

Pluggable Access Control Mechanism

API Specification

Richard Offer
Silicon Graphics, Inc.

offer@sgi.com

This report attempts to expand on the paper *Rationale and Design for a Pluggable Access Control Interface*, and includes details of the proposed architecture and full descriptions of the API.

Table of Contents

Application APIs.....	3
Module APIs.....	8
A. Guide to Hacking.....	9

Application APIs

pacm_init

Name

`pacm_init` — Initialize a PACM application.

Synopsis

```
#include <pacm/pacm.h>
pacm_t pacm_init(const char * id);
```

Purpose

Initialize the library and read the file corresponding to *module*. Configurations files are normally stored in `/etc/pacm.d/`, but this may be overridden at compile time.

The function returns an opaque pointer (or `NULL` on error) that will be passed to all subsequent calls. Once `pacm_init()` has returned (successfully), the policy file will not be accessed again for this process.

Rationale

Accessing the policy file only once allows for the changing of policies while not affecting long running processes.

Additional Information

module should be a static string and not anything from `argv` or the environment.

pacm_close

Name

`pacm_close` — End a PACM session.

Synopsis

```
#include <pacm/pacm.h>
void pacm_close(pacm_t handle);
```

Purpose

Close the library, frees all the memory that the library has allocated.

No further PACM calls are permitted once this function has been called without first re-calling `pacm_init()`.

pacm_error

Name

`pacm_close` — End a PACM session.

Synopsis

```
#include <pacm/pacm.h>
int pacm_errno(pacm_t handle);
char * pacm_error(pacm_t handle);
```

Purpose

Obtain details on the last error inside PACM.

`pacm_errno()` returns the last `errno` generated by a PACM call. `pacm_error()` will return a string containing a description of the error encountered.

pacm_get_...

Name

`pacm_get_process` — Get security relevant attributes.

Synopsis

```
#include <pacm/pacm.h>
pacm_attr_t pacm_get_process(pacm_t handle, pid_t pid);
pacm_attr_t pacm_get_file(pacm_t handle, const char * path);
pacm_attr_t pacm_get_fd(pacm_t handle, int fd);
pacm_attr_t pacm_get_socket(pacm_t handle, pid_t sfd);
pacm_attr_t pacm_get_ipc(pacm_t handle, void * ipc);
```

Purpose

Obtain the attributes for a process, file, socket or System V IPC object.

If *pid* is 0, then obtain the current processes attributes.

These functions will return `NULL` on error. Information on the cause of the error maybe obtained from `pacm_error()`.

<i>what</i>	Common Unix™ concept
PACM_INDIVIDUAL	uid
PACM_FAMILIY	gid
PACM_CLUB	supplimentary groups
PACM_CLEARANCE	Role or label
PACM_IMPORTANCE	privilege

<i>function</i>	Description
PACM_IDENTITY	only call the PACM modules that are flagged as "identity".
PACM_PRIVILEGE	only call the PACM modules that are flagged as "privilige"
PACM_ALL	call all modules

pacm_clone

Name

`pacm_clone` — Generate a new set of security relevent attributes from two other sets.

Synopsis

```
#include <pacm/pacm.h>
pacm_attr_t pacm_clone(pacm_t handle, pacm_attr_t mother, long
m_genes, pacm_attr_t father, long r_genes);
```

Purpose

Generate a new `pacm_attr_t` by combining *m_genes* from *mother* and *f_genes* from *father*.

<i>m_genes / f_genes</i>	Common Unix™ concept
PACM_INDIVIDUAL	uid
PACM_FAMILIY	gid
PACM_CLUB	supplimentary groups

<i>m_genes / f_genes</i>	Common Unix™ concept
PACM_CLEARANCE	Role or label
PACM_IMPORTANCE	privilege

pacm_get_attr

Name

pacm_get_attr — Extract values from an attribute.

Synopsis

```
#include <pacm/pacm.h>
int pacm_get_attr(pacm_attr_t subj, long what, const char * fmt ...);
```

Purpose

Extract *what* from the *subj* and return it in a format specified in *fmt*.

<i>what</i>	Common Unix™ concept
PACM_INDIVIDUAL	uid
PACM_FAMILY	gid
PACM_CLUB	supplimentary groups
PACM_CLEARANCE	Role or label
PACM_IMPORTANCE	privilege

fmt is a format string where the following format identifiers are recognised.

<i>fmt</i>	ISO C99 type	<i>fmt</i>	ISO C99 type
d	int16_t	ud	u_int16_t
i	int32_t	ui	u_int32_t
l	int64_t	ul	u_int64_t
s	char *	hs	char *
p#	void *, size_t	[items]	array of type items, number of items in array

An example would be

```
int32_t uid, *grps;
char *username, *groups;

(void) pacm_get_attr(attr, PACM_INDIVIDUAL, "is", &uid, &username);
(void) pacm_get_attr(attr, PACM_CLUB, "[i]hs", &grps, &ngrps, &groups);
```

In the above example, the difference between `[i]` and `hs` is the `[i]` should return an array containing all of the group ids in the attribute, whereas `hs` would probably return a string containing a reasonable human readable output such as `10001(trust),100(users),111(cvs)`.

All modules must present human readable output when asked for a `hs` format (even if the data is held in an array).

For example, obtaining the `PACM_INDIVIDUAL` data as a integer format should result in the uid being returned, asking for it in string format should return the username.

If the list of PACM modules called returns multiple values for a given request, the information will be presented as an array of values. The order presented maynot be the same order that the modules were called under, nor will it be possible for an application to determine what module filled in which piece of data.

If this method of parameter passing looks fammiliar, it is based on the Python methodology for communicating between a C extension and the Python object model.

pacm_cando

Name

`pacm_cando` — Make a policy decision.

Synopsis

```
#include <pacm/pacm.h>
int pacm_cando(pacm_attr_t subject, pacm_attr_t object, pacm_action_t do_what);
```

Purpose

Note: To Do

<i>do_what</i>	Common Unix™ concept
<code>PACM_READ</code>	Read the object
<code>PACM_WRITE_APPEND</code>	Append to the end of the object
<code>PACM_WRITE</code>	Write to the object. This is shorthand for <code>PACM_DELETE PACM_CREATE PACM_WRITE_APPEND</code>
<code>PACM_CREATE</code>	Create the object if it doesn't already exist

<i>do_what</i>	Common Unix™ concept
PACM_DELETE	Delete the object.
PACM_SIGNAL	This is an alias for PACM_WRITE_APPEND.
PACM_EXECUTE	Execute (<code>exec()</code>) the object.

Note: This table in particular is just a place-marker, I need to identify finer granularity for actions.

Module APIs

Registering a Module

To register a module to the PACM library, a module should write an initialization function, and declare it with the macro `PACM_INIT_MODULE()`. The reason for this indirection is to allow for the possibility of linking a module directly into an application.

```
/* must define the module name before including the header file */
#define PACM_MODULE pacm_unix

#include <pacm/pacm_module.h>

PACM_INIT_MODULE(pacm_module_t * module, int argc, char ** argv)
{
    /* do stuff */

    return 0;
}
```

Intermodule Communication

To allow for modules to communicate to each other, PACM copies PAM in using keyword=value options to be defined in a module.

Both intra-module and inter-module communication is supported, in the first case through the api

```
void * pacm_get_module_value(pacm_module_t * mod, const char * key);
int pacm_set_module_value(pacm_module_t * mod, const char * key, void
* value);
```

The latter case through the api.

```
void * pacm_get_value(pacm_t handle, const char * key);
int pacm_set_value(pacm_t handle, const char * key, void * value);
```

Note: This API should be re-examined in the light of the new attribute handling API.

Attribute Handling

To store a value into an attribute, `pacm_store_attr()` should be used.

```
#include <pacm/pacm.h>
int pacm_store_attr(pacm_attr_t attr, long what, const char * fmt
...);
```

Predefined Types

The following *fmt* flags are predefined.

<i>fmt</i>	ISO C99 type	<i>fmt</i>	ISO C99 type
d	int16_t	ud	u_int16_t
i	int32_t	ui	u_int32_t
l	int64_t	ul	u_int64_t
s	char *	hs	char *
p#	void *, size_t	[items]	array of typeitems, number of items in array

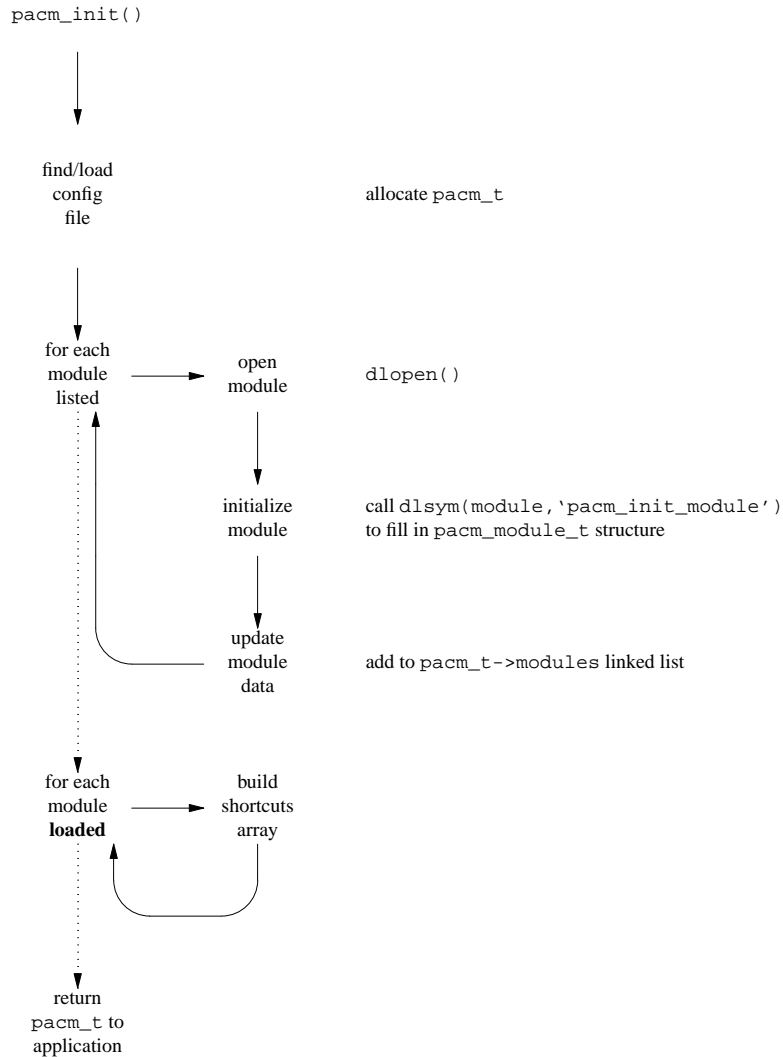
Argument Handling

A. Guide to Hacking

Internal Architecture

This section is a brief outline of the internal structure of the library and modules, its of no use unless you are developing PACM or a PACM module.

Library Initialization



Data Structures

There are three key data structures to PACM, the first two `pacm_t` and `pacm_attr_t` are used by applications (although they are opaque), the third, `pacm_module_t` is the interface between the PACM library and a policy module.

`pacm_t`

The `pacm_t` is the application's handle on a policy. It is returned by `pacm_init()`, and can be deleted by `pacm_close()`.

`pacm_t` is opaque, it cannot be examined by an application, as far as an application is concerned it is simply a `void *` pointer (this is what is defined as in `pacm/pacm.h`), see `pacm_rec` for the full (internal) details..

`pacm_attr_t`

The `pacm_attr_t` is the application's handle on an attribute. It is returned by one of the `pacm_get...()` APIs, and can be deleted by `pacm_attr_delete()`.

`pacm_attr_t` is opaque, it cannot be examined by an application, as far as an application is concerned it is simply a `void *` pointer (this is what is defined as in `pacm/pacm.h`), see `pacm_attr_rec` for the full (internal) details..

pacm_module_t

The `pacm_module_t` is the means by which a module registers the functionality it provides to the PACM library. The memory allocation is handled by the library, a pointer to a pre-allocated region is passed to the modules `pacm_init_module()` where it should be filled in.

`pacm_module_t` is a pointer to a structure that will have at least the following members. `argc` and `argv` allow a module to access any additional options that the site administrator may have added to the configuration line.

`const char * _module_file`

The (fully qualified) filename of the module, this is filled in by the PACM library, not by the module. This is intended for debugging purposed only, if the library is compiled with debugging disabled, this will be `NULL`.

`_proc_get_func _get_proc()`

The function to call to obtain the attributes for a process. `_proc_get_func` is defined as

```
pacm_attr_t (*_proc_get_func)(pacm_module_t * mod,
pid_t pid, int argc, char ** argv);
```

`_fd_get_func _get_fd()`

The function to call to obtain the attributes for a file descriptor. `_fd_get_func` is defined as

```
pacm_attr_t * (*_fd_get_func)(pacm_module_t * mod,
int fd, int argc, char ** argv);
```

`_file_get_func _get_file()`

The function to call to obtain the attributes for a file name. `_file_get_func` is defined as

```
pacm_attr_t * (*_file_get_func)(pacm_module_t * mod,
const char * path, int argc, char ** argv);
```

`_socket_get_func _get_socket()`

The function to call to obtain the attributes for a socket. `_socket_get_func` is defined as

```
pacm_attr_t * (*_socket_get_func)(pacm_module_t * mod,
int sfd, int argc, char ** argv);
```

`_ipc_get_func _get_ipc()`

The function to call to obtain the attributes for a System V IPC object. `_ipc_get_func` is defined as

```
pacm_attr_t * (*_ipc_get_func)(pacm_module_t * mod,
void * ipc, int argc, char ** argv);
```

`_proc_set_func _set_proc()`

The function to call to set the attributes for a process. `_proc_set_func` is defined as

```
int (*_proc_set_func)(pacm_module_t * mod,
pid_t pid, pacm_attr_t attr, int argc, char ** argv);
```

`_fd_set_func _set_fd()`

The function to call to set the attributes for a file descriptor. `_fd_set_func` is defined as

```
int (*_fd_set_func)(pacm_module_t * mod,
int fd, pacm_attr_t attrs, int argc, char ** argv);
```

`_socket_set_func _set_socket()`

The function to call to set the attributes for a socket. `_socket_set_func` is defined as

```
int (*_socket_set_func)(pacm_module_t * mod,
int sfd, pacm_attr_t attr, int argc, char ** argv);
```

`_ipc_set_func _set_ipc()`

The function to call to set the attributes for an System V IPC object. `_ipc_set_func` is defined as

```
int (*_ipc_set_func)(pacm_module_t * mod,
void * ipc, pacm_attr_t attr, int argc, char ** argv);
```

Additional members are present that allow for walking through the linked list, and accessing the `pacm_t`.

pacm_rec

`pacm_rec` is the structure behind the opaque type `pacm_t`. All library calls only accept `pacm_t`, they then typecast that to a `pacm_rec` before processing can continue.

At least the following members are present.

`const char * _application`

The name that `pacm_init()` was registered with.

`pacm_module_t * _modules`

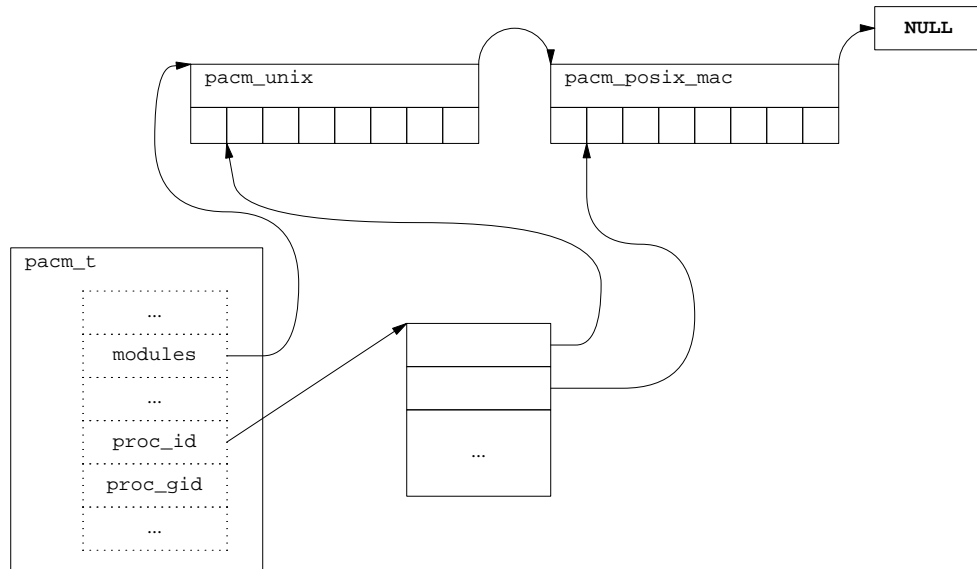
The head of the linked list of modules that have been registered for this application.

There is also one member for each type registered in the `pacm_module_t`, ie.

`_proc_get_func * _proc_get_funcs`

A null terminated list of all the `_get_proc` functions that have been registered in all the modules.

This is a performance improvement as a means of quickly calling all the functions registered for a particular action.



pacm_attr_rec

`pacm_attr_rec` is the structure behind the opaque type `pacm_attr_t`. All library and module calls only accept `pacm_attr_t`, typecasting as needed.

Attributes are multipart (much as a multi-part MIME document), in that they can contain multiple representations of the same piece of data. If multiple attributes are present, the library will merge them into a single multi-valued attribute before giving the application access to the attribute.

Allocation and Deallocation

A `pacm_attr_rec` should be allocated via `pacm_alloc_attr()` and freed with `pacm_free_attr()`. Failure to free with `pacm_free_attr()` will lead to memory leaks in cases where the attribute is multi-part.

Creating New Types

How the hell do we do this ? Should we ?

Header Files

There are two public header files `pacm/pacm.h` and `pacm/pacm_module.h`. A private header file, `pacm/pacm_private.h` provides the internal structures.

pacm/pacm_private.h

`pacm/pacm_private.h` provides the full declarations of the structures that `pacm_t` and `pacm_attr_t` are cast to internally.

