pylibacl Documentation

Release 0.5.4

Iustin Pop

Jun 19, 2020

Contents

See the README for start, or the detailed *module* information.

CHAPTER 1

Contents

toctree contains reference to nonexisting document u'readme'

toctree contains reference to nonexisting document u'news'

1.1 Implementation details

1.1.1 Functionality level

The IEEE 1003.1e draft 17 ("POSIX.1e") describes a set of 28 functions. These are grouped into three groups, based on their portability:

- first group, the most portable one. All systems which claim to support POSIX.1e should implement these:

  acl_delete_def_file(3), acl_dup(3), acl_free(3), acl_from_text(3), acl_get_fd(3), acl_get_file(3), acl_init(3), acl_set_fd(3), acl_set_file(3), acl_to_text(3), acl_valid(3)

- second group, containing the rest of the POSIX ACL functions. Systems which claim to fully implement POSIX.1e should implement these:

  acl_add_perm(3), acl_calc_mask(3), acl_clear_perms(3), acl_copy_entry(3), acl_copy_ext(3), acl_copy_int(3), acl_create_entry(3), acl_delete_entry(3), acl_delete_perm(3), acl_get_entry(3), acl_get_permset(3), acl_get_qualifier(3), acl_get_tag_type(3), acl_set_permset(3), acl_set_qualifier(3), acl_set_tag_type(3), acl_size(3)

- third group, containing extra functions implemented by each OS. These are non-portable version. Both Linux and FreeBSD implement some extra functions.

Thus we have the level of compliance. Depending on whether the system library support the second group, you get some extra methods for the ACL object.

The implementation of the second group of function can be tested by checking the module-level constant HAS_ACL_ENTRY. The extra functionality available on Linux can be tested by additional HAS_* constants.

1.1.2 Internal structure

The POSIX draft has the following stuff (correct me if I'm wrong):

- an ACL is denoted by acl_t
- an ACL contains many acl_entry_t, these are the individual entries in the list; they always(!) belong to an acl_t
- each entry_t has a qualifier (think uid_t or gid_t), whose type is denoted by the acl_tag_t type, and an acl_permset_t
- the acl_permset_t can contain acl_perm_t value (ACL_READ, ACL_WRITE, ACL_EXECUTE, ACL_ADD, ACL_DELETE, ...)
- functions to manipulate all these, and functions to manipulate files

1.1.3 Currently supported platforms

For any other platforms, volunteers are welcome.

Linux

It needs kernel 2.4 or higher and the libacl library installed (with development headers, if installing from rpm). This library is available on all modern distributions.

The level of compliance is level 2 (see IMPLEMENTATION), plus some extra functions; and as my development is done on Linux, I try to implement these extensions when it makes sense.

FreeBSD

The current tested version is 7.0. FreeBSD supports all the standards functions, but 7.0-RELEASE seems to have some issues regarding the acl_valid() function when the qualifier of an ACL_USER or ACL_GROUP entry is the same as the current uid. By my interpretation, this should be a valid ACL, but FreeBSD declares the ACL invalid. As such, some unittests fail on FreeBSD.

1.1.4 Porting to other platforms

First, determine if your OS supports the full 28 functions of the POSIX.1e draft (if so, define HAVE_LEVEL2) or only the first 11 functions (most common case, meaning only HAVE_LEVEL1).

If your OS supports only LEVEL1, modify `setup.py` as appropriately; unfortunately, the functionality of the module is quite low.

If your OS supports LEVEL2, there is a function which you must define: testing if an acl_permset_t contains a given permission. For example, under Linux, the acl library defines:

```
int acl_get_perm(acl_permset_t permset_d, acl_perm_t perm);
```

under FreeBSD, the library defines `acl_get_perm_np` with a similar syntax. So just see how this is implemented in your platform and either define a simple macro or a full function with the syntax:

```
static int get_perm(acl_permset_t permset_d, acl_perm_t perm);
```

which must return 1 if the permset contains perm and 0 otherwise.

Also see the search.