NAME

libcapsicum — library interface to capability-mode services

LIBRARY

library “libcapsicum”

SYNOPSIS

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

int
lch_start(const char *sandbox, char *const argv[], u_int flags,
    struct lc_sandbox **lcsp);

int
lch_startfd(int fd_sandbox, char *const argv[], u_int flags,
    struct lc_sandbox **lcsp);

void
lch_stop(struct lc_sandbox *lcsp);

int
lch_autosandbox_isenabled(const char *servicename);

int
lch_getsock(struct lc_sandbox *lcsp, int *fdp);

int
lch_getpid(struct lc_sandbox *lcsp, pid_t *pidp);

int
lch_getprocdesc(struct lc_sandbox *lcsp, int *fdp);

ssize_t
lch_recv(struct lc_sandbox *lcsp, void *buf, size_t len, int flags);

ssize_t
lch_recv_rights(struct lc_sandbox *lcsp, void *buf, size_t len, int flags,
    int *fdp, int *fdcountp);

int
lch_rpc(struct lc_sandbox *lcsp, u_int32_t opno, struct iovec *req,
    int reqcount, struct iovec *rep, int repcount, size_t *replenp);

int
lch_rpc_rights(struct lc_sandbox *lcsp, u_int32_t opno, struct iovec *req,
    int reqcount, int *req_fdp, int req_fdcount, struct iovec *rep,
    int repcount, size_t *replenp, int *rep_fdp, int *rep_fdcountp);

ssize_t
lch_send(struct lc_sandbox *lcsp, const void *msg, size_t len, int flags);

ssize_t
lch_send_rights(struct lc_sandbox *lcsp, const void *msg, size_t len,
    int flags, int *fdp, int fdcount);
```

DESCRIPTION

The **libcapsicum** library routines provide services for processes hosting or running in capability mode. Depending on the requirements of the host and sandbox, the API can simply be used to set up and stop sandboxes, used to manage I/O using a `unix`(4) domain socket connection to the sandbox, or can provide a basic remote procedure call (RPC) facility. Applications may also use RPC generators such as `rpcgen`(1) to build event handling and marshaling code.

This man page describes the host API. General information on **libcapsicum** may be found in `libcapsicum`(3). Information on the sandbox API may be found in `libcapsicum_sandbox`(3).

HOST API

The **libcapsicum** host API allows processes to start, stop, and manage sandboxes running in capability mode. Host API functions can be identified by their function name prefix, `lch_`.

Each executing sandbox instance is described by an opaque which is returned by **lch_start**() for successfully started sandboxes, and passed into other APIs to indicate which sandbox should be acted on. **lch_start**() creates a new executing sandboxes, given the name of the sandbox binary via *sandbox*, and command line arguments *argv*, and optional flags *flags* to fine-tune aspects of sandbox operation; the only currently defined flag is `LCH_PERMIT_STDERR`, which allows the sandbox to write to the current process's `stderr`. By default, this is not permitted.

lch_startfd() accept a file descriptor argument, *fd_sandbox*, rather than a path, so is appropriate for use within a sandbox.

Executing sandboxes may be stopped (and all state freed) using **lch_stop**(). Following a call to **lch_stop**(), the *lchp* argument will no longer be valid.

Libraries and tools performing self-compartmentalization can use the interface **lch_autosandbox_isenabled** along with a unique string identifying their service to determine whether or not a global policy affecting the service requires sandboxing to be enabled or not.

Properties of the sandbox, such as the socket used to communicate with it, the proces descriptor for the sandbox process, and the pid, may be queried using **lch_getsock**(), **lch_getprocdesc**(), and **lch_getpid**().

libcapsicum implements a number of I/O functions as part of the host API, which are documented in `libcapsicum_host`(3). **lch_recv**() and **lch_send**() provide simple wrappers around `recv`(2) and `send`(2) to avoid sandbox consumers from having to query sandbox socket file descriptors before use.

lch_recv_rights() and **lch_send_rights**() are similar, but allow file descriptors to be attached the the messages received and sent. Both accept a pointer to a file descriptor array, *fdp*. Callers to **lch_recv_rights**() will pass in the length of the array via *fdcountp*, whose value will be changed to the actual number of file descriptors received. Callers to **lch_send_rights**() will pass in the number of file descriptors in the array via *fdcount*.

lch_rpc() provides a simple synchronous RPC facility, and is intended to be used in coordination with the **lcs_recvrpc**() and **lcs_sendrpc**() sandbox APIs. The host provides an operation number meaningful to the sandbox, *opno,* RPC arguments represented by *req* and *reqcount* using an ***iovec*** in the style of `writev`(2), and similar receive buffers passed via *rep* and *repcount*. If the RPC fails, -1 will be returned, or 0 and the size of any reply will be returned by reference using *replenp*. **lch_rpc_rights** allows the sending and receiving of file descriptors as part of the RPC operation.

SEE ALSO

`rpcgen`(1), `recv`(2), `send`(2), `writev`(2), `free`(3), `libcapsicum`(3), `libcapsicum_sandbox`(3), `malloc`(3), `unix`(4)

HISTORY

Support for capabilities and capabilities mode was developed as part of the TrustedBSD Project.

BUGS

WARNING: THIS IS EXPERIMENTAL SECURITY SOFTWARE THAT MUST NOT BE RELIED ON IN PRODUCTION SYSTEMS. IT WILL BREAK YOUR SOFTWARE IN NEW AND UNEXPECTED WAYS.

All sequence numbers will always have the value 0. This is fine from a retransmission perspective, as generally no retransmission should be required, but consumers should serialize use of the RPC service when consuming it from concurrent callers (such as multiple threads or multiple processes) to prevent I/O interlacing from corrupting the RPC stream.

AUTHORS

These functions and the capability facility were created by Robert N. M. Watson at the University of Cambridge Computer Laboratory with support from a grant from Google, Inc.