kAFS: AFS FILESYSTEM

Overview

This filesystem provides a fairly simple secure AFS filesystem driver. It is under development and does not yet provide the full feature set. The features it does support include:

(*) Security (currently only AFS kaserver and KerberosIV tickets).

(*) File reading and writing.

(*) Automounting.

(*) Local caching (via fscache).

It does not yet support the following AFS features:

(*) pioctl() system call.

Compilation

The filesystem should be enabled by turning on the kernel configuration options:

```
CONFIG_AF_RXRPC         - The RxRPC protocol transport
CONFIG_RXKAD            - The RxRPC Kerberos security handler
CONFIG_AFS              - The AFS filesystem
```

Additionally, the following can be turned on to aid debugging:

```
CONFIG_AF_RXRPC_DEBUG   - Permit AF_RXRPC debugging to be enabled
CONFIG_AFS_DEBUG        - Permit AFS debugging to be enabled
```

They permit the debugging messages to be turned on dynamically by manipulating the masks in the following files:

```
/sys/module/af_rxrpc/parameters/debug
/sys/module/kafs/parameters/debug
```

Usage

When inserting the driver modules the root cell must be specified along with a list of volume location server IP addresses:

```
modprobe rxrpc
modprobe kafs rootcell=cambridge.redhat.com:172.16.18.73:172.16.18.91
```

The first module is the AF_RXRPC network protocol driver. This provides the RxRPC remote operation protocol and may also be accessed from userspace. See:

Documentation/networking/rxrpc.rst

The second module is the kerberos RxRPC security driver, and the third module is the actual filesystem driver for the AFS filesystem.

Once the module has been loaded, more modules can be added by the following procedure:

```
echo add grand.central.org 18.9.48.14:128.2.203.61:130.237.48.87 >/proc/fs/afs/cells
```

Where the parameters to the "add" command are the name of a cell and a list of volume location servers within that cell, with the latter separated by colons.

Filesystems can be mounted anywhere by commands similar to the following:

```
mount -t afs "%cambridge.redhat.com:root.afs." /afs
mount -t afs "#cambridge.redhat.com:root.cell." /afs/cambridge
mount -t afs "#root.afs." /afs
mount -t afs "#root.cell." /afs/cambridge
```

Where the initial character is either a hash or a percent symbol depending on whether you definitely want a R/W volume (percent) or whether you'd prefer a R/O volume, but are willing to use a R/W volume instead (hash).

The name of the volume can be suffixes with ".backup" or ".readonly" to specify connection to only volumes of those types.

The name of the cell is optional, and if not given during a mount, then the named volume will be looked up in the cell specified during modprobe.

Additional cells can be added through /proc (see later section).

Mount

Mountpoints

AFS has a concept of mountpoints. In AFS terms, these are specially formatted symbolic links (of the same form as the "device name" passed to mount). kAFS presents these to the user as directories that have a follow-link capability (i.e.: symbolic link semantics). If anyone attempts to access them, they will automatically cause the target volume to be mounted (if possible) on that site.

Automatically mounted filesystems will be automatically unmounted approximately twenty minutes after they were last used. Alternatively they can be unmounted directly with the umount() system call.

Manually unmounting an AFS volume will cause any idle submounts upon it to be culled first. If all are culled, then the requested volume will also be unmounted, otherwise error EBUSY will be returned.

This can be used by the administrator to attempt to unmount the whole AFS tree mounted on /afs in one go by doing:

```
umount /afs
```

Dynamic Root

A mount option is available to create a serverless mount that is only usable for dynamic lookup. Creating such a mount can be done by, for example:

```
mount -t afs none /afs -o dyn
```

This creates a mount that just has an empty directory at the root. Attempting to look up a name in this directory will cause a mountpoint to be created that looks up a cell of the same name, for example:

```
ls /afs/grand.central.org/
```

Proc Filesystem

The AFS module creates a "/proc/fs/afs/" directory and populates it:

(*) A "cells" file that lists cells currently known to the afs module and

their usage counts:

```
[root@andromeda ~]# cat /proc/fs/afs/cells
USE NAME
  3 cambridge.redhat.com
```

(*) A directory per cell that contains files that list volume location

servers, volumes, and active servers known within that cell:

```
[root@andromeda ~]# cat /proc/fs/afs/cambridge.redhat.com/servers
USE ADDR            STATE
  4 172.16.18.91        0
[root@andromeda ~]# cat /proc/fs/afs/cambridge.redhat.com/vlservers
ADDRESS
172.16.18.91
[root@andromeda ~]# cat /proc/fs/afs/cambridge.redhat.com/volumes
USE STT VLID[0]  VLID[1]  VLID[2]  NAME
  1 Val 20000000 20000001 20000002 root.afs
```

The Cell Database

The filesystem maintains an internal database of all the cells it knows and the IP addresses of the volume location servers for those cells. The cell to which the system belongs is added to the database when modprobe is performed by the "rootcell=" argument or, if compiled in, using a "kafs.rootcell=" argument on the kernel command line.

Further cells can be added by commands similar to the following:

```
echo add CELLNAME VLADDR[:VLADDR][:VLADDR]... >/proc/fs/afs/cells
echo add grand.central.org 18.9.48.14:128.2.203.61:130.237.48.87 >/proc/fs/afs/cells
```

No other cell database operations are available at this time.

Security

Secure operations are initiated by acquiring a key using the klog program. A very primitive klog program is available at:

https://people.redhat.com/~dhowells/rxrpc/klog.c

This should be compiled by:

```
make klog LDLIBS="-lcrypto -lcrypt -lkrb4 -lkeyutils"
```

And then run as:

```
./klog
```

Assuming it's successful, this adds a key of type RxRPC, named for the service and cell, e.g.: "afs@<cellname>". This can be viewed with the keyctl program or by cat'ing /proc/keys:

```
[root@andromeda ~]# keyctl show
Session Keyring
       -3 --alswrv      0     0  keyring: _ses.3268
        2 --alswrv      0     0   \_ keyring: _uid.0
111416553 --als--v      0     0   \_ rxrpc: afs@CAMBRIDGE.REDHAT.COM
```

Currently the username, realm, password and proposed ticket lifetime are compiled into the program.

It is not required to acquire a key before using AFS facilities, but if one is not acquired then all operations will be governed by the anonymous user parts of the ACLs.

If a key is acquired, then all AFS operations, including mounts and automounts, made by a possessor of that key will be secured with that key.

If a file is opened with a particular key and then the file descriptor is passed to a process that doesn't have that key (perhaps over an AF_UNIX socket), then the operations on the file will be made with key that was used to open the file.

The @sys Substitution

The list of up to 16 @sys substitutions for the current network namespace can be configured by writing a list to /proc/fs/afs/sysname:

```
[root@andromeda ~]# echo foo amd64_linux_26 >/proc/fs/afs/sysname
```

or cleared entirely by writing an empty list:

```
[root@andromeda ~]# echo >/proc/fs/afs/sysname
```

The current list for current network namespace can be retrieved by:

```
[root@andromeda ~]# cat /proc/fs/afs/sysname
foo
amd64_linux_26
```

When @sys is being substituted for, each element of the list is tried in the order given.

By default, the list will contain one item that conforms to the pattern "<arch>_linux_26", amd64 being the name for x86_64.