the specified entityname

`--gen-print-key`
will generate a new secret key for the specified entityname, without altering the keyringfile, printing the secret to stdout

`--import-keyring` `*secondkeyringfile*`
will import the content of a given keyring to the keyringfile

`-n`, **`--name`** `*name*`
specify entityname to operate on

`-u`, **`--set-uid`** `*auid*`
sets the auid (authenticated user id) for the specified entityname

`-a`, **`--add-key`** `*base64_key*`
will add an encoded key to the keyring

`--cap` `*subsystem* *capability*`
will set the capability for given subsystem

`--caps` `*capsfile*`
will set all of capabilities associated with a given key, for all subsystems

CAPABILITIES

The subsystem is the name of a Ceph subsystem: `mon`, `mds`, or `osd`.

The capability is a string describing what the given user is allowed to do. This takes the form of a comma separated list of allow clauses with a permission specifier containing one or more of rwx for read, write, and execute permission. The `allow *` grants

full superuser permissions for the given subsystem.

For example:

```
# can read, write, and execute objects
osd = "allow rwx"

# can access mds server
mds = "allow"

# can modify cluster state (i.e., is a server daemon)
mon = "allow rwx"
```

A librados user restricted to a single pool might look like:

```
mon = "allow r"

osd = "allow rw pool foo"
```

A client using rbd with read access to one pool and read/write access to another:

```
mon = "allow r"

osd = "allow class-read object_prefix rbd_children, allow pool templates r class-read, allow
```

A client mounting the file system with minimal permissions would need caps like:

```
mds = "allow"

osd = "allow rw pool data"

mon = "allow r"
```

OSD CAPABILITIES

In general, an osd capability follows the grammar:

```
osdcap  := grant[,grant...]
grant   := allow (match capspec | capspec match)
match   := [ pool[=]<poolname> | object_prefix <prefix>
           | namespace[=]<rados-namespace>
           | tag <application-name> <key>=<value> ]
capspec := * | [r][w][x] [class-read] [class-write]
```

The capspec determines what kind of operations the entity can perform:

```
r           = read access to objects
w           = write access to objects
x           = can call any class method (same as class-read class-write)
class-read  = can call class methods that are reads
class-write = can call class methods that are writes
* or "all"  = equivalent to rwx, plus the ability to run osd admin commands,
               i.e. ceph osd tell ...
```

The match criteria restrict a grant based on the pool being accessed. Grants are additive if the client fulfills the match condition. For example, if a client has the osd capabilities: “allow r object_prefix prefix, allow w pool foo, allow x pool bar”, then it has rw access to pool foo, rx access to pool bar, and r access to objects whose names begin with ‘prefix’ in any pool.

CAPS FILE FORMAT

The caps file format consists of zero or more key/value pairs, one per line. The key and value are separated by an =, and the value must be quoted (with ' or ") if it contains any whitespace. The key is the name of the Ceph subsystem (osd, mds, mon), and the value is the capability string (see above).

EXAMPLE

To create a new keyring containing a key for client.foo:

```
ceph-authtool -C -n client.foo --gen-key keyring
```

To associate some capabilities with the key (namely, the ability to mount a Ceph filesystem):

```
ceph-authtool -n client.foo --cap mds 'allow' --cap osd 'allow rw pool=data' --cap mon 'allow
```

To display the contents of the keyring:

```
ceph-authtool -l keyring
```

When mounting a Ceph file system, you can grab the appropriately encoded secret key with:

```
mount -t ceph serverhost:/ mountpoint -o name=foo,secret=`ceph-authtool -p -n client.foo keyr
```

AVAILABILITY

ceph-authtool is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

SEE ALSO

ceph(8)