Subject: Re: [PATCH 2/2] capabilities: introduce per-process capability bounding set (v7)
Posted by Andrew Morgan on Fri, 16 Nov 2007 17:18:03 GMT
View Forum Message <> Reply to Message

Serge,

I've been thinking a lot about this one. As an alternative
implementation, have you considered changing one bounding capability bit
per system call? Something like this:

```
prctl(PR_CAPBSET_READ, CAPVERSION, CAP_NET_RAW);
    returns -> 1(allowed) or 0(blocked)
prctl(PR_CAPBSET_DROP, CAPVERSION, CAP_NET_RAW)
    returns -> 0(success) or -EPERM;
```

I also think we should use CAP_SETPCAP for the privilege of manipulating
the bounding set. In many ways irrevocably removing a permission
requires the same level of due care as adding one (to pI).

This has scalability designed in, at the expense of more system calls to
get the same (rare) work done.

Cheers

Andrew

Serge E. Hallyn wrote:
>>From 9ba95f1dbf88a512ffd423f6ccd627dc0460b052 Mon Sep 17 00:00:00 2001
> From: Serge E. Hallyn <serue@us.ibm.com>
> Date: Mon, 12 Nov 2007 16:50:04 -0500
> Subject: [PATCH 2/2] capabilities: introduce per-process capability bounding set (v7)
>
> The capability bounding set is a set beyond which capabilities
> cannot grow. Currently cap_bset is per-system. It can be
> manipulated through sysctl, but only init can add capabilities.
> Root can remove capabilities. By default it includes all caps
> except CAP_SETPCAP.
>
> This patch makes the bounding set per-process. It is inherited
> at fork from parent. Noone can add elements, CAP_SYS_ADMIN is
> required to remove them. Perhaps a new capability should be
> introduced to control the ability to remove capabilities, in
> order to help prevent running a privileged app with enough
> privs to be dangerous but not enough to be successful.
>
> One example use of this is to start a safer container. For
> instance, until device namespaces or per-container device

> whitelists are introduced, it is best to take CAP_MKNOD away
> from a container.
>
> Two questions:
>
> 1. I set CAP_FULL_SET and CAP_INIT_EFF_SET to contain
> only valid capabilities. Does that seem like a future maintenance
> headache? We only want the capability bounding set returned from kernel
> to container valid capabilities, so having CAP_FULL_SET contain all
> capabilities would mean that on every cap_prctl_getbset() we'd have to
> either manually clear invalid bits or let userspace sort it out.
>
> 2. Would getting and setting the bounding sets be
> better done through syscall? That better mirrors the capset+capget,
> but using prctl better mirrors the keep_capabilities setting.
>
> The following test program will get and set the bounding
> set. For instance
>
> ./bset get
>   (lists capabilities in bset)
> ./bset strset cap_sys_admin
>   (starts shell with new bset)
>   (use capset, setuid binary, or binary with
>   file capabilities to try to increase caps)
>
> ==========
> bset.c:
> ==========
> #include <sys/prctl.h>
> #include <linux/capability.h>
> #include <sys/types.h>
> #include <unistd.h>
> #include <stdio.h>
> #include <stdlib.h>
> #include <string.h>
>
> #ifndef PR_GET_CAPBSET
> #define PR_GET_CAPBSET 23
> #endif
>
> #ifndef PR_SET_CAPBSET
> #define PR_SET_CAPBSET 24
> #endif
>
> #define _LINUX_CAPABILITY_VERSION_1 0x19980330
> #define _LINUX_CAPABILITY_VERSION_2 0x20071026
> #define CAPVERSION _LINUX_CAPABILITY_VERSION_2

```
>
>  #define NUMCAPS 31
>
> int usage(char *me)
> {
>  printf("Usage: %s get\n", me);
>  printf("       %s set capability_string\n", me);
>  printf("         capability_string is for instance:\n");
>  printf("         cap_sys_admin,cap_mknod,cap_dac_override\n");
>  return 1;
> }
>
> char *captable[] = {
>  "cap_dac_override",
>  "cap_dac_read_search",
>  "cap_fowner",
>  "cap_fsetid",
>  "cap_kill",
>  "cap_setgid",
>  "cap_setuid",
>  "cap_setpcap",
>  "cap_linux_immutable",
>  "cap_net_bind_service",
>  "cap_net_broadcast",
>  "cap_net_admin",
>  "cap_net_raw",
>  "cap_ipc_lock",
>  "cap_ipc_owner",
>  "cap_sys_module",
>  "cap_sys_rawio",
>  "cap_sys_chroot",
>  "cap_sys_ptrace",
>  "cap_sys_pacct",
>  "cap_sys_admin",
>  "cap_sys_boot",
>  "cap_sys_nice",
>  "cap_sys_resource",
>  "cap_sys_time",
>  "cap_sys_tty_config",
>  "cap_mknod",
>  "cap_lease",
>  "cap_audit_write",
>  "cap_audit_control",
>  "cap_setfcap"
> };
>
> char *bittostr(unsigned int i, unsigned int j)
> {
```

```
>  if (i!=0 || j>31)
>   return "invalid";
>  return captable[j];
> }
>
> void print_capset(unsigned int *bset)
> {
>  unsigned int i, j, comma=0;
>  printf("Capability bounding set: ");
>  for (i=0; i<2; i++) {
>   for (j=0; j<31; j++)
>    if (bset[i] & (1 << (j+1)))
>     printf("%s%s", comma++?",":"",bittostr(i, j));
>  }
>  printf("\n");
> }
>
> int getbcap(void)
> {
>  unsigned int bset[2];
>  if (prctl(PR_GET_CAPBSET, CAPVERSION, &bset)) {
>   perror("prctl");
>   return 1;
>  }
>  print_capset(bset);
>  return 0;
> }
>
> int captoint(char *cap)
> {
>  int i;
>  for (i=0; i<NUMCAPS; i++)
>   if (strcmp(captable[i], cap) == 0)
>    return i+1;
>  return -1;
> }
>
> int setbcap(char *str)
> {
>  int ret;
>  unsigned int bset[2];
>  char *token = strtok(str, ",");
>
>  bset[0] = bset[1] = 0;
>  while (token) {
>   int bit = captoint(token);
>   if (bit < 0) {
>    printf("invalid cap: %s\n", token);
```

```
>    return 1;
>   }
>   bset[bit/32] |= 1 << (bit%32);
>   token = strtok(NULL, ",");
>
>  }
>  if (prctl(PR_SET_CAPBSET, CAPVERSION, &bset)) {
>   perror("prctl");
>   return 1;
>  }
>  return 0;
> }
>
> int main(int argc, char *argv[])
> {
>  if (argc<2)
>   return usage(argv[0]);
>  if (strcmp(argv[1], "get")==0)
>   return getbcap();
>  if (strcmp(argv[1], "set")!=0 || argc<3)
>   return usage(argv[0]);
>  if (setbcap(argv[2]))
>   return 1;
>  return execl("/bin/bash", "/bin/bash", NULL);
> }
> ==========================================================
>
> Changelog:
>  Enforce current-> capabilities are subsets of the
>  new bounding set.
>
>  As suggested by Andrew Morgan, send the capability
>  version along with the bset for prctl(PR_SET_CAPBSET)
>  and PR_GET_CAPBSET)
>
>  Adapt to 64-bit capabilities.
>
>  Update CAP_FULL_SET and CAP_INIT_EFF_SET to only
>  contain valid capabilities.
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> ---
>  include/linux/capability.h |   34 ++++++++++++++++++++++++++++++--
>  include/linux/init_task.h  |    1 +
>  include/linux/prctl.h      |    4 +++
>  include/linux/sched.h      |    2 +-
>  include/linux/security.h   |    5 ----
>  include/linux/sysctl.h     |    3 --
```

> kernel/fork.c | 1 +
> kernel/sys.c | 53 ++
> kernel/sysctl.c | 35 ----------------------------
> kernel/sysctl_check.c | 7 -----
> security/commoncap.c | 37 ++++++++++++++++++++++++++++++---
> 11 files changed, 124 insertions(+), 58 deletions(-)
>
> diff --git a/include/linux/capability.h b/include/linux/capability.h
> index a1d93da..64e668a 100644
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -202,7 +202,6 @@ typedef struct kernel_cap_struct {
> #define CAP_IPC_OWNER   15
>
> /* Insert and remove kernel modules - modify kernel without limit */
> -/* Modify cap_bset */
> #define CAP_SYS_MODULE   16
>
> /* Allow ioperm/iopl access */
> @@ -259,6 +258,7 @@ typedef struct kernel_cap_struct {
>   arbitrary SCSI commands */
> /* Allow setting encryption key on loopback filesystem */
> /* Allow setting zone reclaim policy */
> +/* Allow taking bits out of capability bounding set */
>
> #define CAP_SYS_ADMIN   21
>
> @@ -315,6 +315,12 @@ typedef struct kernel_cap_struct {
> #define CAP_SETFCAP   31
>
> /*
> + * XXX
> + * When adding a capability, please update the definitions of
> + * CAP_FULL_SET and CAP_INIT_EFF_SET below
> + */
> +
> +/*
>   * Bit location of each capability (used by user-space library and kernel)
>   */
>
> @@ -341,8 +347,8 @@ typedef struct kernel_cap_struct {
> #else /* HAND-CODED capability initializers */
>
> # define CAP_EMPTY_SET   {{ 0, 0 }}
> -# define CAP_FULL_SET   {{ ~0, ~0 }}
> -# define CAP_INIT_EFF_SET {{ ~CAP_TO_MASK(CAP_SETPCAP), ~0 }}
> +# define CAP_FULL_SET   {{ ~0, 0 }}
> +# define CAP_INIT_EFF_SET {{ ~CAP_TO_MASK(CAP_SETPCAP), 0 }}

```
>  # define CAP_FS_SET       {{ CAP_FS_MASK_B0, 0 }}
>  # define CAP_NFSD_SET     {{
CAP_FS_MASK_B0|CAP_TO_MASK(CAP_SYS_RESOURCE), 0 }}
>
> @@ -350,6 +356,17 @@ typedef struct kernel_cap_struct {
>
>  #define CAP_INIT_INH_SET    CAP_EMPTY_SET
>
> +#ifdef CONFIG_SECURITY_FILE_CAPABILITIES
> +/*
> + * Because of the reduced scope of CAP_SETPCAP when filesystem
> + * capabilities are in effect, it is safe to allow this capability to
> + * be available in the default configuration.
> + */
> +# define CAP_INIT_BSET  CAP_FULL_SET
> +#else
> +# define CAP_INIT_BSET  CAP_INIT_EFF_SET
> +#endif
> +
>  # define cap_clear(c)         do { (c) = __cap_empty_set; } while (0)
>  # define cap_set_full(c)      do { (c) = __cap_full_set; } while (0)
>  # define cap_set_init_eff(c)  do { (c) = __cap_init_eff_set; } while (0)
> @@ -465,6 +482,17 @@ extern const kernel_cap_t __cap_init_eff_set;
>  int capable(int cap);
>  int __capable(struct task_struct *t, int cap);
>
> +#ifdef CONFIG_COMMONCAP
> +extern int cap_prctl_setbset(kernel_cap_t new_bset);
> +extern int cap_prctl_getbset(kernel_cap_t *bset);
> +#else
> +#include <linux/errno.h>
> +static inline int cap_prctl_setbset(kernel_cap_t new_bset)
> +{ return -EINVAL; }
> +static inline int cap_prctl_getbset(kernel_cap_t *bset)
> +{ return -EINVAL; }
> +#endif
> +
>  #endif /* __KERNEL__ */
>
>  #endif /* !_LINUX_CAPABILITY_H */
> diff --git a/include/linux/init_task.h b/include/linux/init_task.h
> index cae35b6..5c84d14 100644
> --- a/include/linux/init_task.h
> +++ b/include/linux/init_task.h
> @@ -147,6 +147,7 @@ extern struct group_info init_groups;
>   .cap_effective = CAP_INIT_EFF_SET,    \
>   .cap_inheritable = CAP_INIT_INH_SET,    \
>   .cap_permitted = CAP_FULL_SET,     \
```

> + .cap_bset = CAP_INIT_BSET,   \
>   .keep_capabilities = 0,   \
>   .user = INIT_USER,   \
>   .comm = "swapper",   \
> diff --git a/include/linux/prctl.h b/include/linux/prctl.h
> index e2eff90..a7de023 100644
> --- a/include/linux/prctl.h
> +++ b/include/linux/prctl.h
> @@ -63,4 +63,8 @@
> #define PR_GET_SECCOMP 21
> #define PR_SET_SECCOMP 22
>
> +/* Get/set the capability bounding set */
> +#define PR_GET_CAPBSET 23
> +#define PR_SET_CAPBSET 24
> +
> #endif /* _LINUX_PRCTL_H */
> diff --git a/include/linux/sched.h b/include/linux/sched.h
> index 1d17f7c..bf51a16 100644
> --- a/include/linux/sched.h
> +++ b/include/linux/sched.h
> @@ -1041,7 +1041,7 @@ struct task_struct {
>   uid_t uid,euid,suid,fsuid;
>   gid_t gid,egid,sgid,fsgid;
>   struct group_info *group_info;
> - kernel_cap_t   cap_effective, cap_inheritable, cap_permitted;
> + kernel_cap_t   cap_effective, cap_inheritable, cap_permitted, cap_bset;
>   unsigned keep_capabilities:1;
>   struct user_struct *user;
> #ifdef CONFIG_KEYS
> diff --git a/include/linux/security.h b/include/linux/security.h
> index f771ad8..04b18f1 100644
> --- a/include/linux/security.h
> +++ b/include/linux/security.h
> @@ -34,11 +34,6 @@
> #include <linux/xfrm.h>
> #include <net/flow.h>
>
> -/*
> - * Bounding set
> - */
> -extern kernel_cap_t cap_bset;
> -
> extern unsigned securebits;
>
> struct ctl_table;
> diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
> index 4f5047d..fa900cb 100644

```
> --- a/include/linux/sysctl.h
> +++ b/include/linux/sysctl.h
> @@ -102,7 +102,6 @@ enum
>   KERN_NODENAME=7,
>   KERN_DOMAINNAME=8,
>
> - KERN_CAP_BSET=14, /* int: capability bounding set */
>   KERN_PANIC=15,  /* int: panic timeout */
>   KERN_REALROOTDEV=16, /* real root device to mount after initrd */
>
> @@ -962,8 +961,6 @@ extern int proc_dostring(struct ctl_table *, int, struct file *,
>      void __user *, size_t *, loff_t *);
>  extern int proc_dointvec(struct ctl_table *, int, struct file *,
>      void __user *, size_t *, loff_t *);
> -extern int proc_dointvec_bset(struct ctl_table *, int, struct file *,
> -        void __user *, size_t *, loff_t *);
>  extern int proc_dointvec_minmax(struct ctl_table *, int, struct file *,
>      void __user *, size_t *, loff_t *);
>  extern int proc_dointvec_jiffies(struct ctl_table *, int, struct file *,
> diff --git a/kernel/fork.c b/kernel/fork.c
> index 5639b3e..9e4a5e1 100644
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1087,6 +1087,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
>  #ifdef CONFIG_SECURITY
>   p->security = NULL;
>  #endif
> + p->cap_bset = current->cap_bset;
>   p->io_context = NULL;
>   p->audit_context = NULL;
>   cgroup_fork(p);
> diff --git a/kernel/sys.c b/kernel/sys.c
> index 4c77ed2..b2bca40 100644
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -1637,7 +1637,56 @@ asmlinkage long sys_umask(int mask)
>   mask = xchg(&current->fs->umask, mask & S_IRWXUGO);
>   return mask;
>  }
> +
> +long prctl_get_capbset(unsigned long vp, unsigned long bp)
> +{
> + long error;
> + int tocopy;
> + int i;
> + kernel_cap_t bset;
> +
> + if (vp == _LINUX_CAPABILITY_VERSION_2)
```

```
> +  tocopy = _LINUX_CAPABILITY_U32S_2;
> + else if (vp == _LINUX_CAPABILITY_VERSION_1)
> +  tocopy = _LINUX_CAPABILITY_U32S_1;
> + else
> +  return -EINVAL;
> +
> + error = cap_prctl_getbset(&bset);
> + if (error)
> +  return error;
> +
> + for (i = tocopy; i < _LINUX_CAPABILITY_U32S; i++) {
> +  if (bset.cap[i])
> +   /* Cannot represent w/ legacy structure */
> +   return -ERANGE;
> + }
> +
> + error = copy_to_user((__u32 __user *)bp, &bset, tocopy * sizeof(__u32));
> + return error;
> +}
>
> +long prctl_set_capbset(unsigned long vp, unsigned long bp)
> +{
> + int tocopy;
> + int i;
> + kernel_cap_t bset;
> +
> + if (vp == _LINUX_CAPABILITY_VERSION_2)
> +  tocopy = _LINUX_CAPABILITY_U32S_2;
> + else if (vp == _LINUX_CAPABILITY_VERSION_1)
> +  tocopy = _LINUX_CAPABILITY_U32S_1;
> + else
> +  return -EINVAL;
> +
> + if (copy_from_user(&bset, (__u32 __user *)bp, tocopy*sizeof(__u32)))
> +  return -EFAULT;
> + for (i = tocopy; i < _LINUX_CAPABILITY_U32S; i++)
> +  bset.cap[i] = 0;
> +
> + return cap_prctl_setbset(bset);
> +}
> +
>  asmlinkage long sys_prctl(int option, unsigned long arg2, unsigned long arg3,
>       unsigned long arg4, unsigned long arg5)
>  {
> @@ -1738,6 +1787,10 @@ asmlinkage long sys_prctl(int option, unsigned long arg2, unsigned
long arg3,
>    case PR_GET_SECCOMP:
>     error = prctl_get_seccomp();
```

```
>     break;
> +  case PR_GET_CAPBSET:
> +   return prctl_get_capbset(arg2, arg3);
> +  case PR_SET_CAPBSET:
> +   return prctl_set_capbset(arg2, arg3);
>    case PR_SET_SECCOMP:
>     error = prctl_set_seccomp(arg2);
>     break;
> diff --git a/kernel/sysctl.c b/kernel/sysctl.c
> index 489b0d1..d858819 100644
> --- a/kernel/sysctl.c
> +++ b/kernel/sysctl.c
> @@ -383,15 +383,6 @@ static struct ctl_table kern_table[] = {
>     .proc_handler = &proc_dointvec_taint,
>   },
>  #endif
> -#ifdef CONFIG_SECURITY_CAPABILITIES
> - {
> -  .procname = "cap-bound",
> -  .data  = &cap_bset,
> -  .maxlen  = sizeof(kernel_cap_t),
> -  .mode  = 0600,
> -  .proc_handler = &proc_dointvec_bset,
> - },
> -#endif /* def CONFIG_SECURITY_CAPABILITIES */
>  #ifdef CONFIG_BLK_DEV_INITRD
>   {
>    .ctl_name = KERN_REALROOTDEV,
> @@ -1910,26 +1901,6 @@ static int do_proc_dointvec_bset_conv(int *negp, unsigned long
*lvalp,
>   return 0;
>  }
>
> -#ifdef CONFIG_SECURITY_CAPABILITIES
> -/*
> - * init may raise the set.
> - */
> -
> -int proc_dointvec_bset(struct ctl_table *table, int write, struct file *filp,
> -   void __user *buffer, size_t *lenp, loff_t *ppos)
> -{
> - int op;
> -
> - if (write && !capable(CAP_SYS_MODULE)) {
> -  return -EPERM;
> - }
> -
> - op = is_global_init(current) ? OP_SET : OP_AND;
```

> - return do_proc_dointvec(table,write,filp,buffer,lenp,ppos,
> -   do_proc_dointvec_bset_conv,&op);
> -}
> -#endif /* def CONFIG_SECURITY_CAPABILITIES */
> -
> /*
>   * Taint values can only be increased
>   */
> @@ -2343,12 +2314,6 @@ int proc_dointvec(struct ctl_table *table, int write, struct file *filp,
>   return -ENOSYS;
> }
>
> -int proc_dointvec_bset(struct ctl_table *table, int write, struct file *filp,
> -   void __user *buffer, size_t *lenp, loff_t *ppos)
> -{
> - return -ENOSYS;
> -}
> -
> int proc_dointvec_minmax(struct ctl_table *table, int write, struct file *filp,
>   void __user *buffer, size_t *lenp, loff_t *ppos)
> {
> diff --git a/kernel/sysctl_check.c b/kernel/sysctl_check.c
> index 8f5baac..526fa36 100644
> --- a/kernel/sysctl_check.c
> +++ b/kernel/sysctl_check.c
> @@ -38,10 +38,6 @@ static struct trans_ctl_table trans_kern_table[] = {
>   { KERN_NODENAME, "hostname" },
>   { KERN_DOMAINNAME, "domainname" },
>
> -#ifdef CONFIG_SECURITY_CAPABILITIES
> - { KERN_CAP_BSET, "cap-bound" },
> -#endif /* def CONFIG_SECURITY_CAPABILITIES */
> -
>   { KERN_PANIC,   "panic" },
>   { KERN_REALROOTDEV, "real-root-dev" },
>
> @@ -1522,9 +1518,6 @@ int sysctl_check_table(struct ctl_table *table)
>   (table->strategy == sysctl_ms_jiffies) ||
>   (table->proc_handler == proc_dostring) ||
>   (table->proc_handler == proc_dointvec) ||
> -#ifdef CONFIG_SECURITY_CAPABILITIES
> -   (table->proc_handler == proc_dointvec_bset) ||
> -#endif /* def CONFIG_SECURITY_CAPABILITIES */
>   (table->proc_handler == proc_dointvec_minmax) ||
>   (table->proc_handler == proc_dointvec_jiffies) ||
>   (table->proc_handler == proc_dointvec_userhz_jiffies) ||
> diff --git a/security/commoncap.c b/security/commoncap.c
> index 3a95990..d28222f 100644

```
> --- a/security/commoncap.c
> +++ b/security/commoncap.c
> @@ -36,9 +36,6 @@
>  # define CAP_INIT_BSET  CAP_INIT_EFF_SET
>  #endif /* def CONFIG_SECURITY_FILE_CAPABILITIES */
>
> -kernel_cap_t cap_bset = CAP_INIT_BSET;    /* systemwide capability bound */
> -EXPORT_SYMBOL(cap_bset);
> -
>  /* Global security state */
>
>  unsigned securebits = SECUREBITS_DEFAULT; /* systemwide security settings */
> @@ -330,7 +327,8 @@ void cap_bprm_apply_creds (struct linux_binprm *bprm, int unsafe)
>   /* Derived from fs/exec.c:compute_creds. */
>   kernel_cap_t new_permitted, working;
>
> - new_permitted = cap_intersect (bprm->cap_permitted, cap_bset);
> + new_permitted = cap_intersect (bprm->cap_permitted,
> +     current->cap_bset);
>   working = cap_intersect (bprm->cap_inheritable,
>       current->cap_inheritable);
>   new_permitted = cap_combine (new_permitted, working);
> @@ -611,3 +609,34 @@ int cap_vm_enough_memory(struct mm_struct *mm, long pages)
>   return __vm_enough_memory(mm, pages, cap_sys_admin);
>  }
>
> +/*
> + * cap_prctl_setbset currently requires CAP_SYS_ADMIN.  The reason is
> + * my fear that an ordinary user could selectively take capabilities
> + * out, then run a setuid root binary or binary with file capabilities,
> + * which would perform part of a dangerous action with CAP_SOMECAP1,
> + * then fail to perform the second part of the action because
> + * CAP_SOMECAP2 is not in bset, leaving things in an unsafe state,
> + * i.e a sensitive file owned by the non-root user because CAP_CHOWN
> + * was not allowed.
> + */
> +int cap_prctl_setbset(kernel_cap_t new_bset)
> +{
> + if (!capable(CAP_SYS_ADMIN))
> +  return -EPERM;
> + if (!cap_issubset(new_bset, current->cap_bset))
> +  return -EPERM;
> + current->cap_bset = new_bset;
> + current->cap_effective = cap_intersect(current->cap_effective,
> +  new_bset);
> + current->cap_permitted = cap_intersect(current->cap_permitted,
> +  new_bset);
> + current->cap_inheritable = cap_intersect(current->cap_inheritable,
```

> + new_bset);
> + return 0;
> +}
> +
> +int cap_prctl_getbset(kernel_cap_t *bset)
> +{
> + *bset = current->cap_bset;
> + return 0;
> +}

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers