NAME

snap-confine − internal tool for confining snappy applications

SYNOPSIS

snap−confine [−−classic] [−−base BASE] SECURITY_TAG COMMAND [...ARGUMENTS]

DESCRIPTION

The *snap−confine* is a program used internally by *snapd* to construct the execution environment for snap applications.

OPTIONS

The *snap−confine* program accepts two options:

−−*classic* requests the so−called _classic_ _confinement_ in which applications are not confined at all (like in classic systems, hence the name). This disables the use of a dedicated, per−snap mount namespace. The *snapd* service generates permissive apparmor and seccomp profiles that allow everything.

−−*base BASE* directs snap−confine to use the given base snap as the root filesystem. If omitted it defaults to the *core* snap. This is derived from snap meta−data by *snapd* when starting the application process.

FEATURES

Apparmor profiles

snap−confine switches to the apparmor profile *$SECURITY_TAG*. The profile is **mandatory** and *snap−confine* will refuse to run without it.

The profile has to be loaded into the kernel prior to using *snap−confine*. Typically this is arranged for by *snapd*. The profile contains rich description of what the application process is allowed to do, this includes system calls, file paths, access patterns, linux capabilities, etc. The apparmor profile can also do extensive dbus mediation. Refer to apparmor documentation for more details.

Seccomp profiles

snap−confine looks for the */var/lib/snapd/seccomp/bpf/$SECURITY_TAG.bin* file. This file is **mandatory** and *snap−confine* will refuse to run without it. This file contains the seccomp bpf binary program that is loaded into the kernel by snap−confine.

The file is generated with the */usr/lib/snapd/snap−seccomp* compiler from the *$SECURITY_TAG.src* file that uses a custom syntax that describes the set of allowed system calls and optionally their arguments. The profile is then used to confine the started application.

As a security precaution disallowed system calls cause the started application executable to be killed by the kernel. In the future this restriction may be lifted to return *EPERM* instead.

Mount profiles

snap−confine uses a helper process, *snap−update−ns*, to apply the mount namespace profile to freshly constructed mount namespace. That tool looks for the */var/lib/snapd/mount/snap.$SNAP_NAME.fstab* file. If present it is read, parsed and treated like a mostly−typical *fstab(5)* file. The mount directives listed there are executed in order. All directives must succeed as any failure will abort execution.

By default all mount entries start with the following flags: *bind*, *ro*, *nodev*, *nosuid*. Some of those flags can be reversed by an appropriate option (e.g. *rw* can cause the mount point to be writable).

Certain additional features are enabled and conveyed through the use of mount options prefixed with *x−snapd−*.

As a security precaution only *bind* mounts are supported at this time.

Sharing of the mount namespace

As of version 1.0.41 all the applications from the same snap will share the same mount namespace. Applications from different snaps continue to use separate mount namespaces.

ENVIRONMENT

snap−confine responds to the following environment variables

SNAP_CONFINE_DEBUG:
When defined the program will print additional diagnostic information about the actions being performed. All the output goes to stderr.

The following variables are only used when *snap−confine* is not setuid root. This is only applicable when testing the program itself.

SNAPPY_LAUNCHER_INSIDE_TESTS:
Internal variable that should not be relied upon.

SNAP_CONFINE_NO_ROOT:
Internal variable that should not be relied upon.

SNAPPY_LAUNCHER_SECCOMP_PROFILE_DIR:
Internal variable that should not be relied upon.

SNAP_USER_DATA:
Full path to the directory like /home/$LOGNAME/snap/$SNAP_NAME/$SNAP_REVISION.

This directory is created by snap−confine on startup. This is a temporary feature that will be merged into snapd's snap−run command. The set of directories that can be created is confined with apparmor.

FILES

snap−confine and *snap−update−ns* use the following files:

/var/lib/snapd/mount/snap..fstab*:
Description of the mount profile.

/var/lib/snapd/seccomp/bpf/.src*:
Input for the /usr/lib/snapd/snap−seccomp profile compiler.

/var/lib/snapd/seccomp/bpf/.bin*:
Compiled seccomp bpf profile programs.

/run/snapd/ns/:
Directory used to keep shared mount namespaces.

snap−confine internally converts this directory to a private bind mount. Semantically the behavior is identical to the following mount commands:

mount −−bind /run/snapd/ns /run/snapd/ns mount −−make−private /run/snapd/ns

/run/snapd/ns/.lock:
A *flock(2)*−based lock file acquired to create and convert */run/snapd/ns/* to a private bind mount.

/run/snapd/ns/$SNAP_NAME.lock:
A *flock(2)*−based lock file acquired to create or join the mount namespace represented as */run/snaps/ns/$SNAP_NAME.mnt*.

/run/snapd/ns/$SNAP_NAME.mnt:
This file can be either:

- An empty file that may be seen before the mount namespace is preserved or when the mount namespace is unmounted.

- A file belonging to the *nsfs* file system, representing a fully populated mount namespace of a given snap. The file is bind mounted from */proc/self/ns/mnt* from the first process in any snap.

/proc/self/mountinfo:
This file is read to decide if */run/snapd/ns/* needs to be created and converted to a private bind mount, as described above.

Note that the apparmor profile is external to *snap−confine* and is loaded directly into the kernel. The actual apparmor profile is managed by *snapd*.

BUGS

Please report all bugs with *https://bugs.launchpad.net/snap−confine/+filebug*

AUTHOR

zygmunt.krynicki@canonical.com

COPYRIGHT