# Application Confinement with User namespaces

Stéphane Graber, Serge Hallyn

Canonical, Inc

*stgraber@ubuntu.com, serge.hallyn@ubuntu.com*

August 19, 2014

# Overview

1. State of Containers Prior to User namespaces
2. User namespaces
3. Application confinement demo
4. LSMs

# Containers prior to user namespaces

### Namespaces

- *id* → *resource* mapping
  - Prevent resource access by not providing a handle
  - i.e. pid 1 is not global init
  - `/etc/shadow` not accessible
- Tons of "leaks" exist

### Control groups

1. Resource limits and accounting
2. Limit device access
3. If root, re-mount cgroups and change/escape limits.

### Capabilities bounding set

1. Limit privs of root in container
2. Root still owns most host files
3. http://www.sevagas.com/IMG/pdf/exploiting_capabilities_the_dark_side.pdf
4. Prevents useful things like tmpfs mounts

## LSMs

1. Paper of the (huge) remaining holes
2. i.e. prevent `/proc/sys/*` writing, etc
3. "Safe from accidental damage by container root"
4. People always want unsafe exceptions
5. Lack of policy nesting limits use *in* containers

## Seccomp

1. Prevent use of some syscalls
2. Reduce exposed kernel surface
3. Hard to do generally

```
2
blacklist
[all]
kexec_load errno 1
open_by_handle_at errno 1
init_module errno 1
finit_module errno 1
delete_module errno 1
```

Figure: Stock Ubuntu LXC Seccomp filter

1. Nevertheless
    1. Root in container is still root on host
    2. Any leak = game over
    3. Answer: "Wait for user namespaces"

# User namespaces

### Goals

1. Uid separation
    1. c1.500 != c2.500
    2. Separate access controls (kill, open, etc)
    3. Separate accounting, limits
2. Container root privileged over container
    1. uids
    2. network
    3. etc
3. Container root has no privilege outside of container
    1. Root in container as safe as unpriv user on host
    2. Safe for use by untrusted users
4. Able to be nested

## Original user namespace design

1. Per-userns uid table
    1. Simple separate accounting
2. user $= \{uid, userns\}$
    1. Access checks complicated
    2. Performance impact
    3. No verification that conversion is complete
    4. No confidence
3. On-disk representation options
    1. use xattrs
    2. mount-time mapping definition
    3. `/etc/` file naming namespaces, consulted at mount
4. Many years, little progress

## Current user namespace design

1. By Eric Biederman
2. In-kernel uids become new type (kuid_t)

   ```
   typedef struct {
   uid_t val;
   } kuid_t;
   ```

   Compiler enforces type safety
3. Uids map 1-1 to kuids
   1. Translated at kernel-user boundary
   2. Default mapping 0-4294967295:0-4294967295
   3. Unmapped userids show up as -1, has ‘o’ perms
   4. Unpriv user can only map own host uid
4. Other namespaces owned by a user ns
   1. Root in ns has full privilege over what it owns

## Uid delegation

1. Root delegates *subuids* to users
   1. /etc/subuid and /etc/subgid: serge:100000:65536
   2. Set using usermod: usermod -v 100000-200000 -w 100000-200000 serge
2. Setuid-root programs write to /proc/self/{ug}id_map
3. Each user may be delegated a set of subuids and subgids

# LXC Integration

1. Container configuration file lists id mappings:

   ```
   lxc.id_map = u 0 100000 1000
   lxc.id_map = g 0 100000 1000
   lxc.id_map = u 1000 1000 1
   lxc.id_map = g 1000 1000 1
   lxc.id_map = u 1001 101001 64535
   lxc.id_map = g 1001 101001 64535
   ```

2. lxc-create untars rootfs in namespace
3. lxc-user-nic: hook veth up to container bridge
   1. Subject to `/etc/lxc/lxc-usernet`

      ```
      # USERNAME TYPE BRIDGE COUNT
      serge veth lxcbr0 10
      ```

# Take it away, Stéphane

# LSM Interaction

1. LSMs:
    1. Only reduce access
    2. MAC orthogolan to DAC
2. However, transitions *do* lead to "privileged" types
3. Examples:
    1. *passwd*:
    2. *signals*:
4. So DAC ends up segragating the MAC
5. Is this a problem, or by design?