Subject: [RFC][patch 0/3] Network container subsystem - bind filtering
Posted by Daniel Lezcano on Tue, 04 Sep 2007 17:00:22 GMT
View Forum Message <> Reply to Message

Paul Menage mentionned, a few weeks ago, he wanted a bind filtering
for containers. Here it is :)

The following patches are a proposition to bring IP isolation to a container.

After looking more closely at the code I found that security hooks are
at the right place to catch socket calls. The IP isolation relies on the
security hooks and that has the advantage of not having the kernel code modified,
(expect container.h and makefile/kconfig), the patchset provide just a new
file container_network.c

Roughly, a container has a subsystem for the network (only ipv4).
This subsystem contains the list of the addresses allowed to be used by the
container. If a container tries to bind to an address not contained into
this list, the bind will fail with EPERM. Of course the bind is allowed to
INADDR_ANY.

If this approach is ok for everyone, I can extend the bind filtering to
consolidate the IP isolation.

Regards.

--

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

Subject: [RFC][patch 1/3] network container subsystem
Posted by Daniel Lezcano on Tue, 04 Sep 2007 17:00:23 GMT
View Forum Message <> Reply to Message

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patch creates the network container subsystem.
It consists for the moment on a single file "network.ipv4".

The interface is pretty simple:

To add an IP address to the container:

```
 echo add AB12FFFF > network.ipv4
```

To remove this IP address:

```
 echo del AB12FFFF > network.ipv4
```

To list the addresses:

```
 cat network.ipv4
```

The parameter is an IPV4 address in the hexa format. The parsing of a dotted-decimal parameter is totally painful. If this format hurts someone, I can change it to a dotted format at the risk of having something buggy.

This patch by itself does nothing more than adding/removing elements from a list.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
 include/linux/container_subsys.h |    4
 init/Kconfig                     |    8 +
 kernel/Makefile                  |    1
 kernel/container_network.c       |  285 ++++++++++++++++++++++++++++++++++++++++
 4 files changed, 298 insertions(+)

Index: 2.6-mm/include/linux/container_subsys.h
===================================================================
--- 2.6-mm.orig/include/linux/container_subsys.h
+++ 2.6-mm/include/linux/container_subsys.h
@@ -30,3 +30,7 @@
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_NETWORK
+SUBSYS(network)
+#endif
Index: 2.6-mm/init/Kconfig
===================================================================
--- 2.6-mm.orig/init/Kconfig
+++ 2.6-mm/init/Kconfig
@@ -326,6 +326,14 @@
    Provides a simple Resource Controller for monitoring the
    total CPU consumed by the tasks in a container

+config CONTAINER_NETWORK
+       bool "Network container subsystem"
+       depends on CONTAINERS && SECURITY_NETWORK
```

```
+       help
+    Provides a network controller to isolate network traffic
+
+    Say N if unsure
+
 config CPUSETS
  bool "Cpuset support"
  depends on SMP && CONTAINERS
Index: 2.6-mm/kernel/Makefile
===================================================================
--- 2.6-mm.orig/kernel/Makefile
+++ 2.6-mm/kernel/Makefile
@@ -43,6 +43,7 @@
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
 obj-$(CONFIG_CONTAINER_NS) += ns_container.o
+obj-$(CONFIG_CONTAINER_NETWORK) += container_network.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
Index: 2.6-mm/kernel/container_network.c
===================================================================
--- /dev/null
+++ 2.6-mm/kernel/container_network.c
@@ -0,0 +1,285 @@
+/*
+ * container_network.c -  container network subsystem
+ *
+ * Copyright 2006, 2007 IBM Corp
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
+#include <linux/ctype.h>
+#include <linux/list.h>
+#include <linux/spinlock.h>
+
+struct network {
+ struct container_subsys_state css;
+ struct list_head ipv4_list; /* store the IPV4 addresses */
+ rwlock_t ipv4_list_lock;
+};
+
+struct ipv4_list {
+ __be32 address;
+ struct list_head list;
```

```
+};
+
+static struct network top_network = {
+ .ipv4_list = LIST_HEAD_INIT(top_network.ipv4_list),
+ .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
+};
+
+struct container_subsys network_subsys;
+
+enum container_filetype {
+ FILE_IPV4,
+};
+
+static inline struct network *container_network(struct container *container)
+{
+ return container_of(
+  container_subsys_state(container, network_subsys_id),
+  struct network, css);
+}
+
+static struct container_subsys_state *network_create(struct container_subsys *ss,
+           struct container *container)
+{
+ struct network *network;
+
+ /* Don't let anybody do that */
+ if (!capable(CAP_NET_ADMIN))
+  return ERR_PTR(-EPERM);
+
+ /* The current container is the initial container */
+ if (!container->parent)
+  return &top_network.css;
+
+ network = kzalloc(sizeof(*network), GFP_KERNEL);
+ if (!network)
+  return ERR_PTR(-ENOMEM);
+
+ INIT_LIST_HEAD(&network->ipv4_list);
+ network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
+
+ return &network->css;
+}
+
+static void network_destroy(struct container_subsys *ss,
+       struct container *container)
+{
+ struct network *network;
+ struct ipv4_list *entry, *next;
```

```
+ struct list_head *l;
+ rwlock_t *lock;
+
+ network = container_network(container);
+ l = &network->ipv4_list;
+ lock = &network->ipv4_list_lock;
+
+ /* flush the ipv4 list */
+ write_lock(lock);
+ list_for_each_entry_safe(entry, next, l, list) {
+  list_del(&entry->list);
+  kfree(entry);
+ }
+ write_unlock(lock);
+
+ kfree(network);
+}
+
+static int network_add_ipv4_address(struct container *container, __be32 address)
+{
+ struct ipv4_list *entry;
+ struct network *network;
+
+ entry = kmalloc(sizeof(*entry), GFP_KERNEL);
+ if (!entry)
+  return -ENOMEM;
+ entry->address = address;
+
+ network = container_network(container);
+ write_lock(&network->ipv4_list_lock);
+ list_add(&entry->list, &network->ipv4_list);
+ write_unlock(&network->ipv4_list_lock);
+
+ return 0;
+}
+
+static int network_del_ipv4_address(struct container *container, __be32 address)
+{
+ struct ipv4_list *entry;
+ struct network *network;
+ int ret = 0;
+
+ network = container_network(container);
+ write_lock(&network->ipv4_list_lock);
+ list_for_each_entry(entry, &network->ipv4_list, list) {
+  if (entry->address != address)
+   continue;
+
```

```
+  list_del(&entry->list);
+  goto out_free;
+ }
+ ret = -EINVAL;
+out:
+ write_unlock(&network->ipv4_list_lock);
+ return ret;
+
+out_free:
+ kfree(entry);
+ goto out;
+}
+
+static int network_parse_ipv4_address(struct container *container, char *buffer)
+{
+ int len = strlen(buffer);
+ char *addr;
+ __be32 address;
+
+ /* remove trailing left space */
+ while(isspace(*buffer))
+  buffer++;
+
+ /* remove trailing right space */
+ while(isspace(buffer[len - 1]))
+  buffer[(len--) - 1] = 0;
+
+ len = strlen(buffer);
+        addr = memchr(buffer, ' ', len);
+ if (!addr)
+  return -EINVAL;
+ *addr++ = 0;
+
+ /* remove trailing left space again */
+ while(isspace(*addr))
+  addr++;
+
+ /* Shall I check if the address is setup on the host ? */
+ sscanf(addr, "%X", &address);
+
+ if (!strcmp(buffer, "add"))
+  return network_add_ipv4_address(container, address);
+ else if (!strcmp(buffer, "del"))
+  return network_del_ipv4_address(container, address);
+
+ return -EINVAL;
+}
+
```

```
+static int network_fill_ipv4_address(struct container *container, char *buffer)
+{
+ struct network *network;
+ struct ipv4_list *entry;
+ char *s = buffer;
+ network = container_network(container);
+
+ read_lock(&network->ipv4_list_lock);
+ list_for_each_entry(entry, &network->ipv4_list, list)
+  s += sprintf(s, "%X\n", entry->address);
+ read_unlock(&network->ipv4_list_lock);
+
+  return strlen(buffer);
+}
+
+static ssize_t network_write(struct container *container,
+        struct cftype *cft,
+        struct file *file,
+        const char __user *userbuf,
+        size_t nbytes, loff_t *unused_ppos)
+{
+ enum container_filetype type = cft->private;
+ char *buffer;
+ int retval = 0;
+
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;
+
+ if (nbytes >= PATH_MAX)
+  return -E2BIG;
+
+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
+ if (!buffer)
+  return -ENOMEM;
+
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+  retval = -EFAULT;
+  goto out_free;
+ }
+ buffer[nbytes] = 0;
+
+ container_lock();
+ switch(type) {
+
+ case FILE_IPV4:
+  retval = network_parse_ipv4_address(container, buffer);
+  break;
+
```

```
+ default:
+  retval = -EINVAL;
+  break;
+ };
+ container_unlock();
+
+out_free:
+ if (!retval)
+  retval = nbytes;
+
+ kfree(buffer);
+ return retval;
+}
+
+static ssize_t network_read(struct container *container,
+       struct cftype *cft,
+       struct file *file,
+       char __user *userbuf,
+       size_t nbytes, loff_t *ppos)
+{
+ enum container_filetype type = cft->private;
+ char *page;
+ int retval;
+
+ page = (char *)__get_free_page(GFP_TEMPORARY);
+ if (!page)
+  return -ENOMEM;
+
+ container_lock();
+ switch(type) {
+ case FILE_IPV4:
+  retval = network_fill_ipv4_address(container, page);
+  break;
+
+ default:
+  retval = -EINVAL;
+ };
+ container_unlock();
+
+ retval = simple_read_from_buffer(userbuf, nbytes, ppos, page, retval);
+
+ free_page((unsigned long)page);
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "ipv4",
```

```
+  .read = network_read,
+  .write = network_write,
+  .private = FILE_IPV4,
+ },
+};
+
+static int network_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_files(cont, ss, files, ARRAY_SIZE(files));
+}
+
+struct container_subsys network_subsys = {
+ .name = "network",
+ .create = network_create,
+ .destroy = network_destroy,
+ .populate = network_populate,
+ .subsys_id = network_subsys_id,
+ .can_attach = NULL,
+ .attach = NULL,
+ .fork = NULL,
+ .exit = NULL,
+};

--
```

Subject: [RFC][patch 2/3] network security hooks
Posted by Daniel Lezcano on Tue, 04 Sep 2007 17:00:24 GMT
View Forum Message <> Reply to Message

From: Daniel Lezcano <dlezcano@fr.ibm.com>

After all, the network security hooks are placed exactly at the places we need.
This patch plugs the network security hooks with network container subsystem.

The hooks always do nothing when they are called from a process running inside
the root container.

The security hooks are not activated by default when the root container is created,
I let the first child container to do that, that ensure correct kernel initialization.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 kernel/container_network.c |  148 ++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 148 insertions(+)

Index: 2.6-mm/kernel/container_network.c
===================================================================
--- 2.6-mm.orig/kernel/container_network.c
+++ 2.6-mm/kernel/container_network.c
@@ -11,6 +11,7 @@
 #include <linux/ctype.h>
 #include <linux/list.h>
 #include <linux/spinlock.h>
+#include <linux/security.h>

 struct network {
  struct container_subsys_state css;
@@ -28,12 +29,21 @@
  .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
 };

+static int security_registered;
+static int secondary;
+
 struct container_subsys network_subsys;

 enum container_filetype {
  FILE_IPV4,
 };

+static inline struct network *task_network(struct task_struct *task)
+{
+ return container_of(task_subsys_state(task, network_subsys_id),
+       struct network, css);
+}
+
 static inline struct network *container_network(struct container *container)
 {
  return container_of(
@@ -41,6 +51,125 @@
   struct network, css);
 }

+static int network_socket_create(int family, int type, int protocol, int kern)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
```

```
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_post_create(struct socket *sock, int family,
+          int type, int protocol, int kern)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_bind(struct socket *sock,
+          struct sockaddr *address,
+          int addrlen)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_connect(struct socket * sock,
+       struct sockaddr * address,
+       int addrlen)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_listen(struct socket * sock, int backlog)
+{
+ struct network *network;
+
+ network = task_network(current);
```

```
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_accept(struct socket *sock,
+     struct socket *newsock)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static void network_socket_post_accept(struct socket *sock,
+           struct socket *newsock)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return;
+}
+
+static int network_socket_sendmsg(struct socket *sock,
+      struct msghdr *msg, int size)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
+  return 0;
+
+ return 0;
+}
+
+static int network_socket_recvmsg(struct socket *sock,
+      struct msghdr *msg, int size,
+      int flags)
+{
+ struct network *network;
+
+ network = task_network(current);
+ if (!network || network == &top_network)
```

```
+  return 0;
+
+ return 0;
+}
+
+static struct security_operations network_security_ops = {
+ .socket_create = network_socket_create,
+ .socket_post_create = network_socket_post_create,
+ .socket_bind = network_socket_bind,
+ .socket_connect = network_socket_connect,
+ .socket_listen = network_socket_listen,
+ .socket_accept = network_socket_accept,
+ .socket_post_accept = network_socket_post_accept,
+ .socket_sendmsg = network_socket_sendmsg,
+ .socket_recvmsg = network_socket_recvmsg,
+};
+
 static struct container_subsys_state *network_create(struct container_subsys *ss,
           struct container *container)
 {
@@ -61,6 +190,25 @@
  INIT_LIST_HEAD(&network->ipv4_list);
  network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);

+ /*
+  * register the network security hooks only one time
+  * after the root container is created, the first non
+  * root container has the assignment to register the
+  * security hooks
+  *
+  */
+ if (!security_registered) {
+  if (register_security(&network_security_ops)) {
+   if (mod_reg_security(KBUILD_MODNAME,
+         &network_security_ops)) {
+    kfree(network);
+    return ERR_PTR(-EINVAL);
+   }
+   secondary = 1;
+  }
+  security_registered = 1;
+ }
+
  return &network->css;
 }


--
```

Subject: [RFC][patch 3/3] activate filtering for the bind
Posted by Daniel Lezcano on Tue, 04 Sep 2007 17:00:25 GMT
View Forum Message <> Reply to Message

From: Daniel Lezcano <dlezcano@fr.ibm.com>

For the moment, I only made this patch for the RFC. It shows how simple it is
to hook different socket syscalls. This patch denies bind to any addresses
which are not in the container IPV4 address list, except for the INADDR_ANY.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

---
 kernel/container_network.c | 66 ++++++++++++++++++++++----------------------
 1 file changed, 35 insertions(+), 31 deletions(-)

Index: 2.6-mm/kernel/container_network.c
===
--- 2.6-mm.orig/kernel/container_network.c
+++ 2.6-mm/kernel/container_network.c
@@ -12,6 +12,9 @@
 #include <linux/list.h>
 #include <linux/spinlock.h>
 #include <linux/security.h>
+#include <linux/in.h>
+#include <linux/net.h>
+#include <linux/socket.h>

 struct network {
   struct container_subsys_state css;
@@ -53,24 +56,14 @@

 static int network_socket_create(int family, int type, int protocol, int kern)
 {
- struct network *network;
-
- network = task_network(current);
- if (!network || network == &top_network)
-   return 0;
-
+ /* nothing to do right now */
   return 0;

```
 }

 static int network_socket_post_create(struct socket *sock, int family,
          int type, int protocol, int kern)
 {
- struct network *network;
-
- network = task_network(current);
- if (!network || network == &top_network)
-  return 0;
-
+ /* nothing to do right now */
  return 0;
 }

@@ -79,47 +72,58 @@
          int addrlen)
 {
  struct network *network;
+ struct list_head *l;
+ rwlock_t *lock;
+ struct ipv4_list *entry;
+ __be32 addr;
+ int ret = -EPERM;

+ /* Do nothing for the root container */
  network = task_network(current);
  if (!network || network == &top_network)
   return 0;

- return 0;
+ /* Check we have to do some filtering */
+ if (sock->ops->family != AF_INET)
+  return 0;
+
+ l = &network->ipv4_list;
+ lock = &network->ipv4_list_lock;
+ addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
+
+ if (addr == INADDR_ANY)
+  return 0;
+
+ read_lock(lock);
+ list_for_each_entry(entry, l, list) {
+  if (entry->address != addr)
+   continue;
+  ret = 0;
+  break;
```

```
+ }
+ read_unlock(lock);
+
+ return ret;
 }

 static int network_socket_connect(struct socket * sock,
       struct sockaddr * address,
       int addrlen)
 {
- struct network *network;
-
- network = task_network(current);
- if (!network || network == &top_network)
-  return 0;
-
+ /* nothing to do right now */
  return 0;
 }

 static int network_socket_listen(struct socket * sock, int backlog)
 {
- struct network *network;
-
- network = task_network(current);
- if (!network || network == &top_network)
-  return 0;
-
+ /* nothing to do right now */
  return 0;
 }

 static int network_socket_accept(struct socket *sock,
      struct socket *newsock)
 {
- struct network *network;
-
- network = task_network(current);
- if (!network || network == &top_network)
-  return 0;
-
+ /* nothing to do right now */
  return 0;
 }


--
_____________________________________________________
```

Subject: Re: [RFC][patch 1/3] network container subsystem
Posted by Benjamin Thery on Wed, 05 Sep 2007 11:50:13 GMT
View Forum Message <> Reply to Message

dlezcano@fr.ibm.com wrote:
> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> This patch creates the network container subsystem.
> It consists for the moment on a single file "network.ipv4".
>
> The interface is pretty simple:
>
> To add an IP address to the container:
>
> echo add AB12FFFF > network.ipv4
>
> To remove this IP address:
> -------------------------
>
> echo del AB12FFFF > network.ipv4
>
> To list the addresses:
> ----------------------
>
> cat network.ipv4
>
> The parameter is an IPV4 address in the hexa format. The parsing of a dotted-decimal
> parameter is totally painful. If this format hurts someone, I can change it to a dotted
> format at the risk of having something buggy.

I think passing ipv4 addresses in hexa form is painful :)
Why not use something like this:

```
  __be32 addr;
  unsigned int a, b, c, d;

  if (sscanf (buffer, "%u.%u.%u.%u", &a, &b, &c, &d) == 4) {
     if (a < 256 && b < 256 && c < 256 && d < 256) {
        addr = htonl (a<<24 | b<<16 | c<<8 | d);
  return 0;
    }
  }
  return -EINVAL;
```

>
> This patch by itself does nothing more than adding/removing elements from a list.
>
> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> ---
> include/linux/container_subsys.h | 4
> init/Kconfig | 8 +
> kernel/Makefile | 1
> kernel/container_network.c | 285 ++
> 4 files changed, 298 insertions(+)
>
> Index: 2.6-mm/include/linux/container_subsys.h
> ===
> --- 2.6-mm.orig/include/linux/container_subsys.h
> +++ 2.6-mm/include/linux/container_subsys.h
> @@ -30,3 +30,7 @@
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CONTAINER_NETWORK
> +SUBSYS(network)
> +#endif
> Index: 2.6-mm/init/Kconfig
> ===
> --- 2.6-mm.orig/init/Kconfig
> +++ 2.6-mm/init/Kconfig
> @@ -326,6 +326,14 @@
>   Provides a simple Resource Controller for monitoring the
>   total CPU consumed by the tasks in a container
>
> +config CONTAINER_NETWORK
> +   bool "Network container subsystem"
> +   depends on CONTAINERS && SECURITY_NETWORK
> +   help
> +   Provides a network controller to isolate network traffic
> +
> +   Say N if unsure
> +
> config CPUSETS
>   bool "Cpuset support"
>   depends on SMP && CONTAINERS
> Index: 2.6-mm/kernel/Makefile
> ===
> --- 2.6-mm.orig/kernel/Makefile
> +++ 2.6-mm/kernel/Makefile

> @@ -43,6 +43,7 @@
> obj-$(CONFIG_CPUSETS) += cpuset.o
> obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
> obj-$(CONFIG_CONTAINER_NS) += ns_container.o
> +obj-$(CONFIG_CONTAINER_NETWORK) += container_network.o
> obj-$(CONFIG_IKCONFIG) += configs.o
> obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
> obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
> Index: 2.6-mm/kernel/container_network.c
> ===
> --- /dev/null
> +++ 2.6-mm/kernel/container_network.c
> @@ -0,0 +1,285 @@
> +/*
> + * container_network.c - container network subsystem
> + *
> + * Copyright 2006, 2007 IBM Corp
> + */
> +
> +#include <linux/module.h>
> +#include <linux/container.h>
> +#include <linux/fs.h>
> +#include <linux/uaccess.h>
> +#include <linux/ctype.h>
> +#include <linux/list.h>
> +#include <linux/spinlock.h>
> +
> +struct network {
> + struct container_subsys_state css;
> + struct list_head ipv4_list; /* store the IPV4 addresses */
> + rwlock_t ipv4_list_lock;
> +};
> +
> +struct ipv4_list {
> + __be32 address;
> + struct list_head list;
> +};

Can we use "struct in_addr" instead of __be32?
(may be you don't want to include in.h?)

> +
> +static struct network top_network = {
> + .ipv4_list = LIST_HEAD_INIT(top_network.ipv4_list),
> + .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
> +};
> +

```
> +struct container_subsys network_subsys;
> +
> +enum container_filetype {
> + FILE_IPV4,
> +};
> +
> +static inline struct network *container_network(struct container *container)
> +{
> + return container_of(
> +  container_subsys_state(container, network_subsys_id),
> +  struct network, css);
> +}
> +
> +static struct container_subsys_state *network_create(struct container_subsys *ss,
> +           struct container *container)
> +{
> + struct network *network;
> +
> + /* Don't let anybody do that */
> + if (!capable(CAP_NET_ADMIN))
> +  return ERR_PTR(-EPERM);
> +
> + /* The current container is the initial container */
> + if (!container->parent)
> +  return &top_network.css;
> +
> + network = kzalloc(sizeof(*network), GFP_KERNEL);
> + if (!network)
> +  return ERR_PTR(-ENOMEM);
> +
> + INIT_LIST_HEAD(&network->ipv4_list);
> + network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
> +
> + return &network->css;
> +}
> +
> +static void network_destroy(struct container_subsys *ss,
> +       struct container *container)
> +{
> + struct network *network;
> + struct ipv4_list *entry, *next;
> + struct list_head *l;
> + rwlock_t *lock;
> +
> + network = container_network(container);
> + l = &network->ipv4_list;
> + lock = &network->ipv4_list_lock;
> +
```

```
> + /* flush the ipv4 list */
> + write_lock(lock);
> + list_for_each_entry_safe(entry, next, l, list) {
> +  list_del(&entry->list);
> +  kfree(entry);
> + }
> + write_unlock(lock);
> +
> + kfree(network);
> +}
> +
> +static int network_add_ipv4_address(struct container *container, __be32 address)
> +{
> + struct ipv4_list *entry;
> + struct network *network;
> +
> + entry = kmalloc(sizeof(*entry), GFP_KERNEL);
> + if (!entry)
> +  return -ENOMEM;
> + entry->address = address;
> +
> + network = container_network(container);
> + write_lock(&network->ipv4_list_lock);
> + list_add(&entry->list, &network->ipv4_list);
> + write_unlock(&network->ipv4_list_lock);
> +
> + return 0;
> +}
> +
> +static int network_del_ipv4_address(struct container *container, __be32 address)
> +{
> + struct ipv4_list *entry;
> + struct network *network;
> + int ret = 0;
> +
> + network = container_network(container);
> + write_lock(&network->ipv4_list_lock);
> + list_for_each_entry(entry, &network->ipv4_list, list) {
> +  if (entry->address != address)
> +   continue;
> +
> +  list_del(&entry->list);
> +  goto out_free;
> + }
> + ret = -EINVAL;
> +out:
> + write_unlock(&network->ipv4_list_lock);
> + return ret;
```

```
> +
> +out_free:
> + kfree(entry);
> + goto out;
> +}
> +
> +static int network_parse_ipv4_address(struct container *container, char *buffer)
> +{
> + int len = strlen(buffer);
> + char *addr;
> + __be32 address;
> +
> + /* remove trailing left space */
```

May be "leading space" is better than "trailing left space" :)

```
> + while(isspace(*buffer))
> +  buffer++;
> +
> + /* remove trailing right space */
> + while(isspace(buffer[len - 1]))
> +  buffer[(len--) - 1] = 0;
> +
> + len = strlen(buffer);
> +        addr = memchr(buffer, ' ', len);
> + if (!addr)
> +  return -EINVAL;
> + *addr++ = 0;
> +
> + /* remove trailing left space again */
> + while(isspace(*addr))
> +  addr++;
> +
> + /* Shall I check if the address is setup on the host ? */
> + sscanf(addr, "%X", &address);
> +
> + if (!strcmp(buffer, "add"))
> +  return network_add_ipv4_address(container, address);
> + else if (!strcmp(buffer, "del"))
> +  return network_del_ipv4_address(container, address);
> +
> + return -EINVAL;
> +}
> +
> +static int network_fill_ipv4_address(struct container *container, char *buffer)
> +{
> + struct network *network;
```

```
> + struct ipv4_list *entry;
> + char *s = buffer;
> + network = container_network(container);
> +
> + read_lock(&network->ipv4_list_lock);
> + list_for_each_entry(entry, &network->ipv4_list, list)
> +  s += sprintf(s, "%X\n", entry->address);
```

Pretty print:

```
 s+= sprintf(s, NIPQUAD_FMT "\n", NIPQUAD(entry->address));
```

```
> + read_unlock(&network->ipv4_list_lock);
> +
> +  return strlen(buffer);
> +}
> +
> +static ssize_t network_write(struct container *container,
> +        struct cftype *cft,
> +        struct file *file,
> +        const char __user *userbuf,
> +        size_t nbytes, loff_t *unused_ppos)
> +{
> + enum container_filetype type = cft->private;
> + char *buffer;
> + int retval = 0;
> +
> + if (!capable(CAP_NET_ADMIN))
> +  return -EPERM;
> +
> + if (nbytes >= PATH_MAX)
> +  return -E2BIG;
> +
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (!buffer)
> +  return -ENOMEM;
> +
> + if (copy_from_user(buffer, userbuf, nbytes)) {
> +  retval = -EFAULT;
> +  goto out_free;
> + }
> + buffer[nbytes] = 0;
> +
> + container_lock();
> + switch(type) {
> +
> + case FILE_IPV4:
```

```
> +  retval = network_parse_ipv4_address(container, buffer);
> +  break;
> +
> + default:
> +  retval = -EINVAL;
> +  break;
> + };
> + container_unlock();
> +
> +out_free:
> + if (!retval)
> +  retval = nbytes;
> +
> + kfree(buffer);
> + return retval;
> +}
> +
> +static ssize_t network_read(struct container *container,
> +       struct cftype *cft,
> +       struct file *file,
> +       char __user *userbuf,
> +       size_t nbytes, loff_t *ppos)
> +{
> + enum container_filetype type = cft->private;
> + char *page;
> + int retval;
> +
> + page = (char *)__get_free_page(GFP_TEMPORARY);
> + if (!page)
> +  return -ENOMEM;
> +
> + container_lock();
> + switch(type) {
> + case FILE_IPV4:
> +  retval = network_fill_ipv4_address(container, page);
> +  break;
> +
> + default:
> +  retval = -EINVAL;
> + };
> + container_unlock();
> +
> + retval = simple_read_from_buffer(userbuf, nbytes, ppos, page, retval);
> +
> + free_page((unsigned long)page);
> + return retval;
> +}
> +
```

> +static struct cftype files[] = {
> + {
> + .name = "ipv4",
> + .read = network_read,
> + .write = network_write,
> + .private = FILE_IPV4,
> + },
> +};
> +
> +static int network_populate(struct container_subsys *ss, struct container *cont)
> +{
> + return container_add_files(cont, ss, files, ARRAY_SIZE(files));
> +}
> +
> +struct container_subsys network_subsys = {
> + .name = "network",
> + .create = network_create,
> + .destroy = network_destroy,
> + .populate = network_populate,
> + .subsys_id = network_subsys_id,
> + .can_attach = NULL,
> + .attach = NULL,
> + .fork = NULL,
> + .exit = NULL,
> +};
>
> -- _____ Containers mailing list
Containers@lists.linux-foundation.org https://lists.linux-foundation.org/mailman/listinfo/containers

--
B e n j a m i n T h e r y - BULL/DT/Open Software R&D

http://www.bull.com

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

---

Subject: Re: [RFC][patch 1/3] network container subsystem
Posted by Daniel Lezcano on Wed, 05 Sep 2007 12:43:32 GMT
View Forum Message <> Reply to Message

Benjamin Thery wrote:
> dlezcano@fr.ibm.com wrote:
>> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>>

>> This patch creates the network container subsystem. It consists for
>> the moment on a single file "network.ipv4".
>>
>> The interface is pretty simple:
>>
>> To add an IP address to the container:
>>
>>   echo add AB12FFFF > network.ipv4
>>
>> To remove this IP address:
>> -------------------------
>>
>>   echo del AB12FFFF > network.ipv4
>>
>> To list the addresses:
>> ----------------------
>>
>>   cat network.ipv4
>>
>> The parameter is an IPV4 address in the hexa format. The parsing of a
>> dotted-decimal
>> parameter is totally painful. If this format hurts someone, I can
>> change it to a dotted
>> format at the risk of having something buggy.
>
> I think passing ipv4 addresses in hexa form is painful :)
> Why not use something like this:
>
>   __be32 addr;
>   unsigned int a, b, c, d;
>
>   if (sscanf (buffer, "%u.%u.%u.%u", &a, &b, &c, &d) == 4) {
>   if (a < 256 && b < 256 && c < 256 && d < 256) {
>   addr = htonl (a<<24 | b<<16 | c<<8 | d);
>   return 0;
>   }
>   }
>   return -EINVAL;
>

I didn't think about this format, I will change it.

>>
>> This patch by itself does nothing more than adding/removing elements
>> from a list.
>>
>> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>>

```
>> ---
>>  include/linux/container_subsys.h |    4
>>  init/Kconfig                     |    8 +
>>  kernel/Makefile                  |    1
>>  kernel/container_network.c       |  285
>> ++++++++++++++++++++++++++++++++++++++
>>  4 files changed, 298 insertions(+)
>>
>> Index: 2.6-mm/include/linux/container_subsys.h
>> ===================================================================
>> --- 2.6-mm.orig/include/linux/container_subsys.h
>> +++ 2.6-mm/include/linux/container_subsys.h
>> @@ -30,3 +30,7 @@
>>  #endif
>>
>>  /* */
>> +
>> +#ifdef CONFIG_CONTAINER_NETWORK
>> +SUBSYS(network)
>> +#endif
>> Index: 2.6-mm/init/Kconfig
>> ===================================================================
>> --- 2.6-mm.orig/init/Kconfig
>> +++ 2.6-mm/init/Kconfig
>> @@ -326,6 +326,14 @@
>>        Provides a simple Resource Controller for monitoring the
>>        total CPU consumed by the tasks in a container
>>
>> +config CONTAINER_NETWORK
>> +       bool "Network container subsystem"
>> +       depends on CONTAINERS && SECURITY_NETWORK
>> +       help
>> +       Provides a network controller to isolate network traffic
>> +
>> +       Say N if unsure
>> +
>>  config CPUSETS
>>      bool "Cpuset support"
>>      depends on SMP && CONTAINERS
>> Index: 2.6-mm/kernel/Makefile
>> ===================================================================
>> --- 2.6-mm.orig/kernel/Makefile
>> +++ 2.6-mm/kernel/Makefile
>> @@ -43,6 +43,7 @@
>>  obj-$(CONFIG_CPUSETS) += cpuset.o
>>  obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
>>  obj-$(CONFIG_CONTAINER_NS) += ns_container.o
>> +obj-$(CONFIG_CONTAINER_NETWORK) += container_network.o
```

>> obj-$(CONFIG_IKCONFIG) += configs.o
>> obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
>> obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
>> Index: 2.6-mm/kernel/container_network.c
>> ===
>> --- /dev/null
>> +++ 2.6-mm/kernel/container_network.c
>> @@ -0,0 +1,285 @@
>> +/*
>> + * container_network.c - container network subsystem
>> + *
>> + * Copyright 2006, 2007 IBM Corp
>> + */
>> +
>> +#include <linux/module.h>
>> +#include <linux/container.h>
>> +#include <linux/fs.h>
>> +#include <linux/uaccess.h>
>> +#include <linux/ctype.h>
>> +#include <linux/list.h>
>> +#include <linux/spinlock.h>
>> +
>> +struct network {
>> +   struct container_subsys_state css;
>> +   struct list_head ipv4_list; /* store the IPV4 addresses */
>> +   rwlock_t ipv4_list_lock;
>> +};
>> +
>> +struct ipv4_list {
>> +   __be32 address;
>> +   struct list_head list;
>> +};
>
> Can we use "struct in_addr" instead of __be32?
> (may be you don't want to include in.h?)

Perhaps struct sockaddr is more adequate. We can just use a list for
storing ipv4, ipv6, ...

>> +
>> +static struct network top_network = {
>> +   .ipv4_list = LIST_HEAD_INIT(top_network.ipv4_list),
>> +   .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
>> +};
>> +
>> +struct container_subsys network_subsys;
>> +
>> +enum container_filetype {

```
>> +    FILE_IPV4,
>> +};
>> +
>> +static inline struct network *container_network(struct container
>> *container)
>> +{
>> +    return container_of(
>> +        container_subsys_state(container, network_subsys_id),
>> +        struct network, css);
>> +}
>> +
>> +static struct container_subsys_state *network_create(struct
>> container_subsys *ss,
>> +                            struct container *container)
>> +{
>> +    struct network *network;
>> +
>> +    /* Don't let anybody do that */
>> +    if (!capable(CAP_NET_ADMIN))
>> +        return ERR_PTR(-EPERM);
>> +
>> +    /* The current container is the initial container */
>> +    if (!container->parent)
>> +        return &top_network.css;
>> +
>> +    network = kzalloc(sizeof(*network), GFP_KERNEL);
>> +    if (!network)
>> +        return ERR_PTR(-ENOMEM);
>> +
>> +    INIT_LIST_HEAD(&network->ipv4_list);
>> +    network->ipv4_list_lock =
>> __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
>> +
>> +    return &network->css;
>> +}
>> +
>> +static void network_destroy(struct container_subsys *ss,
>> +                struct container *container)
>> +{
>> +    struct network *network;
>> +    struct ipv4_list *entry, *next;
>> +    struct list_head *l;
>> +    rwlock_t *lock;
>> +
>> +    network = container_network(container);
>> +    l = &network->ipv4_list;
>> +    lock = &network->ipv4_list_lock;
>> +
```

```
>> +    /* flush the ipv4 list */
>> +    write_lock(lock);
>> +    list_for_each_entry_safe(entry, next, l, list) {
>> +        list_del(&entry->list);
>> +        kfree(entry);
>> +    }
>> +    write_unlock(lock);
>> +
>> +    kfree(network);
>> +}
>> +
>> +static int network_add_ipv4_address(struct container *container,
>> __be32 address)
>> +{
>> +    struct ipv4_list *entry;
>> +    struct network *network;
>> +
>> +    entry = kmalloc(sizeof(*entry), GFP_KERNEL);
>> +    if (!entry)
>> +        return -ENOMEM;
>> +    entry->address = address;
>> +
>> +    network = container_network(container);
>> +    write_lock(&network->ipv4_list_lock);
>> +    list_add(&entry->list, &network->ipv4_list);
>> +    write_unlock(&network->ipv4_list_lock);
>> +
>> +    return 0;
>> +}
>> +
>> +static int network_del_ipv4_address(struct container *container,
>> __be32 address)
>> +{
>> +    struct ipv4_list *entry;
>> +    struct network *network;
>> +    int ret = 0;
>> +
>> +    network = container_network(container);
>> +    write_lock(&network->ipv4_list_lock);
>> +    list_for_each_entry(entry, &network->ipv4_list, list) {
>> +        if (entry->address != address)
>> +            continue;
>> +
>> +        list_del(&entry->list);
>> +        goto out_free;
>> +    }
>> +    ret = -EINVAL;
>> +out:
```

```
>> +    write_unlock(&network->ipv4_list_lock);
>> +    return ret;
>> +
>> +out_free:
>> +    kfree(entry);
>> +    goto out;
>> +}
>> +
>> +static int network_parse_ipv4_address(struct container *container,
>> char *buffer)
>> +{
>> +    int len = strlen(buffer);
>> +    char *addr;
>> +    __be32 address;
>> +
>> +    /* remove trailing left space */
>
> May be "leading space" is better than "trailing left space" :)

... yes :D
>
>
>> +    while(isspace(*buffer))
>> +        buffer++;
>> +
>> +    /* remove trailing right space */
>> +    while(isspace(buffer[len - 1]))
>> +        buffer[(len--) - 1] = 0;
>> +
>> +    len = strlen(buffer);
>> +          addr = memchr(buffer, ' ', len);
>> +    if (!addr)
>> +        return -EINVAL;
>> +    *addr++ = 0;
>> +
>> +    /* remove trailing left space again */
>> +    while(isspace(*addr))
>> +        addr++;
>> +
>> +    /* Shall I check if the address is setup on the host ? */
>> +    sscanf(addr, "%X", &address);
>> +
>> +    if (!strcmp(buffer, "add"))
>> +        return network_add_ipv4_address(container, address);
>> +    else if (!strcmp(buffer, "del"))
>> +        return network_del_ipv4_address(container, address);
>> +
>> +    return -EINVAL;
```

```
>> +}
>> +
>> +static int network_fill_ipv4_address(struct container *container,
>> char *buffer)
>> +{
>> +    struct network *network;
>> +    struct ipv4_list *entry;
>> +    char *s = buffer;
>> +    network = container_network(container);
>> +
>> +    read_lock(&network->ipv4_list_lock);
>> +    list_for_each_entry(entry, &network->ipv4_list, list)
>> +        s += sprintf(s, "%X\n", entry->address);
>
> Pretty print:
>
>     s+= sprintf(s, NIPQUAD_FMT "\n", NIPQUAD(entry->address));
```

Thanks.

```
>
>
>> +    read_unlock(&network->ipv4_list_lock);
>> +
>> +     return strlen(buffer);
>> +}
>> +
>> +static ssize_t network_write(struct container *container,
>> +                struct cftype *cft,
>> +                struct file *file,
>> +                const char __user *userbuf,
>> +                size_t nbytes, loff_t *unused_ppos)
>> +{
>> +    enum container_filetype type = cft->private;
>> +    char *buffer;
>> +    int retval = 0;
>> +
>> +    if (!capable(CAP_NET_ADMIN))
>> +        return -EPERM;
>> +
>> +    if (nbytes >= PATH_MAX)
>> +        return -E2BIG;
>> +
>> +    buffer = kmalloc(nbytes + 1, GFP_KERNEL);
>> +    if (!buffer)
>> +        return -ENOMEM;
>> +
>> +    if (copy_from_user(buffer, userbuf, nbytes)) {
>> +        retval = -EFAULT;
```

```
>> +        goto out_free;
>> +    }
>> +    buffer[nbytes] = 0;
>> +
>> +    container_lock();
>> +    switch(type) {
>> +
>> +    case FILE_IPV4:
>> +        retval = network_parse_ipv4_address(container, buffer);
>> +        break;
>> +
>> +    default:
>> +        retval = -EINVAL;
>> +        break;
>> +    };
>> +    container_unlock();
>> +
>> +out_free:
>> +    if (!retval)
>> +        retval = nbytes;
>> +
>> +    kfree(buffer);
>> +    return retval;
>> +}
>> +
>> +static ssize_t network_read(struct container *container,
>> +                struct cftype *cft,
>> +                struct file *file,
>> +                char __user *userbuf,
>> +                size_t nbytes, loff_t *ppos)
>> +{
>> +    enum container_filetype type = cft->private;
>> +    char *page;
>> +    int retval;
>> +
>> +    page = (char *)__get_free_page(GFP_TEMPORARY);
>> +    if (!page)
>> +        return -ENOMEM;
>> +
>> +    container_lock();
>> +    switch(type) {
>> +    case FILE_IPV4:
>> +        retval = network_fill_ipv4_address(container, page);
>> +        break;
>> +
>> +    default:
>> +        retval = -EINVAL;
>> +    };
```

```
>> +    container_unlock();
>> +
>> +    retval = simple_read_from_buffer(userbuf, nbytes, ppos, page,
>> retval);
>> +
>> +    free_page((unsigned long)page);
>> +    return retval;
>> +}
>> +
>> +static struct cftype files[] = {
>> +    {
>> +        .name = "ipv4",
>> +        .read = network_read,
>> +        .write = network_write,
>> +        .private = FILE_IPV4,
>> +    },
>> +};
>> +
>> +static int network_populate(struct container_subsys *ss, struct
>> container *cont)
>> +{
>> +    return container_add_files(cont, ss, files, ARRAY_SIZE(files));
>> +}
>> +
>> +struct container_subsys network_subsys = {
>> +    .name = "network",
>> +    .create     = network_create,
>> +    .destroy = network_destroy,
>> +    .populate = network_populate,
>> +    .subsys_id = network_subsys_id,
>> +    .can_attach = NULL,
>> +    .attach     = NULL,
>> +    .fork = NULL,
>> +    .exit = NULL,
>> +};
>>
>> -- _____ Containers mailing
>> list Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>
>
```

_____

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

Subject: Re: [RFC][patch 0/3] Network container subsystem - bind filtering
Posted by Benjamin Thery on Wed, 05 Sep 2007 14:05:22 GMT
View Forum Message <> Reply to Message

The patchset looks fine to me.
This looks like a nice and simple way to obtain basic IP isolation.

Benjamin

dlezcano@fr.ibm.com wrote:
> Paul Menage mentionned, a few weeks ago, he wanted a bind filtering
> for containers. Here it is :)
>
> The following patches are a proposition to bring IP isolation to a container.
>
> After looking more closely at the code I found that security hooks are
> at the right place to catch socket calls. The IP isolation relies on the
> security hooks and that has the advantage of not having the kernel code modified,
> (expect container.h and makefile/kconfig), the patchset provide just a new
> file container_network.c
>
> Roughly, a container has a subsystem for the network (only ipv4).
> This subsystem contains the list of the addresses allowed to be used by the
> container. If a container tries to bind to an address not contained into
> this list, the bind will fail with EPERM. Of course the bind is allowed to
> INADDR_ANY.
>
> If this approach is ok for everyone, I can extend the bind filtering to
> consolidate the IP isolation.
>
> Regards.
>

--
B e n j a m i n T h e r y - BULL/DT/Open Software R&D

http://www.bull.com

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

Subject: Re: [RFC][patch 0/3] Network container subsystem - bind filtering
Posted by serue on Wed, 05 Sep 2007 15:37:41 GMT
View Forum Message <> Reply to Message

Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
> Paul Menage mentionned, a few weeks ago, he wanted a bind filtering
> for containers. Here it is :)
>
> The following patches are a proposition to bring IP isolation to a container.
>
> After looking more closely at the code I found that security hooks are
> at the right place to catch socket calls. The IP isolation relies on the
> security hooks and that has the advantage of not having the kernel code modified,
> (expect container.h and makefile/kconfig), the patchset provide just a new
> file container_network.c
>
> Roughly, a container has a subsystem for the network (only ipv4).

Just curious - why ipv4 only? When i wrote the bsdjail lsm, using the
same approach, doing ipv6 address was pretty simple. Did something
change? Or is ipv4 just a temporary restriction while you prototype?

> This subsystem contains the list of the addresses allowed to be used by the
> container. If a container tries to bind to an address not contained into
> this list, the bind will fail with EPERM. Of course the bind is allowed to
> INADDR_ANY.
>
> If this approach is ok for everyone, I can extend the bind filtering to
> consolidate the IP isolation.

You'll want to cc: the linux-security-module@vger.kernel.org list on
this patchset.

thanks,
-serge

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

---

Subject: Re: [RFC][patch 0/3] Network container subsystem - bind filtering
Posted by Daniel Lezcano on Wed, 05 Sep 2007 15:41:03 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
>> Paul Menage mentionned, a few weeks ago, he wanted a bind filtering
>> for containers. Here it is :)
>>
>> The following patches are a proposition to bring IP isolation to a container.
>>

>> After looking more closely at the code I found that security hooks are
>> at the right place to catch socket calls. The IP isolation relies on the
>> security hooks and that has the advantage of not having the kernel code modified,
>> (expect container.h and makefile/kconfig), the patchset provide just a new
>> file container_network.c
>>
>> Roughly, a container has a subsystem for the network (only ipv4).
>
> Just curious - why ipv4 only? When i wrote the bsdjail lsm, using the
> same approach, doing ipv6 address was pretty simple. Did something
> change? Or is ipv4 just a temporary restriction while you prototype?

It is a temporary restriction for the prototype.
>
>> This subsystem contains the list of the addresses allowed to be used by the
>> container. If a container tries to bind to an address not contained into
>> this list, the bind will fail with EPERM. Of course the bind is allowed to
>> INADDR_ANY.
>>
>> If this approach is ok for everyone, I can extend the bind filtering to
>> consolidate the IP isolation.
>
> You'll want to cc: the linux-security-module@vger.kernel.org list on
> this patchset.

Sure.

> thanks,
> -serge
>

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

---

Subject: Re: [RFC][patch 1/3] network container subsystem
Posted by serue on Wed, 05 Sep 2007 15:49:50 GMT
View Forum Message <> Reply to Message

Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> This patch creates the network container subsystem.
> It consists for the moment on a single file "network.ipv4".
>
> The interface is pretty simple:

>
> To add an IP address to the container:
>
> echo add AB12FFFF > network.ipv4
>
> To remove this IP address:
> ------------------------
>
> echo del AB12FFFF > network.ipv4
>
> To list the addresses:
> ---------------------
>
> cat network.ipv4
>
> The parameter is an IPV4 address in the hexa format. The parsing of a dotted-decimal
> parameter is totally painful. If this format hurts someone, I can change it to a dotted
> format at the risk of having something buggy.
>
> This patch by itself does nothing more than adding/removing elements from a list.
>
> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> ---
> include/linux/container_subsys.h | 4
> init/Kconfig | 8 +
> kernel/Makefile | 1
> kernel/container_network.c | 285 ++
> 4 files changed, 298 insertions(+)
>
> Index: 2.6-mm/include/linux/container_subsys.h
> ===
> --- 2.6-mm.orig/include/linux/container_subsys.h
> +++ 2.6-mm/include/linux/container_subsys.h
> @@ -30,3 +30,7 @@
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CONTAINER_NETWORK
> +SUBSYS(network)
> +#endif

I think Paul wants the /* */ after your endif as well to reduce patch
conflicts.

> Index: 2.6-mm/init/Kconfig
> ===

> --- 2.6-mm.orig/init/Kconfig
> +++ 2.6-mm/init/Kconfig
> @@ -326,6 +326,14 @@
>   Provides a simple Resource Controller for monitoring the
>   total CPU consumed by the tasks in a container
>
> +config CONTAINER_NETWORK
> +   bool "Network container subsystem"
> +   depends on CONTAINERS && SECURITY_NETWORK
> +   help
> +   Provides a network controller to isolate network traffic
> +
> +   Say N if unsure
> +
> config CPUSETS
>   bool "Cpuset support"
>   depends on SMP && CONTAINERS
> Index: 2.6-mm/kernel/Makefile
> ===
> --- 2.6-mm.orig/kernel/Makefile
> +++ 2.6-mm/kernel/Makefile
> @@ -43,6 +43,7 @@
> obj-$(CONFIG_CPUSETS) += cpuset.o
> obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
> obj-$(CONFIG_CONTAINER_NS) += ns_container.o
> +obj-$(CONFIG_CONTAINER_NETWORK) += container_network.o
> obj-$(CONFIG_IKCONFIG) += configs.o
> obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
> obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
> Index: 2.6-mm/kernel/container_network.c
> ===
> --- /dev/null
> +++ 2.6-mm/kernel/container_network.c
> @@ -0,0 +1,285 @@
> +/*
> + * container_network.c - container network subsystem
> + *
> + * Copyright 2006, 2007 IBM Corp
> + */
> +
> +#include <linux/module.h>
> +#include <linux/container.h>
> +#include <linux/fs.h>
> +#include <linux/uaccess.h>
> +#include <linux/ctype.h>
> +#include <linux/list.h>
> +#include <linux/spinlock.h>
> +

> +struct network {
> + struct container_subsys_state css;
> + struct list_head ipv4_list; /* store the IPV4 addresses */
> + rwlock_t ipv4_list_lock;

Hmm, i assume you'll need to take this read_lock during all of the lsm
hooks then? So rcu might be a far better choice so you can just grab
rcu_read_lock in the lsm hooks.

> +};
> +
> +struct ipv4_list {
> + __be32 address;
> + struct list_head list;
> +};
> +
> +static struct network top_network = {
> + .ipv4_list = LIST_HEAD_INIT(top_network.ipv4_list),
> + .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
> +};
> +
> +struct container_subsys network_subsys;
> +
> +enum container_filetype {
> + FILE_IPV4,
> +};
> +
> +static inline struct network *container_network(struct container *container)
> +{
> + return container_of(
> + container_subsys_state(container, network_subsys_id),
> + struct network, css);
> +}
> +
> +static struct container_subsys_state *network_create(struct container_subsys *ss,
> + struct container *container)
> +{
> + struct network *network;
> +
> + /* Don't let anybody do that */
> + if (!capable(CAP_NET_ADMIN))
> + return ERR_PTR(-EPERM);
> +
> + /* The current container is the initial container */
> + if (!container->parent)
> + return &top_network.css;
> +
> + network = kzalloc(sizeof(*network), GFP_KERNEL);

```
> + if (!network)
> +  return ERR_PTR(-ENOMEM);
> +
> + INIT_LIST_HEAD(&network->ipv4_list);
> + network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
> +
> + return &network->css;
> +}
> +
> +static void network_destroy(struct container_subsys *ss,
> +       struct container *container)
> +{
> + struct network *network;
> + struct ipv4_list *entry, *next;
> + struct list_head *l;
> + rwlock_t *lock;
> +
> + network = container_network(container);
> + l = &network->ipv4_list;
> + lock = &network->ipv4_list_lock;
> +
> + /* flush the ipv4 list */
> + write_lock(lock);
> + list_for_each_entry_safe(entry, next, l, list) {
> +  list_del(&entry->list);
> +  kfree(entry);
> + }
> + write_unlock(lock);
> +
> + kfree(network);
> +}
> +
> +static int network_add_ipv4_address(struct container *container, __be32 address)
> +{
> + struct ipv4_list *entry;
> + struct network *network;
> +
> + entry = kmalloc(sizeof(*entry), GFP_KERNEL);
> + if (!entry)
> +  return -ENOMEM;
> + entry->address = address;
> +
> + network = container_network(container);
> + write_lock(&network->ipv4_list_lock);
> + list_add(&entry->list, &network->ipv4_list);
> + write_unlock(&network->ipv4_list_lock);
> +
> + return 0;
```

```
> +}
> +
> +static int network_del_ipv4_address(struct container *container, __be32 address)
> +{
> + struct ipv4_list *entry;
> + struct network *network;
> + int ret = 0;
> +
> + network = container_network(container);
> + write_lock(&network->ipv4_list_lock);
> + list_for_each_entry(entry, &network->ipv4_list, list) {
> +  if (entry->address != address)
> +   continue;
> +
> +  list_del(&entry->list);
> +  goto out_free;
> + }
> + ret = -EINVAL;
> +out:
> + write_unlock(&network->ipv4_list_lock);
> + return ret;
> +
> +out_free:
> + kfree(entry);
> + goto out;
> +}
> +
> +static int network_parse_ipv4_address(struct container *container, char *buffer)
> +{
> + int len = strlen(buffer);
> + char *addr;
> + __be32 address;
> +
> + /* remove trailing left space */
> + while(isspace(*buffer))
> +  buffer++;
> +
> + /* remove trailing right space */
> + while(isspace(buffer[len - 1]))
> +  buffer[(len--) - 1] = 0;
> +
> + len = strlen(buffer);
> +        addr = memchr(buffer, ' ', len);
> + if (!addr)
> +  return -EINVAL;
> + *addr++ = 0;
> +
> + /* remove trailing left space again */
```

```
> + while(isspace(*addr))
> +  addr++;
> +
> + /* Shall I check if the address is setup on the host ? */
> + sscanf(addr, "%X", &address);
> +
> + if (!strcmp(buffer, "add"))
> +  return network_add_ipv4_address(container, address);
> + else if (!strcmp(buffer, "del"))
> +  return network_del_ipv4_address(container, address);
> +
> + return -EINVAL;
> +}
> +
> +static int network_fill_ipv4_address(struct container *container, char *buffer)
> +{
> + struct network *network;
> + struct ipv4_list *entry;
> + char *s = buffer;
> + network = container_network(container);
> +
> + read_lock(&network->ipv4_list_lock);
> + list_for_each_entry(entry, &network->ipv4_list, list)
> +  s += sprintf(s, "%X\n", entry->address);
> + read_unlock(&network->ipv4_list_lock);
> +
> +  return strlen(buffer);
> +}
> +
> +static ssize_t network_write(struct container *container,
> +        struct cftype *cft,
> +        struct file *file,
> +        const char __user *userbuf,
> +        size_t nbytes, loff_t *unused_ppos)
> +{
> + enum container_filetype type = cft->private;
> + char *buffer;
> + int retval = 0;
> +
> + if (!capable(CAP_NET_ADMIN))
> +  return -EPERM;
> +
> + if (nbytes >= PATH_MAX)
> +  return -E2BIG;
> +
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (!buffer)
> +  return -ENOMEM;
```

```
> +
> + if (copy_from_user(buffer, userbuf, nbytes)) {
> +  retval = -EFAULT;
> +  goto out_free;
> + }
> + buffer[nbytes] = 0;
> +
> + container_lock();
> + switch(type) {
> +
> + case FILE_IPV4:
> +  retval = network_parse_ipv4_address(container, buffer);
> +  break;
> +
> + default:
> +  retval = -EINVAL;
> +  break;
> + };
> + container_unlock();
> +
> +out_free:
> + if (!retval)
> +  retval = nbytes;
> +
> + kfree(buffer);
> + return retval;
> +}
> +
> +static ssize_t network_read(struct container *container,
> +       struct cftype *cft,
> +       struct file *file,
> +       char __user *userbuf,
> +       size_t nbytes, loff_t *ppos)
> +{
> + enum container_filetype type = cft->private;
> + char *page;
> + int retval;
> +
> + page = (char *)__get_free_page(GFP_TEMPORARY);
> + if (!page)
> +  return -ENOMEM;
> +
> + container_lock();
> + switch(type) {
> + case FILE_IPV4:
> +  retval = network_fill_ipv4_address(container, page);
> +  break;
> +
```

> + default:
> + retval = -EINVAL;
> + };
> + container_unlock();
> +
> + retval = simple_read_from_buffer(userbuf, nbytes, ppos, page, retval);
> +
> + free_page((unsigned long)page);
> + return retval;
> +}
> +
> +static struct cftype files[] = {
> + {
> + .name = "ipv4",
> + .read = network_read,
> + .write = network_write,
> + .private = FILE_IPV4,
> + },
> +};
> +
> +static int network_populate(struct container_subsys *ss, struct container *cont)
> +{
> + return container_add_files(cont, ss, files, ARRAY_SIZE(files));
> +}
> +
> +struct container_subsys network_subsys = {
> + .name = "network",
> + .create = network_create,
> + .destroy = network_destroy,
> + .populate = network_populate,
> + .subsys_id = network_subsys_id,
> + .can_attach = NULL,

Heh, well you'll definately want to define can_attach() to prevent a
task simply entering another container, right?

> + .attach = NULL,
> + .fork = NULL,
> + .exit = NULL,
> +};
>
> --
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

_____
Containers mailing list

Subject: Re: [RFC][patch 2/3] network security hooks
Posted by serue on Wed, 05 Sep 2007 16:04:08 GMT
View Forum Message <> Reply to Message

Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> After all, the network security hooks are placed exactly at the places we need.
> This patch plugs the network security hooks with network container subsystem.
>
> The hooks always do nothing when they are called from a process running inside
> the root container.
>
> The security hooks are not activated by default when the root container is created,
> I let the first child container to do that, that ensure correct kernel initialization.
>
> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>
>
> ---
> kernel/container_network.c | 148 ++
> 1 file changed, 148 insertions(+)
>
> Index: 2.6-mm/kernel/container_network.c
> ===
> --- 2.6-mm.orig/kernel/container_network.c
> +++ 2.6-mm/kernel/container_network.c
> @@ -11,6 +11,7 @@
> #include <linux/ctype.h>
> #include <linux/list.h>
> #include <linux/spinlock.h>
> +#include <linux/security.h>
>
> struct network {
>   struct container_subsys_state css;
> @@ -28,12 +29,21 @@
>   .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
> };
>
> +static int security_registered;
> +static int secondary;
> +
> struct container_subsys network_subsys;
>

> enum container_filetype {
>   FILE_IPV4,
> };
>
> +static inline struct network *task_network(struct task_struct *task)
> +{
> + return container_of(task_subsys_state(task, network_subsys_id),
> +   struct network, css);
> +}
> +
> static inline struct network *container_network(struct container *container)
> {
>   return container_of(
> @@ -41,6 +51,125 @@
>   struct network, css);
> }
>
> +static int network_socket_create(int family, int type, int protocol, int kern)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)

I'm not sure that's the way to go here is it? Maybe you want to allow a
CAP_NET_ADMIN process in top_network to drop a network address so that
if it does a bind(INADDR_ANY) it won't get connections for vserver1?

> + return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_post_create(struct socket *sock, int family,
> +   int type, int protocol, int kern)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> + return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_bind(struct socket *sock,
> +   struct sockaddr *address,
> +   int addrlen)

```
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_connect(struct socket * sock,
> +      struct sockaddr * address,
> +      int addrlen)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_listen(struct socket * sock, int backlog)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_accept(struct socket *sock,
> +     struct socket *newsock)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static void network_socket_post_accept(struct socket *sock,
> +           struct socket *newsock)
```

```
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return;
> +}
> +
> +static int network_socket_sendmsg(struct socket *sock,
> +      struct msghdr *msg, int size)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static int network_socket_recvmsg(struct socket *sock,
> +      struct msghdr *msg, int size,
> +      int flags)
> +{
> + struct network *network;
> +
> + network = task_network(current);
> + if (!network || network == &top_network)
> +  return 0;
> +
> + return 0;
> +}
> +
> +static struct security_operations network_security_ops = {
> + .socket_create = network_socket_create,
> + .socket_post_create = network_socket_post_create,
> + .socket_bind = network_socket_bind,
> + .socket_connect = network_socket_connect,
> + .socket_listen = network_socket_listen,
> + .socket_accept = network_socket_accept,
> + .socket_post_accept = network_socket_post_accept,
> + .socket_sendmsg = network_socket_sendmsg,
> + .socket_recvmsg = network_socket_recvmsg,
> +};
> +
>  static struct container_subsys_state *network_create(struct container_subsys *ss,
>             struct container *container)
>  {
```

```
> @@ -61,6 +190,25 @@
>   INIT_LIST_HEAD(&network->ipv4_list);
>   network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
>
> + /*
> +  * register the network security hooks only one time
> +  * after the root container is created, the first non
> +  * root container has the assignment to register the
> +  * security hooks
> +  *
> +  */
> + if (!security_registered) {
> +  if (register_security(&network_security_ops)) {
> +   if (mod_reg_security(KBUILD_MODNAME,
> +         &network_security_ops)) {
> +    kfree(network);
> +    return ERR_PTR(-EINVAL);
> +   }
> +   secondary = 1;
> +  }
> +  security_registered = 1;
> + }
> +
>   return &network->css;
>  }
```

This is an unusual way to do this... Usually one registers the lsm hooks at module load. Though I can't think of any reason it's particularly bad, other than the register_security() might have succeeded when you are first loaded and might fail later.

Congratulations, I think you're the first to do this :)

Note, you don't ever unregister yourself, so there is no point in defining 'secondary'.

-serge

_____

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

---

Subject: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by serue on Wed, 05 Sep 2007 16:18:48 GMT
View Forum Message <> Reply to Message

Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):

> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> For the moment, I only made this patch for the RFC. It shows how simple it is
> to hook different socket syscalls. This patch denies bind to any addresses
> which are not in the container IPV4 address list, except for the INADDR_ANY.
>
> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>
> ---
> kernel/container_network.c | 66 ++++++++++++++++++++++----------------------
> 1 file changed, 35 insertions(+), 31 deletions(-)
>
> Index: 2.6-mm/kernel/container_network.c
> ===
> --- 2.6-mm.orig/kernel/container_network.c
> +++ 2.6-mm/kernel/container_network.c
> @@ -12,6 +12,9 @@
> #include <linux/list.h>
> #include <linux/spinlock.h>
> #include <linux/security.h>
> +#include <linux/in.h>
> +#include <linux/net.h>
> +#include <linux/socket.h>
>
> struct network {
>   struct container_subsys_state css;
> @@ -53,24 +56,14 @@
>
> static int network_socket_create(int family, int type, int protocol, int kern)
> {
> - struct network *network;
> -
> - network = task_network(current);
> - if (!network || network == &top_network)
> - return 0;
> -
> + /* nothing to do right now */
>   return 0;
> }
>
> static int network_socket_post_create(struct socket *sock, int family,
>   int type, int protocol, int kern)
> {
> - struct network *network;
> -
> - network = task_network(current);
> - if (!network || network == &top_network)
> - return 0;

> -
> + /* nothing to do right now */
> return 0;
> }
>
> @@ -79,47 +72,58 @@

Please so send -p diffs. I'll assume this is network_socket_bind()
given your patch description :)

> int addrlen)
> {
> struct network *network;
> + struct list_head *l;
> + rwlock_t *lock;
> + struct ipv4_list *entry;
> + __be32 addr;
> + int ret = -EPERM;
>
> + /* Do nothing for the root container */
> network = task_network(current);
> if (!network || network == &top_network)
> return 0;
>
> - return 0;
> + /* Check we have to do some filtering */
> + if (sock->ops->family != AF_INET)
> + return 0;
> +
> + l = &network->ipv4_list;
> + lock = &network->ipv4_list_lock;
> + addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
> +
> + if (addr == INADDR_ANY)

In bsdjail, if addr == INADDR_ANY, I set addr = jailaddr. Do you think
you want to do that?

> + return 0;
> +
> + read_lock(lock);
> + list_for_each_entry(entry, l, list) {
> + if (entry->address != addr)
> + continue;
> + ret = 0;
> + break;
> + }
> + read_unlock(lock);

> +
> + return ret;
> }
>
> static int network_socket_connect(struct socket * sock,
>   struct sockaddr * address,
>   int addrlen)
> {
> - struct network *network;
> -
> - network = task_network(current);
> - if (!network || network == &top_network)
> - return 0;
> -
> + /* nothing to do right now */
>   return 0;
> }
>
> static int network_socket_listen(struct socket * sock, int backlog)
> {
> - struct network *network;
> -
> - network = task_network(current);
> - if (!network || network == &top_network)
> - return 0;
> -
> + /* nothing to do right now */
>   return 0;
> }
>
> static int network_socket_accept(struct socket *sock,
>   struct socket *newsock)
> {
> - struct network *network;
> -
> - network = task_network(current);
> - if (!network || network == &top_network)
> - return 0;
> -
> + /* nothing to do right now */
>   return 0;
> }
>
>
> --
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org

> https://lists.linux-foundation.org/mailman/listinfo/containers

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

---

Subject: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by Daniel Lezcano on Wed, 05 Sep 2007 16:38:57 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
>> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>>
>> For the moment, I only made this patch for the RFC. It shows how simple it is
>> to hook different socket syscalls. This patch denies bind to any addresses
>> which are not in the container IPV4 address list, except for the INADDR_ANY.
>>
>> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>>
>> ---
>> kernel/container_network.c | 66 ++++++++++++++++++++++----------------------
>> 1 file changed, 35 insertions(+), 31 deletions(-)
>>
>> Index: 2.6-mm/kernel/container_network.c
>> ===
>> --- 2.6-mm.orig/kernel/container_network.c
>> +++ 2.6-mm/kernel/container_network.c
>> @@ -12,6 +12,9 @@
>> #include <linux/list.h>
>> #include <linux/spinlock.h>
>> #include <linux/security.h>
>> +#include <linux/in.h>
>> +#include <linux/net.h>
>> +#include <linux/socket.h>
>>
>> struct network {
>>   struct container_subsys_state css;
>> @@ -53,24 +56,14 @@
>>
>> static int network_socket_create(int family, int type, int protocol, int kern)
>> {
>> - struct network *network;
>> -
>> - network = task_network(current);
>> - if (!network || network == &top_network)
>> - return 0;

```
>> -
>> + /* nothing to do right now */
>>   return 0;
>>  }
>>
>>  static int network_socket_post_create(struct socket *sock, int family,
>>            int type, int protocol, int kern)
>>  {
>> - struct network *network;
>> -
>> - network = task_network(current);
>> - if (!network || network == &top_network)
>> -  return 0;
>> -
>> + /* nothing to do right now */
>>   return 0;
>>  }
>>
>> @@ -79,47 +72,58 @@
>
> Please so send -p diffs.  I'll assume this is network_socket_bind()
> given your patch description :)
>
>>            int addrlen)
>>  {
>>   struct network *network;
>> + struct list_head *l;
>> + rwlock_t *lock;
>> + struct ipv4_list *entry;
>> + __be32 addr;
>> + int ret = -EPERM;
>>
>> + /* Do nothing for the root container */
>>   network = task_network(current);
>>   if (!network || network == &top_network)
>>    return 0;
>>
>> - return 0;
>> + /* Check we have to do some filtering */
>> + if (sock->ops->family != AF_INET)
>> +  return 0;
>> +
>> + l = &network->ipv4_list;
>> + lock = &network->ipv4_list_lock;
>> + addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
>> +
>> + if (addr == INADDR_ANY)
>
```

> In bsdjail, if addr == INADDR_ANY, I set addr = jailaddr. Do you think
> you want to do that?

Good question. This is one think I would like to define. If we do that
we can not connect via 127.0.0.1. and|or a container can have more than
one IP address, no ?
IMHO, we should have the loopback address available for all containers
and that means 127.0.0.1 is an IP address which is not isolated.

If we choose to deny access to 127.0.0.1, then there will be some issues
with the routing. If we connect to 127.0.0.1 (this address belongs to
the root container) from a child container, the source address should be
filled with an IP address belonging to a container (eg 10.0.0.10), so we
have (src)10.0.0.1 -> (dst)127.0.0.1, that means the root container will
answer to 10.0.0.1 and use this address. This is no sense because
routing should be for the loopback: 127.0.0.1<->127.0.0.1, and we break
isolation. Tricky.

>
>> + return 0;
>> +
>> + read_lock(lock);
>> + list_for_each_entry(entry, l, list) {
>> + if (entry->address != addr)
>> +   continue;
>> + ret = 0;
>> + break;
>> + }
>> + read_unlock(lock);
>> +
>> + return ret;
>> }
>>
>> static int network_socket_connect(struct socket * sock,
>>   struct sockaddr * address,
>>   int addrlen)
>> {
>> - struct network *network;
>> -
>> - network = task_network(current);
>> - if (!network || network == &top_network)
>> - return 0;
>> -
>> + /* nothing to do right now */
>>   return 0;
>> }
>>
>> static int network_socket_listen(struct socket * sock, int backlog)

>> {
>> - struct network *network;
>> -
>> - network = task_network(current);
>> - if (!network || network == &top_network)
>> - return 0;
>> -
>> + /* nothing to do right now */
>> return 0;
>> }
>>
>> static int network_socket_accept(struct socket *sock,
>> struct socket *newsock)
>> {
>> - struct network *network;
>> -
>> - network = task_network(current);
>> - if (!network || network == &top_network)
>> - return 0;
>> -
>> + /* nothing to do right now */
>> return 0;
>> }
>>
>>
>> --
>> _____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by serue on Mon, 10 Sep 2007 13:23:30 GMT
View Forum Message <> Reply to Message

Quoting Daniel Lezcano (dlezcano@meiosys.com):
> Serge E. Hallyn wrote:
>> Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
>>> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>
>>> For the moment, I only made this patch for the RFC. It shows how simple

```
>>> it is
>>> to hook different socket syscalls. This patch denies bind to any
>>> addresses
>>> which are not in the container IPV4 address list, except for the
>>> INADDR_ANY.
>>>
>>> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>
>>> ---
>>>  kernel/container_network.c |   66
>>> ++++++++++++++++++++++++----------------------
>>>  1 file changed, 35 insertions(+), 31 deletions(-)
>>>
>>> Index: 2.6-mm/kernel/container_network.c
>>> ===================================================================
>>> --- 2.6-mm.orig/kernel/container_network.c
>>> +++ 2.6-mm/kernel/container_network.c
>>> @@ -12,6 +12,9 @@
>>>  #include <linux/list.h>
>>>  #include <linux/spinlock.h>
>>>  #include <linux/security.h>
>>> +#include <linux/in.h>
>>> +#include <linux/net.h>
>>> +#include <linux/socket.h>
>>>
>>>  struct network {
>>>   struct container_subsys_state css;
>>> @@ -53,24 +56,14 @@
>>>
>>>  static int network_socket_create(int family, int type, int protocol, int
>>> kern)
>>>  {
>>> - struct network *network;
>>> -
>>> - network = task_network(current);
>>> - if (!network || network == &top_network)
>>> -  return 0;
>>> -
>>> + /* nothing to do right now */
>>>   return 0;
>>>  }
>>>
>>>  static int network_socket_post_create(struct socket *sock, int family,
>>>           int type, int protocol, int kern)
>>>  {
>>> - struct network *network;
>>> -
>>> - network = task_network(current);
```

>>> - if (!network || network == &top_network)
>>> - return 0;
>>> -
>>> + /* nothing to do right now */
>>>   return 0;
>>> }
>>>
>>> @@ -79,47 +72,58 @@
>> Please so send -p diffs. I'll assume this is network_socket_bind()
>> given your patch description :)
>>>   int addrlen)
>>> {
>>>   struct network *network;
>>> + struct list_head *l;
>>> + rwlock_t *lock;
>>> + struct ipv4_list *entry;
>>> + __be32 addr;
>>> + int ret = -EPERM;
>>>
>>> + /* Do nothing for the root container */
>>>   network = task_network(current);
>>>   if (!network || network == &top_network)
>>>   return 0;
>>>
>>> - return 0;
>>> + /* Check we have to do some filtering */
>>> + if (sock->ops->family != AF_INET)
>>> + return 0;
>>> +
>>> + l = &network->ipv4_list;
>>> + lock = &network->ipv4_list_lock;
>>> + addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
>>> +
>>> + if (addr == INADDR_ANY)
>> In bsdjail, if addr == INADDR_ANY, I set addr = jailaddr. Do you think
>> you want to do that?
>
> Good question. This is one think I would like to define. If we do that we
> can not connect via 127.0.0.1. and|or a container can have more than one IP
> address, no ?

Yes.

> IMHO, we should have the loopback address available for all containers and
> that means 127.0.0.1 is an IP address which is not isolated.

For real network namespaces yes. For this version, I would have thought
the goal would be to provide a minimal, useful, but very fast

container-paddress binding.

I guess I'll have to see the rest of your implementation, but I have the
feeling that to not have this limitation you'll affect performance a
bit. And since we are also working on full network namespaces,
providing maximal functionality with worse performance would be a poor
tradeoff here.

But let's see the rest of your implementation.

Did you mention somewhere that Eric still prefers using netfilter rather
than LSM? So what... if a packet comes in with a certain destination
address you can tag it with a container, and once a connection starts
you can use connection tracking to continue tagging it with that
container. You tag an outgoing packet with the container as soon as
it's dumped in the socket, and rules enforce that the source address be
valid for that container. Are you saying the netfilter hooks are in
the wrong places for that?

-serge

> If we choose to deny access to 127.0.0.1, then there will be some issues
> with the routing. If we connect to 127.0.0.1 (this address belongs to the
> root container) from a child container, the source address should be filled
> with an IP address belonging to a container (eg 10.0.0.10), so we have
> (src)10.0.0.1 -> (dst)127.0.0.1, that means the root container will answer
> to 10.0.0.1 and use this address. This is no sense because routing should
> be for the loopback: 127.0.0.1<->127.0.0.1, and we break isolation. Tricky.
>
>>> + return 0;
>>> +
>>> + read_lock(lock);
>>> + list_for_each_entry(entry, l, list) {
>>> + if (entry->address != addr)
>>> +   continue;
>>> + ret = 0;
>>> + break;
>>> + }
>>> + read_unlock(lock);
>>> +
>>> + return ret;
>>> }
>>>
>>> static int network_socket_connect(struct socket * sock,
>>>   struct sockaddr * address,
>>>   int addrlen)
>>> {
>>> - struct network *network;

>>> -
>>> - network = task_network(current);
>>> - if (!network || network == &top_network)
>>> - return 0;
>>> -
>>> + /* nothing to do right now */
>>>   return 0;
>>> }
>>>
>>> static int network_socket_listen(struct socket * sock, int backlog)
>>> {
>>> - struct network *network;
>>> -
>>> - network = task_network(current);
>>> - if (!network || network == &top_network)
>>> - return 0;
>>> -
>>> + /* nothing to do right now */
>>>   return 0;
>>> }
>>>
>>> static int network_socket_accept(struct socket *sock,
>>>   struct socket *newsock)
>>> {
>>> - struct network *network;
>>> -
>>> - network = task_network(current);
>>> - if (!network || network == &top_network)
>>> - return 0;
>>> -
>>> + /* nothing to do right now */
>>>   return 0;
>>> }
>>>
>>>
>>> --
>>> _____
>>> Containers mailing list
>>> Containers@lists.linux-foundation.org
>>> https://lists.linux-foundation.org/mailman/listinfo/containers

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

---

Subject: Re: [RFC][patch 3/3] activate filtering for the bind

Serge E. Hallyn wrote:
> Quoting Daniel Lezcano (dlezcano@meiosys.com):
>> Serge E. Hallyn wrote:
>>> Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
>>>> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>>
>>>> For the moment, I only made this patch for the RFC. It shows how simple
>>>> it is
>>>> to hook different socket syscalls. This patch denies bind to any
>>>> addresses
>>>> which are not in the container IPV4 address list, except for the
>>>> INADDR_ANY.
>>>>
>>>> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>>
>>>> ---
>>>> kernel/container_network.c | 66
>>>> ++++++++++++++++++++++----------------
>>>> 1 file changed, 35 insertions(+), 31 deletions(-)
>>>>
>>>> Index: 2.6-mm/kernel/container_network.c
>>>> ===
>>>> --- 2.6-mm.orig/kernel/container_network.c
>>>> +++ 2.6-mm/kernel/container_network.c
>>>> @@ -12,6 +12,9 @@
>>>> #include <linux/list.h>
>>>> #include <linux/spinlock.h>
>>>> #include <linux/security.h>
>>>> +#include <linux/in.h>
>>>> +#include <linux/net.h>
>>>> +#include <linux/socket.h>
>>>>
>>>> struct network {
>>>>   struct container_subsys_state css;
>>>> @@ -53,24 +56,14 @@
>>>>
>>>> static int network_socket_create(int family, int type, int protocol, int
>>>> kern)
>>>> {
>>>> - struct network *network;
>>>> -
>>>> - network = task_network(current);
>>>> - if (!network || network == &top_network)
>>>> - return 0;
>>>> -
>>>> + /* nothing to do right now */

>>>>   return 0;
>>>> }
>>>>
>>>> static int network_socket_post_create(struct socket *sock, int family,
>>>>   int type, int protocol, int kern)
>>>> {
>>>> - struct network *network;
>>>> -
>>>> - network = task_network(current);
>>>> - if (!network || network == &top_network)
>>>> - return 0;
>>>> -
>>>> + /* nothing to do right now */
>>>>   return 0;
>>>> }
>>>>
>>>> @@ -79,47 +72,58 @@
>>> Please so send -p diffs. I'll assume this is network_socket_bind()
>>> given your patch description :)
>>>>   int addrlen)
>>>> {
>>>>   struct network *network;
>>>> + struct list_head *l;
>>>> + rwlock_t *lock;
>>>> + struct ipv4_list *entry;
>>>> + __be32 addr;
>>>> + int ret = -EPERM;
>>>>
>>>> + /* Do nothing for the root container */
>>>>   network = task_network(current);
>>>>   if (!network || network == &top_network)
>>>>   return 0;
>>>>
>>>> - return 0;
>>>> + /* Check we have to do some filtering */
>>>> + if (sock->ops->family != AF_INET)
>>>> + return 0;
>>>> +
>>>> + l = &network->ipv4_list;
>>>> + lock = &network->ipv4_list_lock;
>>>> + addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
>>>> +
>>>> + if (addr == INADDR_ANY)
>>> In bsdjail, if addr == INADDR_ANY, I set addr = jailaddr. Do you think
>>> you want to do that?
>> Good question. This is one think I would like to define. If we do that we
>> can not connect via 127.0.0.1. and|or a container can have more than one IP
>> address, no ?

>
> Yes.
>
>> IMHO, we should have the loopback address available for all containers and
>> that means 127.0.0.1 is an IP address which is not isolated.
>
> For real network namespaces yes. For this version, I would have thought
> the goal would be to provide a minimal, useful, but very fast
> container-paddress binding.
>
> I guess I'll have to see the rest of your implementation, but I have the
> feeling that to not have this limitation you'll affect performance a
> bit. And since we are also working on full network namespaces,
> providing maximal functionality with worse performance would be a poor
> tradeoff here.
>
> But let's see the rest of your implementation.
>
> Did you mention somewhere that Eric still prefers using netfilter rather
> than LSM?

Paul told me about a ip isolation based on the netfilter and a specific
iptable module:

"
On 9/6/07, Daniel Lezcano <dlezcano@fr.ibm.com> wrote:
 > >
 > > I am really not opposed to iptables, I was thinking that if we want to
 > > have bind filtering, security provides the framework for that and
adding
 > > new hooks for the iptable will just add a hook duplication because they
 > > are the same.
 > >
 > > So the result is:
 > >
 > > 1 - create a container => network.ipv4 (allowed addresses)
 > > 2 - echo add 192.168.20.10 > network.ipv4
 > >
 > > The application running inside the container can not use another
address
 > > than the one assigned to it.
 > >
 > > This features is needed for some IP jailing like linux-vserver or for
 > > security. The association container + IP isolation is really a good
feature.
 > >
 >> > > For instance, I personally am much more interested in being able to

>> > > control ports rather than IP addresses (although that could be
>> > > interesting too).
> >
> > What do want to do ? Can you describe the features you want ?
> > Is it a bind filtering for port ? If this is the case, then I can add
> > two new files:
> > network.tcp.ports
> > network.udp.ports
> > and extend the hooks to check the port too.
> >

I think that (at least today :-)) my ideal interface would be a list
of tuples of the form:

local port range/remote ip address/remote ip mask/remote port range

because I don't really care about multiple local addresses, but I do
care about binding to local ports and connecting to remote addresses
and ports.

But other people (e.g. Eric) have completely different requirements.
Creating an API and mechanism that satisfies everyone is going to
result in you reimplementing a significant chunk of the iptables
functionality.

> >
>> > > And someone else might have completely different
>> > > needs (e.g. people mentioned IPv6). Rather than you having to
>> > > implement all of these things, just giving a tag that can be tied to
>> > > iptables means that people can define these rules themselves in
>> > > userspace.
> >
> > I understand. But I don't see how we can handle bind filtering (ip
| port).
> >

1) Completely separately from containers, we create a new iptable
called something like "socket", with predefined chains BIND, ACCEPT
and CONNECT. When ever someone tries to do a bind(), accept() or
connect() we create a fake packet with the appropriate local and
remote addresses and ports, and feed it through the appropriate chain.
If it gets though OK we allow the operation to proceed, else we fail
with EPERM.

2) We create a new container (control group) subsystem, e.g. called
"network_id" that does two things:

- creates a simple state object with a single uniquely-generated

integer network_id for each control group

- provides a new iptable match module ("control_group"?) that matches
if the current task's network_id is within a given range.

Then the user can create pretty much arbitrary rules with the existing
iptables tools and primitives. No complex new user APIs needed.

Paul"

> So what... if a packet comes in with a certain destination
> address you can tag it with a container, and once a connection starts
> you can use connection tracking to continue tagging it with that
> container. You tag an outgoing packet with the container as soon as
> it's dumped in the socket, and rules enforce that the source address be
> valid for that container. Are you saying the netfilter hooks are in
> the wrong places for that?

No, for that netfilters hooks are in the right place.

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

---

Subject: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by serue on Mon, 10 Sep 2007 15:46:30 GMT
View Forum Message <> Reply to Message

Quoting Daniel Lezcano (dlezcano@fr.ibm.com):
> Serge E. Hallyn wrote:
>> Quoting Daniel Lezcano (dlezcano@meiosys.com):
>>> Serge E. Hallyn wrote:
>>>> Quoting dlezcano@fr.ibm.com (dlezcano@fr.ibm.com):
>>>>> From: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>>>
>>>>> For the moment, I only made this patch for the RFC. It shows how simple
>>>>> it is
>>>>> to hook different socket syscalls. This patch denies bind to any
>>>>> addresses
>>>>> which are not in the container IPV4 address list, except for the
>>>>> INADDR_ANY.
>>>>>
>>>>> Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
>>>>>

```
>>>>> ---
>>>>>  kernel/container_network.c |   66
>>>>> ++++++++++++++++++++++++----------------------
>>>>>  1 file changed, 35 insertions(+), 31 deletions(-)
>>>>>
>>>>> Index: 2.6-mm/kernel/container_network.c
>>>>> ===================================================================
>>>>> --- 2.6-mm.orig/kernel/container_network.c
>>>>> +++ 2.6-mm/kernel/container_network.c
>>>>> @@ -12,6 +12,9 @@
>>>>>  #include <linux/list.h>
>>>>>  #include <linux/spinlock.h>
>>>>>  #include <linux/security.h>
>>>>> +#include <linux/in.h>
>>>>> +#include <linux/net.h>
>>>>> +#include <linux/socket.h>
>>>>>
>>>>>  struct network {
>>>>>   struct container_subsys_state css;
>>>>> @@ -53,24 +56,14 @@
>>>>>
>>>>>  static int network_socket_create(int family, int type, int protocol,
>>>>> int kern)
>>>>>  {
>>>>> - struct network *network;
>>>>> -
>>>>> - network = task_network(current);
>>>>> - if (!network || network == &top_network)
>>>>> -  return 0;
>>>>> -
>>>>> + /* nothing to do right now */
>>>>>   return 0;
>>>>>  }
>>>>>
>>>>>  static int network_socket_post_create(struct socket *sock, int family,
>>>>>            int type, int protocol, int kern)
>>>>>  {
>>>>> - struct network *network;
>>>>> -
>>>>> - network = task_network(current);
>>>>> - if (!network || network == &top_network)
>>>>> -  return 0;
>>>>> -
>>>>> + /* nothing to do right now */
>>>>>   return 0;
>>>>>  }
>>>>>
>>>>> @@ -79,47 +72,58 @@
```

>>>> Please so send -p diffs. I'll assume this is network_socket_bind()
>>>> given your patch description :)
>>>>>   int addrlen)
>>>>> {
>>>>>   struct network *network;
>>>>> + struct list_head *l;
>>>>> + rwlock_t *lock;
>>>>> + struct ipv4_list *entry;
>>>>> + __be32 addr;
>>>>> + int ret = -EPERM;
>>>>>
>>>>> + /* Do nothing for the root container */
>>>>>   network = task_network(current);
>>>>>   if (!network || network == &top_network)
>>>>>   return 0;
>>>>>
>>>>> - return 0;
>>>>> + /* Check we have to do some filtering */
>>>>> + if (sock->ops->family != AF_INET)
>>>>> + return 0;
>>>>> +
>>>>> + l = &network->ipv4_list;
>>>>> + lock = &network->ipv4_list_lock;
>>>>> + addr = ((struct sockaddr_in *)address)->sin_addr.s_addr;
>>>>> +
>>>>> + if (addr == INADDR_ANY)
>>>> In bsdjail, if addr == INADDR_ANY, I set addr = jailaddr. Do you think
>>>> you want to do that?
>>> Good question. This is one think I would like to define. If we do that we
>>> can not connect via 127.0.0.1. and|or a container can have more than one
>>> IP address, no ?
>> Yes.
>>> IMHO, we should have the loopback address available for all containers
>>> and that means 127.0.0.1 is an IP address which is not isolated.
>> For real network namespaces yes. For this version, I would have thought
>> the goal would be to provide a minimal, useful, but very fast
>> container-paddress binding.
>> I guess I'll have to see the rest of your implementation, but I have the
>> feeling that to not have this limitation you'll affect performance a
>> bit. And since we are also working on full network namespaces,
>> providing maximal functionality with worse performance would be a poor
>> tradeoff here.
>> But let's see the rest of your implementation.
>> Did you mention somewhere that Eric still prefers using netfilter rather
>> than LSM?
>
> Paul told me about a ip isolation based on the netfilter and a specific
> iptable module:

>
> ---
> "
> On 9/6/07, Daniel Lezcano <dlezcano@fr.ibm.com> wrote:
> > >
> > > I am really not opposed to iptables, I was thinking that if we want to
> > > have bind filtering, security provides the framework for that and
> adding
> > > new hooks for the iptable will just add a hook duplication because they
> > > are the same.
> > >
> > > So the result is:
> > >
> > > 1 - create a container => network.ipv4 (allowed addresses)
> > > 2 - echo add 192.168.20.10 > network.ipv4
> > >
> > > The application running inside the container can not use another
> address
> > > than the one assigned to it.
> > >
> > > This features is needed for some IP jailing like linux-vserver or for
> > > security. The association container + IP isolation is really a good
> feature.
> > >
> >> > > For instance, I personally am much more interested in being able to
> >> > > control ports rather than IP addresses (although that could be
> >> > > interesting too).
> > >
> > > What do want to do ? Can you describe the features you want ?
> > > Is it a bind filtering for port ? If this is the case, then I can add
> > > two new files:
> > >   network.tcp.ports
> > >   network.udp.ports
> > > and extend the hooks to check the port too.
> > >
>
> I think that (at least today :-)) my ideal interface would be a list
> of tuples of the form:
>
> local port range/remote ip address/remote ip mask/remote port range
>
> because I don't really care about multiple local addresses, but I do
> care about binding to local ports and connecting to remote addresses
> and ports.
>
> But other people (e.g. Eric) have completely different requirements.
> Creating an API and mechanism that satisfies everyone is going to
> result in you reimplementing a significant chunk of the iptables

> functionality.
>
> > >
> >> > > And someone else might have completely different
> >> > > needs (e.g. people mentioned IPv6). Rather than you having to
> >> > > implement all of these things, just giving a tag that can be tied to
> >> > > iptables means that people can define these rules themselves in
> >> > > userspace.
> > >
> > > I understand. But I don't see how we can handle bind filtering (ip |
> port).
> > >
>
> 1) Completely separately from containers, we create a new iptable
> called something like "socket", with predefined chains BIND, ACCEPT
> and CONNECT. When ever someone tries to do a bind(), accept() or
> connect() we create a fake packet with the appropriate local and
> remote addresses and ports, and feed it through the appropriate chain.
> If it gets though OK we allow the operation to proceed, else we fail
> with EPERM.
>
> 2) We create a new container (control group) subsystem, e.g. called
> "network_id" that does two things:
>
> - creates a simple state object with a single uniquely-generated
> integer network_id for each control group
>
> - provides a new iptable match module ("control_group"?) that matches
> if the current task's network_id is within a given range.
>
> Then the user can create pretty much arbitrary rules with the existing
> iptables tools and primitives. No complex new user APIs needed.
>
> Paul"
> ---
>
>> So what... if a packet comes in with a certain destination
>> address you can tag it with a container, and once a connection starts
>> you can use connection tracking to continue tagging it with that
>> container. You tag an outgoing packet with the container as soon as
>> it's dumped in the socket, and rules enforce that the source address be
>> valid for that container. Are you saying the netfilter hooks are in
>> the wrong places for that?
>
> No, for that netfilters hooks are in the right place.

Ok, then that approach definately has its upsides.

The only downside I see right now is what to do about a sendto() on a
udp socket that hasn't been bound.

-serge

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

---

Subject: Re: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by Paul Menage on Mon, 10 Sep 2007 17:49:55 GMT
View Forum Message <> Reply to Message

On 9/10/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> The only downside I see right now is what to do about a sendto() on a
> udp socket that hasn't been bound.

Maybe have additional chains in the new iptable called "sendto" and
"recvfrom" that are invoked for those operations on unbound datagram
sockets?

Paul

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

---

Subject: Re: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by serue on Mon, 10 Sep 2007 18:11:31 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On 9/10/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> >
> > The only downside I see right now is what to do about a sendto() on a
> > udp socket that hasn't been bound.
>
> Maybe have additional chains in the new iptable called "sendto" and
> "recvfrom" that are invoked for those operations on unbound datagram
> sockets?

Yup.

Perhaps the biggest upside of this approach is that it's providing

network functionality in a way that should be much more familiar to
network folks. As opposed to using an lsm with a new vfs interface.

Is anyone working on this implementation, for comparison to the lsm
patch?

-serge

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

---

Subject: Re: Re: [RFC][patch 3/3] activate filtering for the bind
Posted by Paul Menage on Mon, 10 Sep 2007 18:15:19 GMT
View Forum Message <> Reply to Message

On 9/10/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> Perhaps the biggest upside of this approach is that it's providing
> network functionality in a way that should be much more familiar to
> network folks. As opposed to using an lsm with a new vfs interface.

Right - one of the things that I promised at the kernel summit was
that we'd be very careful about introducing random new userspace APIs
as part of control files. Reusing existing APIs where practical is way
nicer.

>
> Is anyone working on this implementation, for comparison to the lsm
> patch?

Eric may be; if not then it's something I'd be interested in doing but
probably won't have time for, for a couple of weeks at least. So if
someone else wanted to play with it in the meantime that would be
great.

Paul

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

---