NAME

cap_new_launcher, cap_func_launcher, cap_launcher_callback, cap_launcher_set_mode, cap_launcher_set_iab, cap_launcher_set_chroot, cap_launch, cap_launcher_setuid, cap_launcher_setgroups − libcap launch functionality

SYNOPSYS

```
#include <sys/capability.h>

cap_launch_t cap_new_launcher(const char *arg0, const char *const *argv,
   const char *const *envp);

cap_launch_t cap_func_launcher(int (callback_fn)(void *detail));

int cap_launcher_callback(cap_launch_t attr,
   int (callback_fn)(void *detail));
int cap_launcher_set_mode(cap_launch_t attr, cap_mode_t flavor);
cap_iab_t cap_launcher_set_iab(cap_launch_t attr, cap_iab_t iab);
int cap_launcher_set_chroot(cap_launch_t attr, const char *chroot);

#include <sys/types.h>

pid_t cap_launch(cap_launch_t attr, void *detail);
int cap_launcher_setuid(cap_launch_t attr, uid_t uid);
int cap_launcher_setgroups(cap_launch_t attr, gid_t gid,
   int ngroups, const gid_t *groups);
```

Link with *−lcap*.

DESCRIPTION

A launcher provides a mechanism for code to execute a callback function and/or a program executable in an environment with a modified security context. Essentially it provides a mechanism for a program to **fork**(2) a new context from that of the main program manipulate capability and other privileged state in that **fork**(2)d process before (optionally) **execve**(2)ing a new program. When the application links to *−lpsx* this support is needed to robustly execute the launched code without modifying the privilge of the whole (POSIX semantics honoring) main application.

A launcher is defined by one of these two functions: **cap_new_launcher**() or **cap_func_launcher**(). The return value being of opaque type **cap_launch_t** a return value of NULL implies an error has occurred.

Once defined, a **cap_launch_t** value can be used with **cap_launch**() to execute that *launcher*. In such cases, a non-negative return value indicates success: zero meaning success without a program being invoked; non-zero being equal to the process ID (**pid_t**) of the newly launched program.

A **cap_launch_t** occupies allocated memory and should be freed with **cap_free**(3). Before being **cap_free**(3)d a **cap_value_t** value may be reused for multiple independent launches. The *detail* argument to **cap_launch**(), in conjunction with the launcher's callback function, can be used to customize the invocation of the launch. Care must be taken to leverage custom shared memory (see **mmap**(2)) or some other IPC to return values to the main program via *detail* since the callback and any subsequent program execution will occur outside the main process of the calling application. An example of this would be to allocate detail as follows:

```
const char *args[] = { "echo", "hello", NULL };
cap_launch_t cmd = cap_new_launcher("/usr/bin/echo", args, NULL);
int *detail = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                   MAP_SHARED | MAP_ANONYMOUS, -1, 0);
cap_launcher_callback(cmd, &answer_detail_fn);
*detail = 41;
```

```
pid_t pid = cap_launch(cmd, detail);
printf("launcher callback set detail to %d\n", *detail);
munmap(detail, sizeof(int));
```

Unless modified by the callback function, the launched code will execute with the capability and other security context of the application.

If the callback function returns anything other than zero, a **cap_launch**() will be aborted. If detail of the failure is important to the caller, it should be communicated via the *detail* argument.

The following functions can be used to instruct the launcher to modify the security state of the invoked program without altering the state of the calling program. Such modifications must be performed prior to calling **cap_launch**() if they are to have the desired effect. Further, they are only invoked after any installed callback has completed. For example, one can drop or modify capabilities, *just* for executing a file.

The following functions instruct the launcher to do some common tasks of this sort (note some require permitted capability bits to succeed):

cap_launcher_callback() can be used to install or replace the currently installed callback function of the launcher.

cap_launcher_set_mode() can be used to ensure that the libcap-mode of the launched program is the desired one.

cap_launcher_set_iab() This function returns the **cap_iab_t** previously associated with the launcher. Calling this function with an IAB value of NULL will configure the launcher to not set an IAB value (the default). See**cap_iab**(3) for details on the IAB set. Note, the launcher is associated directly with the supplied *iab* value, and does not make a copy of it. This iab value is locked to the laucher and cannot be modified while associated with the launcher. Set with NULL to regain control over the memory associated with that IAB value, otherwise the IAB value will be **cap_free*()*'d when the launcher is.**

cap_launcher_set_chroot() This function causes the launched program executable to be invoked inside a chroot *root* directory.

cap_launcher_setuid() This function causes the launched program executable to be invoked with the specified user identifier (*uid_t*).

cap_launcher_setgroups() This function causes the launched program executable to be invoked with the specified primary and supplementary group IDs.

Note, if any of the launcher enhancements made by the above functions should fail to take effect (typically for a lack of sufficient privilege), the launch will fail and return -1.

ERRORS

A return of NULL for a **cap_launch_t** should be considered an error.

cap_launch() returns -1 in the case of an error.

In all such cases a return value of 0 implies success. In other cases, consult **errno**(3) for further details.

HISTORY

The **cap_launch**() family of functions were introduced in libcap 2.33. It primarily addresses a complexity with -*lpsx* linked pthreads(7) applications that use capabilities but also honor POSIX semantics.

Using −*lcap* and −*lpthread* together without the POSIX semantics support from −*lpsx* introduces a subtle class of exposure to privilege escalation. (See https://sites.google.com/site/fullycapable/who-ordered-libpsx

for an explanation.)

SEE ALSO

libpsx(3), **psx_syscall**(3), **libcap**(3), **cap_mode**(3), **cap_iab**(3), **capabilities**(7), **errno**(3), **fork**(2), **mmap**(2), **chroot**(2), and **munmap**(2).