Subject: [RFC PATCH 1/2] capabilities: define CONFIG_COMMONCAP
Posted by serue on Thu, 15 Nov 2007 23:16:15 GMT
View Forum Message <> Reply to Message

>From 1512a99aedb7a75ac993ccef91a42c97e1baefc5 Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Fri, 28 Sep 2007 10:33:33 -0500
Subject: [PATCH 1/2] capabilities: define CONFIG_COMMONCAP

currently the compilation of commoncap.c is determined
through Makefile logic. So there is no single CONFIG
variable which can be relied upon to know whether it
will be compiled.

Define CONFIG_COMMONCAP to be true when lsm is not
compiled in, or when the capability or rootplug modules
are compiled. These are the cases when commoncap is
currently compiled. Use this variable in security/Makefile
to determine commoncap.c's compilation.

Apart from being a logic cleanup, this is needed by the
upcoming cap_bset patch so that prctl can know whether
PR_SET_BSET should be supported.

Changelog:
 Nov 15: make CONFIG_FILE_CAPABILITIES just depend on
  COMMONCAP. Unfortunately since rootplug doesn't
  hook cap_setxattr, it would be not quite right
  to allow CONFIG_FILE_CAPABILITIES with rootplug.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 security/Kconfig | 6 +++++-
 security/Makefile | 9 +++------
 2 files changed, 8 insertions(+), 7 deletions(-)

diff --git a/security/Kconfig b/security/Kconfig
index 8086e61..de7f9fe 100644
--- a/security/Kconfig
+++ b/security/Kconfig
@@ -80,9 +80,13 @@ config SECURITY_CAPABILITIES
   This enables the "default" Linux capabilities functionality.
   If you are unsure how to answer this question, answer Y.

+config COMMONCAP
+ bool
+ default !SECURITY || SECURITY_CAPABILITIES || SECURITY_ROOTPLUG
+

```
 config SECURITY_FILE_CAPABILITIES
  bool "File POSIX Capabilities (EXPERIMENTAL)"
- depends on (SECURITY=n || SECURITY_CAPABILITIES!=n) && EXPERIMENTAL
+ depends on COMMONCAP && !SECURITY_ROOTPLUG && EXPERIMENTAL
  default n
  help
    This enables filesystem capabilities, allowing you to give
diff --git a/security/Makefile b/security/Makefile
index ef87df2..7cccc81 100644
--- a/security/Makefile
+++ b/security/Makefile
@@ -5,14 +5,11 @@
 obj-$(CONFIG_KEYS)   += keys/
 subdir-$(CONFIG_SECURITY_SELINUX) += selinux

-# if we don't select a security model, use the default capabilities
-ifneq ($(CONFIG_SECURITY),y)
-obj-y  += commoncap.o
-endif
+obj-$(CONFIG_COMMONCAP)   += commoncap.o

 # Object file lists
 obj-$(CONFIG_SECURITY)   += security.o dummy.o inode.o
 # Must precede capability.o in order to stack properly.
 obj-$(CONFIG_SECURITY_SELINUX)  += selinux/built-in.o
-obj-$(CONFIG_SECURITY_CAPABILITIES) += commoncap.o capability.o
-obj-$(CONFIG_SECURITY_ROOTPLUG)  += commoncap.o root_plug.o
+obj-$(CONFIG_SECURITY_CAPABILITIES) += capability.o
+obj-$(CONFIG_SECURITY_ROOTPLUG)  += root_plug.o
--
1.5.1.1.GIT
```

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

---

Subject: [PATCH 2/2] capabilities: introduce per-process capability bounding set (v7)

Posted by serue on Thu, 15 Nov 2007 23:17:52 GMT

View Forum Message <> Reply to Message

>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;
+}
--
1.5.1.1.GIT
```

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

---

Subject: [PATCH RFC] cgroups: implement device whitelist cgroup+lsm
Posted by serue on Thu, 15 Nov 2007 23:23:25 GMT
View Forum Message <> Reply to Message

(This patch is based against the same CONFIG_COMMONCAP patch as
the capability bounding set patch I just sent out)

```
>From fcbd0bd0a8ee1e37a68f5381b47ec0746cb9b1cc Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Thu, 11 Oct 2007 15:27:48 -0400
Subject: [PATCH 2/2] cgroups: implement device whitelist cgroup+lsm
```

Implement a cgroup using the LSM interface to enforce open and mknod
on device files. Not a line of this code is expected to be used in a
final version, this is just a proof of concept to explore whether we

can or should use an LSM for this purpose until device namespaces are
really needed. The alternative is to simply set up a static /dev for
each container and remove CAP_MKNOD from the container's bounding
set. Several people feel that that approach is insufficient.

This patch implements a simple device access whitelist. A whitelist entry
has 4 fields. 'type' is a (all), c (char), or b (block). 'all' means it
applies to all types, all major numbers, and all minor numbers. Major and
minor are obvious. Access is a composition of r (read), w (write), and
m (mknod).

The root devcgroup starts with rwm to 'all'. A child devcg gets a copy
of the parent. Admins can then add and remove devices to the whitelist.
Once CAP_HOST_ADMIN is introduced it will be needed to add entries as
well or remove entries from another cgroup, though just CAP_SYS_ADMIN
will suffice to remove entries for your own group.

An entry is added by doing "echo <type> <maj> <min> <access>" > devcg.allow,
for instance:

 echo b 7 0 mrw > /cgroups/1/devcg.allow

An entry is removed by doing likewise into devcg.deny. Since this is a
pure whitelist, not acls, you can only remove entries which exist in the
whitelist. You must explicitly

 echo a 0 0 mrw > /cgroups/1/devcg.deny

to remove the "allow all" entry which is automatically inherited from
the root cgroup.

While composing this with the ns_cgroup may seem logical, it may not
be the right thing to do. Note that each newly created devcg gets
a copy of the parent whitelist. So if you had done

 mount -t cgroup -o ns,devcg none /cgroups

then once a process in /cgroup/1 had done an unshare(CLONE_NEWNS)
it would be under /cgroup/1/node_<pid>
if an admin did

 echo b 7 0 m > /cgroups/1/devcg.deny

then the entry would still be in the whitelist for /cgroups/1/node_<pid>.
Something to discuss if we get that far before nixing this whole idea.

The devcg module calls all the capability security hooks, so
it does not need to (cannot) be stacked with capability.ko.

The security hooks are defined in a separate file from the
cgroup code so that the security/Makefile can force its hooks
to be loaded after the selinux hooks. Otherwise selinux would
refuse to load if CONFIG_CGROUP_DEV=y.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/cgroup_subsys.h | 6 +
 init/Kconfig | 7 +
 kernel/Makefile | 1 +
 kernel/dev_cgroup.c | 410 ++
 security/Kconfig | 4 +-
 security/Makefile | 1 +
 6 files changed, 427 insertions(+), 2 deletions(-)
 create mode 100644 kernel/dev_cgroup.c

diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index d3ec2ed..9e2f5f7 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_DEV
+SUBSYS(devcg)
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index 96fba82..2907248 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -324,6 +324,13 @@ config CPUSETS

    Say N if unsure.

+config CGROUP_DEV
+ bool "Device controller for cgroups"
+ depends on CGROUPS && SECURITY && EXPERIMENTAL
+ help
+   Provides a cgroup implementing whitelists for devices which
+   a process in the cgroup can mknod or open.
+
 config FAIR_GROUP_SCHED
  bool "Fair group CPU scheduler"
  default y

```
diff --git a/kernel/Makefile b/kernel/Makefile
index 876dbcd..1da0b66 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -41,6 +41,7 @@ obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
+obj-$(CONFIG_CGROUP_DEV) += dev_cgroup.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
diff --git a/kernel/dev_cgroup.c b/kernel/dev_cgroup.c
new file mode 100644
index 0000000..365877d
--- /dev/null
+++ b/kernel/dev_cgroup.c
@@ -0,0 +1,410 @@
+/*
+ * dev_cgroup.c - device cgroup subsystem
+ *
+ * Copyright 2007 IBM Corp
+ */
+
+#include <linux/devcg.h>
+
+/*
+ * Once 64-bit caps and CAP_HOST_ADMIN exist, we will be
+ * requiring (CAP_HOST_ADMIN|CAP_MKNOD) to create a device
+ * not in the whitelist, * (CAP_HOST_ADMIN|CAP_SYS_ADMIN)
+ * to edit the whitelist,
+ */
+static int devcg_can_attach(struct cgroup_subsys *ss,
+  struct cgroup *new_cgroup, struct task_struct *task)
+{
+ struct cgroup *orig;
+
+ if (current != task) {
+  if (!cgroup_is_descendant(new_cgroup))
+   return -EPERM;
+ }
+
+ if (atomic_read(&new_cgroup->count) != 0)
+  return -EPERM;
+
+ orig = task_cgroup(task, devcg_subsys_id);
+ if (orig && orig != new_cgroup->parent)
+  return -EPERM;
```

```
+
+ return 0;
+}
+
+/*
+ * called under cgroup_lock()
+ */
+int dev_whitelist_copy(struct list_head *dest, struct list_head *orig)
+{
+ struct dev_whitelist_item *wh, *tmp, *new;
+
+ list_for_each_entry(wh, orig, list) {
+  new = kmalloc(sizeof(*wh), GFP_KERNEL);
+  if (!new)
+   goto free_and_exit;
+  new->major = wh->major;
+  new->minor = wh->minor;
+  new->type = wh->type;
+  new->access = wh->access;
+  list_add_tail(&new->list, dest);
+ }
+
+ return 0;
+
+free_and_exit:
+ list_for_each_entry_safe(wh, tmp, dest, list) {
+  list_del(&wh->list);
+  kfree(wh);
+ }
+ return -ENOMEM;
+}
+
+/* Stupid prototype - don't bother combining existing entries */
+/*
+ * called under cgroup_lock()
+ * since the list is visible to other tasks, we need the spinlock also
+ */
+void dev_whitelist_add(struct dev_cgroup *dev_cgroup,
+   struct dev_whitelist_item *wh)
+{
+ spin_lock(&dev_cgroup->lock);
+ list_add_tail(&wh->list, &dev_cgroup->whitelist);
+ spin_unlock(&dev_cgroup->lock);
+}
+
+/*
+ * called under cgroup_lock()
+ * since the list is visible to other tasks, we need the spinlock also
```

```
+ */
+void dev_whitelist_rm(struct dev_cgroup *dev_cgroup,
+   struct dev_whitelist_item *wh)
+{
+ struct dev_whitelist_item *walk, *tmp;
+
+ spin_lock(&dev_cgroup->lock);
+ list_for_each_entry_safe(walk, tmp, &dev_cgroup->whitelist, list) {
+  if (walk->type & DEV_ALL) {
+   list_del(&walk->list);
+   kfree(walk);
+   continue;
+  }
+  if (walk->type != wh->type)
+   continue;
+  if (walk->major != wh->major || walk->minor != wh->minor)
+   continue;
+  walk->access &= ~wh->access;
+  if (!walk->access) {
+   list_del(&walk->list);
+   kfree(walk);
+  }
+ }
+ spin_unlock(&dev_cgroup->lock);
+}
+
+/*
+ * Rules: you can only create a cgroup if
+ *     1. you are capable(CAP_SYS_ADMIN)
+ *     2. the target cgroup is a descendant of your own cgroup
+ *
+ * Note: called from kernel/cgroup.c with cgroup_lock() held.
+ */
+static struct cgroup_subsys_state *devcg_create(struct cgroup_subsys *ss,
+      struct cgroup *cgroup)
+{
+ struct dev_cgroup *dev_cgroup, *parent_dev_cgroup;
+ struct cgroup *parent_cgroup;
+ int ret;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return ERR_PTR(-EPERM);
+ if (!cgroup_is_descendant(cgroup))
+  return ERR_PTR(-EPERM);
+
+ dev_cgroup = kzalloc(sizeof(*dev_cgroup), GFP_KERNEL);
+ if (!dev_cgroup)
+  return ERR_PTR(-ENOMEM);
```

```
+ INIT_LIST_HEAD(&dev_cgroup->whitelist);
+ parent_cgroup = cgroup->parent;
+
+ if (parent_cgroup == NULL) {
+  struct dev_whitelist_item *wh;
+  wh = kmalloc(sizeof(*wh), GFP_KERNEL);
+  wh->minor = wh->major = 0;
+  wh->type = DEV_ALL;
+  wh->access = ACC_MKNOD | ACC_READ | ACC_WRITE;
+  list_add(&wh->list, &dev_cgroup->whitelist);
+ } else {
+  parent_dev_cgroup = cgroup_to_devcg(parent_cgroup);
+  ret = dev_whitelist_copy(&dev_cgroup->whitelist,
+    &parent_dev_cgroup->whitelist);
+  if (ret) {
+   kfree(dev_cgroup);
+   return ERR_PTR(ret);
+  }
+ }
+
+ spin_lock_init(&dev_cgroup->lock);
+ return &dev_cgroup->css;
+}
+
+static void devcg_destroy(struct cgroup_subsys *ss,
+   struct cgroup *cgroup)
+{
+ struct dev_cgroup *dev_cgroup;
+ struct dev_whitelist_item *wh, *tmp;
+
+ dev_cgroup = cgroup_to_devcg(cgroup);
+ list_for_each_entry_safe(wh, tmp, &dev_cgroup->whitelist, list) {
+  list_del(&wh->list);
+  kfree(wh);
+ }
+ kfree(dev_cgroup);
+}
+
+#define DEVCG_ALLOW 1
+#define DEVCG_DENY 2
+
+void set_access(char *acc, short access)
+{
+ int idx = 0;
+ memset(acc, 0, 4);
+ if (access & ACC_READ)
+  acc[idx++] = 'r';
+ if (access & ACC_WRITE)
```

```
+  acc[idx++] = 'w';
+ if (access & ACC_MKNOD)
+  acc[idx++] = 'm';
+}
+
+char type_to_char(short type)
+{
+ if (type == DEV_ALL)
+  return 'a';
+ if (type == DEV_CHAR)
+  return 'c';
+ if (type == DEV_BLOCK)
+  return 'b';
+ return 'X';
+}
+
+char *print_whitelist(struct dev_cgroup *devcgroup, int *len)
+{
+ char *buf, *s, acc[4];
+ struct dev_whitelist_item *wh;
+ int ret;
+ int count = 0;
+
+ buf = kmalloc(4096, GFP_KERNEL);
+ if (!buf)
+  return ERR_PTR(-ENOMEM);
+ s = buf;
+ *s = '\0';
+ *len = 0;
+
+ spin_lock(&devcgroup->lock);
+ list_for_each_entry(wh, &devcgroup->whitelist, list) {
+  set_access(acc, wh->access);
+  printk(KERN_NOTICE
+   "%s (count%d): whtype %hd maj %u min %u acc %hd\n",
+   __FUNCTION__, count, wh->type, wh->major, wh->minor,
+   wh->access);
+  ret = snprintf(s, 4095-(s-buf), "%c %u %u %s\n",
+   type_to_char(wh->type), wh->major, wh->minor, acc);
+  if (s+ret >= buf+4095) {
+   kfree(buf);
+   buf = ERR_PTR(-ENOMEM);
+   break;
+  }
+  s += ret;
+  *len += ret;
+  count++;
+ }
```

```
+ spin_unlock(&devcgroup->lock);
+
+ return buf;
+}
+
+static ssize_t devcg_access_read(struct cgroup *cgroup,
+    struct cftype *cft, struct file *file,
+    char __user *userbuf, size_t nbytes, loff_t *ppos)
+{
+ struct dev_cgroup *devcgrp = cgroup_to_devcg(cgroup);
+ int filetype = cft->private;
+ char *buffer;
+ int len, retval;
+
+ if (filetype != DEVCG_ALLOW)
+  return -EINVAL;
+ buffer = print_whitelist(devcgrp, &len);
+ if (IS_ERR(buffer))
+  return PTR_ERR(buffer);
+
+ retval = simple_read_from_buffer(userbuf, nbytes, ppos, buffer, len);
+ kfree(buffer);
+ return retval;
+}
+
+static inline short convert_access(char *acc)
+{
+ short access = 0;
+
+ while (*acc) {
+  switch (*acc) {
+  case 'r':
+  case 'R': access |= ACC_READ; break;
+  case 'w':
+  case 'W': access |= ACC_WRITE; break;
+  case 'm':
+  case 'M': access |= ACC_MKNOD; break;
+  case '\n': break;
+  default:
+   return -EINVAL;
+  }
+  acc++;
+ }
+
+ return access;
+}
+
+static inline short convert_type(char intype)
```

```
+{
+ short type = 0;
+ switch (intype) {
+ case 'a': type = DEV_ALL; break;
+ case 'c': type = DEV_CHAR; break;
+ case 'b': type = DEV_BLOCK; break;
+ default: type = -EACCES; break;
+ }
+ return type;
+}
+
+/*
+ * Current rules:
+ * CAP_SYS_ADMIN needed for all writes.
+ * when we have CAP_HOST_ADMIN, the rules will become:
+ *   if (!writetoself)
+ *      require capable(CAP_HOST_ADMIN | CAP_SYS_ADMIN);
+ *   if (is_allow)
+ *      require capable(CAP_HOST_ADMIN | CAP_SYS_ADMIN);
+ *   require capable(CAP_SYS_ADMIN);
+ */
+static int have_write_permission(int is_allow, int writetoself)
+{
+ if (!capable(CAP_SYS_ADMIN))
+  return 0;
+ return 1;
+}
+
+static ssize_t devcg_access_write(struct cgroup *cgroup, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ struct cgroup *cur_cgroup;
+ struct dev_cgroup *devcgrp, *cur_devcgroup;
+ int filetype = cft->private;
+ char *buffer, acc[4];
+ int retval = 0;
+ int nitems;
+ char type;
+ struct dev_whitelist_item *wh;
+
+ devcgrp = cgroup_to_devcg(cgroup);
+ cur_cgroup = task_cgroup(current, devcg_subsys.subsys_id);
+ cur_devcgroup = cgroup_to_devcg(cur_cgroup);
+
+ if (!have_write_permission(filetype == DEVCG_ALLOW,
+    cur_devcgroup == devcgrp))
+  return -EPERM;
```

```
+
+ buffer = kmalloc(nbytes+1, GFP_KERNEL);
+ if (!buffer)
+  return -ENOMEM;
+
+ wh = kmalloc(sizeof(*wh), GFP_KERNEL);
+ if (!wh) {
+  kfree(buffer);
+  return -ENOMEM;
+ }
+
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+  retval = -EFAULT;
+  goto out1;
+ }
+ buffer[nbytes] = 0; /* nul-terminate */
+
+ cgroup_lock();
+ if (cgroup_is_removed(cgroup)) {
+  retval = -ENODEV;
+  goto out2;
+ }
+
+ memset(wh, 0, sizeof(*wh));
+ memset(acc, 0, 4);
+ nitems = sscanf(buffer, "%c %u %u %3s", &type, &wh->major, &wh->minor,
+   acc);
+ retval = -EINVAL;
+ if (nitems != 4)
+  goto out2;
+ wh->type = convert_type(type);
+ if (wh->type < 0)
+  goto out2;
+ wh->access = convert_access(acc);
+ if (wh->access < 0)
+  goto out2;
+ retval = 0;
+ switch (filetype) {
+ case DEVCG_ALLOW:
+  printk(KERN_NOTICE
+   "%s: add whtype %hd maj %u min %u acc %hd\n",
+   __FUNCTION__, wh->type, wh->major, wh->minor,
+   wh->access);
+  dev_whitelist_add(devcgrp, wh);
+  break;
+ case DEVCG_DENY:
+  dev_whitelist_rm(devcgrp, wh);
+  break;
```

```
+ default:
+  retval = -EINVAL;
+  goto out2;
+ }
+
+ if (retval == 0)
+  retval = nbytes;
+
+out2:
+ cgroup_unlock();
+out1:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype dev_cgroup_files[] = {
+ {
+  .name = "allow",
+  .read = devcg_access_read,
+  .write  = devcg_access_write,
+  .private = DEVCG_ALLOW,
+ },
+ {
+  .name = "deny",
+  .write = devcg_access_write,
+  .private = DEVCG_DENY,
+ },
+};
+
+static int devcg_populate(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss, dev_cgroup_files,
+     ARRAY_SIZE(dev_cgroup_files));
+}
+
+struct cgroup_subsys devcg_subsys = {
+ .name = "devcg",
+ .can_attach = devcg_can_attach,
+ .create = devcg_create,
+ .destroy  = devcg_destroy,
+ .populate = devcg_populate,
+ .subsys_id = devcg_subsys_id,
+};
diff --git a/security/Kconfig b/security/Kconfig
index de7f9fe..0132e49 100644
--- a/security/Kconfig
+++ b/security/Kconfig
```

```
@@ -75,14 +75,14 @@ config SECURITY_NETWORK_XFRM

 config SECURITY_CAPABILITIES
  bool "Default Linux Capabilities"
- depends on SECURITY
+ depends on SECURITY && !CGROUP_DEV
  help
    This enables the "default" Linux capabilities functionality.
    If you are unsure how to answer this question, answer Y.

 config COMMONCAP
  bool
- default !SECURITY || SECURITY_CAPABILITIES || SECURITY_ROOTPLUG
+ default !SECURITY || SECURITY_CAPABILITIES || SECURITY_ROOTPLUG || CGROUP_DEV

 config SECURITY_FILE_CAPABILITIES
  bool "File POSIX Capabilities (EXPERIMENTAL)"
diff --git a/security/Makefile b/security/Makefile
index 7cccc81..5a83753 100644
--- a/security/Makefile
+++ b/security/Makefile
@@ -13,3 +13,4 @@ obj-$(CONFIG_SECURITY)   += security.o dummy.o inode.o
 obj-$(CONFIG_SECURITY_SELINUX)  += selinux/built-in.o
 obj-$(CONFIG_SECURITY_CAPABILITIES) += capability.o
 obj-$(CONFIG_SECURITY_ROOTPLUG)  += root_plug.o
+obj-$(CONFIG_CGROUP_DEV)  += dev_cgroup_lsm.o
--
1.5.1.1.GIT
```

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

---

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

Subject: Re: [PATCH 2/2] capabilities: introduce per-process capability bounding set (v7)
Posted by serue on Sat, 17 Nov 2007 03:42:26 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morgan (morgan@kernel.org):
> 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;

Interesting. Didn't like it on first read, but it certainly has its
appeal. Sure, I'll whip something like this up.

> 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).

Aside from being heavy-handed, it also means that we are restricting the
use of per-process capability bounding sets to kernels with file
capabilities compiled in, right? Are we ok with that?

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

Thanks,
-serge

>
> 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;
> > +}
>
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

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

---

Subject: Re: [PATCH 2/2] capabilities: introduce per-process capability bounding set (v7)
Posted by Andrew Morgan on Sat, 17 Nov 2007 04:22:42 GMT
View Forum Message <> Reply to Message

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Serge E. Hallyn wrote:
>> 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 pl).
>
> Aside from being heavy-handed, it also means that we are restricting the
> use of per-process capability bounding sets to kernels with file
> capabilities compiled in, right? Are we ok with that?
>

I am. :-)

Cheers

Andrew
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.2.6 (GNU/Linux)

iD8DBQFHPmyQQheEq9QabfIRAnnbAJ0c22LPNc1EnjWyvR4ZrwcyAiJDrgCeOdTj
TJFJwUK7UMkeX5M9ULzbN44=
=LMQP
-----END PGP SIGNATURE-----

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