The trust|me solution for mobile devices

A Secure Architecture for Operating System-Level Virtualization on Mobile Device, February 23, 2017

Dr. Michael Weiß

Fraunhofer
AISEC

A Secure Architecture for Operating System-Level Virtualization on Mobile Devices [HHV+15]

http://dx.doi.org/10.1007/978-3-319-38898-4_25

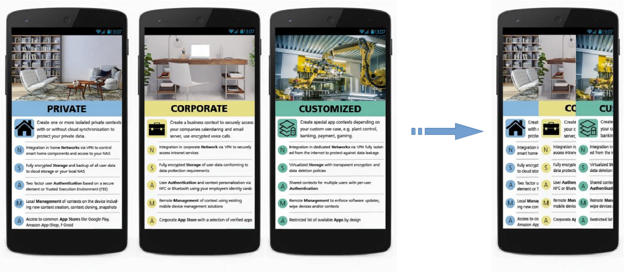
Fraunhofer AISEC

Outline

Fraunhofer AISEC

Motivation

- Sensitive private data exposed on mobile devices
- Devices incorporated for **different use-cases** (private, work, ...)
- Valuable assets on device have to be protected
  $\rightarrow$ OS-level virtualization enables **multiple**, **concurrent** and **separate** *OS instances* on one single physical device and kernel
- Improve the security of mobile devices through **separation** into *containers*



Fraunhofer AISEC

Goal and Contributions

Goal: Data confidentiality at container boundaries through container isolation

Contributions:

- A **secure virtualization architecture**
- **Container isolation** through
  - Confining of containers to only minimal, **controlled functionality**
  - Allowing only specific **communication channels**
- **Secure device virtualization** mechanisms
- A **secure container switching** procedure

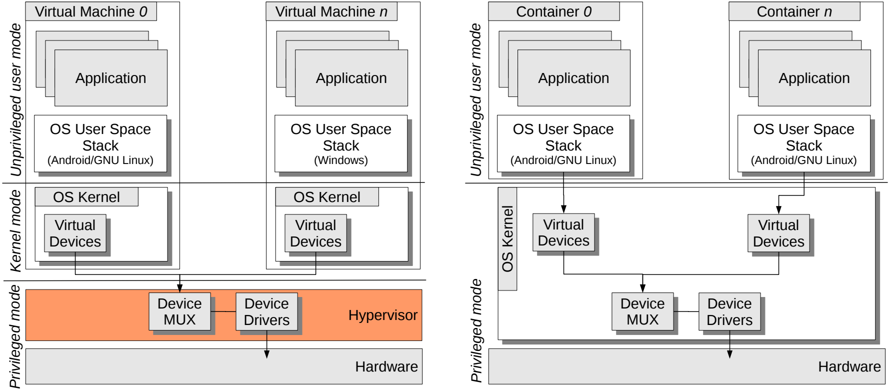
Background

OS-Level Virtualization

Background
OS-Level Virtualization

- OS-Level Virtualization also called Container Virtualization
- Examples:
  - Solaris Zones
  - FreeBSD Jails
  - Linux Containers (Docker, LXC, CoreOS ...)
  - OpenVZ
- trust|me Solution is also a Linux Container-based approach

Background
OS-Level Virtualization vs Full System Virtualization



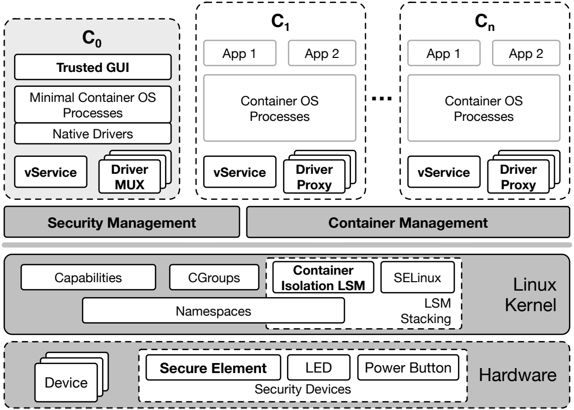
System Virtualization

OS-Level Virtualization

Background
OS-Level Virtualization vs Full System Virtualization (cont'd)

- Both provide full user space execution stacks
- OS-level virtualization uses one kernel for several user space stacks
- System virtualization runs an own kernel instance for every user space stack on top of a privileged kernel
- Trade-off: security/performance
- Root exploit vs kernel exploit

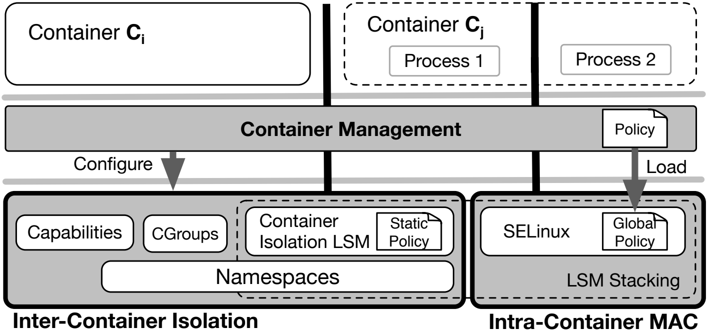
Secure Architecture
Architecture Overview



- *Security devices* and *kernel/user-space virtualized* devices
- Virtualization enablers: Linux kernel, Container Management (*cmld*), OS components
- Privileged container C_0 with Trusted GUI, unprivileged $C_{1..n}$ with virtualization extensions

Fraunhofer AISEC

Secure Architecture
Container Isolation Mechanisms



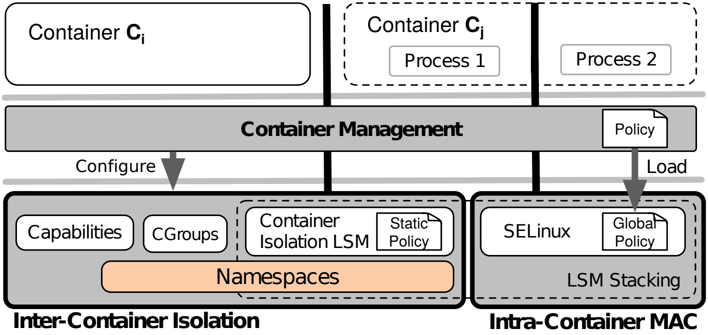
- Isolation of containers achieved with kernel isolation mechanisms
- Protection on inter- and intra-container basis
- Required Mechanisms: capabilities, cgroups, custom LSM, SELinux, namespaces, device namespaces [ADH$^+$11]

Secure Architecture
Linux Separation Mechanisms for Containers

- Namespaces
  - Separate access to kernel data structures
  - pid, uts, network, mount
  - user namespace (considered fully functional since Kernel version 3.8)
  - Since Kernel version 4.6 also CGroup namespaces (allowing nested/transparent CGroup configuration)
- CGroups
  - Restrict resource utilization
  - CPU, RAM, device access,
- Capabilities
  - Restrict root user’s capabilities

Secure Architecture

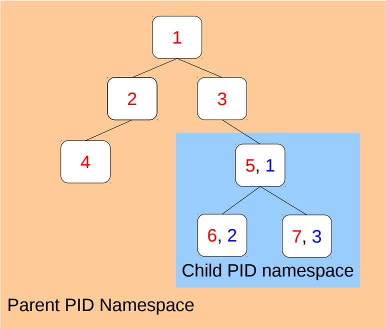
Container Isolation Mechanisms



Container Isolation Mechanisms

Linux Namespaces (PID)

- PID Namespace (CLONE_NEWPID)
- Isolate the process ID number
- Cloneing a child into a new pid results in ID 1 inside the namespace
- The parent pid (`ppid`) of that child is set to 0
- PID 1 of any pidns is considered to be the "*init*" process



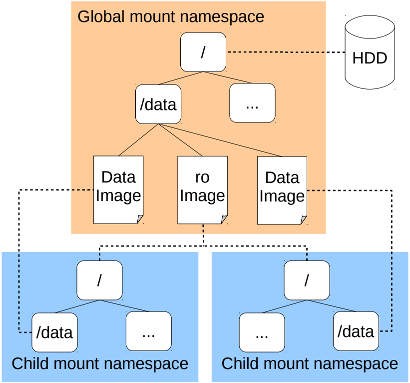
Fraunhofer AISEC

Container Isolation Mechanisms
Linux Namespaces (UTS)

- UTS Namespace (CLONE_NEWUTS)
- Isolate *nodename* and *domainname*
- Allow different hostname and domainname for each container
- Syscalls: uname(), sethostname, setdomainname()

Container Isolation Mechanisms
Linux Namespaces (MOUNT)

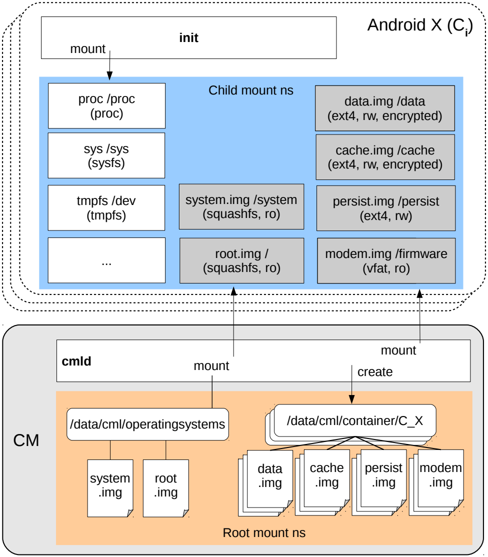
- Mount namespace (CLONE_NEWNS)
- Separate view on file system hierarchy
- It allows to jail a process in a new root directory
- Similar to chroot but better isolated (hide underlying systems mount view)
- Syscalls: mount(), umount()



Fraunhofer AISEC

Container Isolation Mechanisms
Linux Namespaces (MOUNT)

- trust|me storage setup using mount namespaces
- Read-only system images
- Per container encrypted data images
- Mounted and created by container management daemon (*cmld*)
- Pseudo file systems (proc, …) mounted by child's *init*



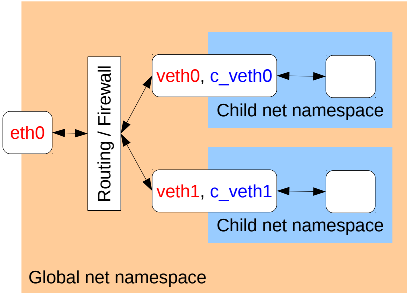
Container Isolation Mechanisms
Linux Namespaces (IPC)

- IPC Namespace (CLONE_NEWIPC)
- Isolate resources for interprocess communication
- System V IPC objects and POSIX message queues
- UNIX domain sockets are not affected by this namespace
  - ⇒ UNIX sockets can be used for intra-container communication
  - File-based IPC: either using a shared file system
  - or Parent PID namespace creates socket and provides file descriptor as argument to child
- Binder (Android IPC) also not affected
  - trust|me uses cell's device namespace to separate binder IPC

Fraunhofer AISEC

Container Isolation Mechanisms
Linux Namespaces (NET)

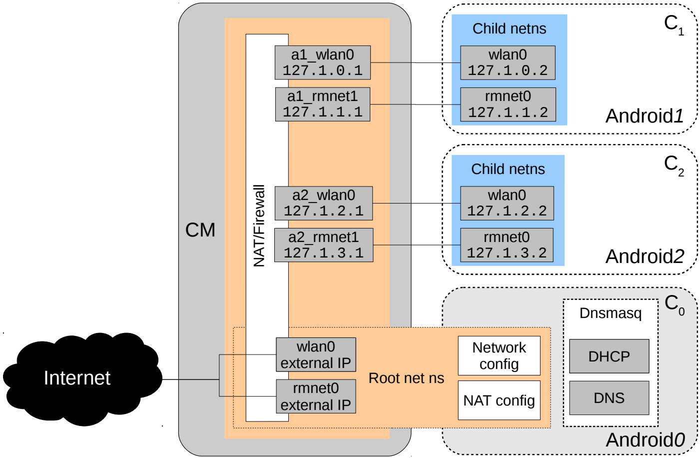
- Network namespace (CLONE_NEWNET)
- Virtual network stack for each namespace
- Own Routing tables, firewall, network interfaces (veth), /proc/net
- Allowing to have network servers on the same port
- Root namespace must provide NAT or bridge setup for outside connectivity



Fraunhofer AISEC

Container Isolation Mechanisms
Linux Namespaces (NET) cont'd

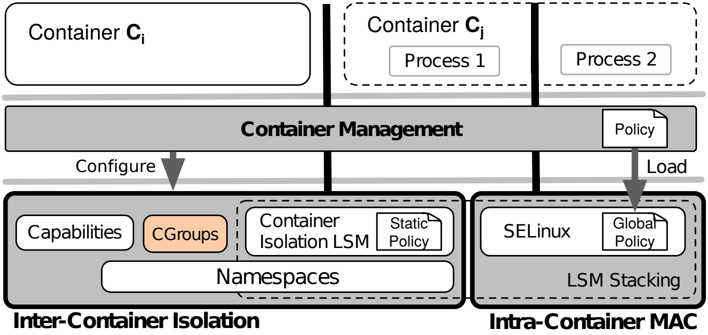
- trust|me network setup using network namespaces
- Local net routing (127.1.x.x/24) to avoid IP clashes with private subnets



Container Isolation Mechanisms
Linux Namespaces (USER)

- User namespace (`CLONE_NEWUSER`)
- Available since 3.8
- Not utilized in trust|me (since current kernel version 3.4 does not support them)
- Allowing root user (uid 0) inside an unprivileged application
- Process has full root privileges inside its user namespace
- Unprivileged outside the namespace

Container Isolation Mechanisms
Linux CGroups



Container Isolation Mechanisms
Linux CGroups

- CGroup subsystems
  - blkio – limit i/o access to block devices (HDD, USB)
  - **cpu** – scheduler configuration, limit CPU utilization of tasks in a cgroups
  - **cpuacct** – automatic reports on CPU resources by task in cgroup
  - cpuset – assign CPUs and memory nodes (NUMA) to tasks in a cgroup
  - **devices** – allow/deny access to devices
  - freezer – suspend or resume tasks in a cgroup
  - **memory** – set memory limits for tasks in a cgroup
  - net_cls – tag network packets
- trust|me utilzes cpu, memory and devices subsystems in CM
- The cpuacct subsystem is mounted for compatibility inside the containers

Container Isolation Mechanisms
Linux CGroups

- CGroups are managed over a virtual file system
- CGroup files can be written by root
- *init* is started as root inside the container

$\Rightarrow$ Without a user namespace it is feasible to change the cgroup configuration inside a container

- The trust|me isolation LSM prohibits the mounting of cgroups in unprivileged containers $C_i, i > 0$

Fraunhofer AISEC

Container Isolation Mechanisms

Linux CGroups

trust|me example: virtual file system for a container C_i view from CM

```
/ # ls /sys/fs/cgroup/trustme-containers/<C_i>/
cgroup.clone_children       memory.max_usage_in_bytes
cgroup.event_control        memory.memsw.failcnt
cgroup.procs                memory.memsw.limit_in_bytes
cpu.notify_on_migrate       memory.memsw.max_usage_in_bytes
cpu.rt_period_us            memory.memsw.usage_in_bytes
cpu.rt_runtime_us           memory.move_charge_at_immigrate
cpu.shares                  memory.oom_control
devices.allow               memory.soft_limit_in_bytes
devices.deny                memory.stat
devices.list                memory.swappiness
freezer.state               memory.usage_in_bytes
memory.failcnt              memory.use_hierarchy
memory.force_empty          notify_on_release
memory.limit_in_bytes       tasks
```

Container Isolation Mechanisms
Linux CGroups – devices

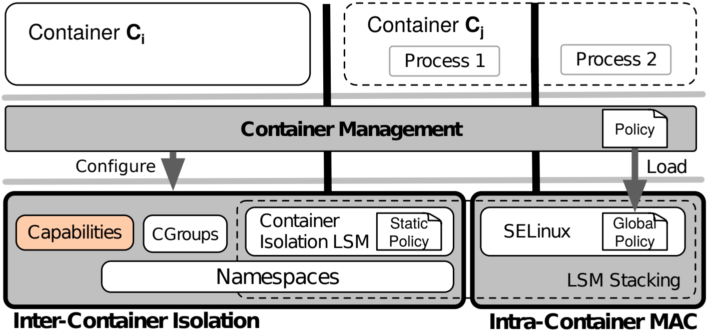
- The devices subsystem is essential for isolating access to device nodes
- White list approach on major minor basis
- A rule consists of
  - the devices type (block or char)
  - major and minor number (can be a wildcard)
  - access types (read, write, mknod)
- E.g, allow cgroup 1 to read and mknod /dev/null
  ```
  echo ’c 1:3 mr’ > /sys/fs/cgroup/1/devices.allow
  ```
- E.g., allow all
  ```
  echo a > /sys/fs/cgroup/1/devices.allow
  ```

Container Isolation Mechanisms
Linux CGroups

- Generic whitelist in *cmld* of devices to be available for all containers
- independent from hardware and container configuration
- merged with hardware specific one
- *cmld* → /sys/fs/cgroup/trustme-containers/devices.allow

```
static const char *c_cgroups_devices_generic_whitelist[] = {
        /* Memory Devices */
        //"c 1:1 rwm", // physical mem
        //"c 1:2 rwm", // kmem
        "c 1:3 rwm", // null
        "c 1:5 rwm", // zero
        "c 1:7 rwm", // full
        "c 1:8 rwm", // random
        "c 1:9 rwm", // urandom
        "c 1:11 rwm", // kmsg
        /* [...] */
        NULL
};
```

Container Isolation Mechanisms
Linux POSIX Capabilities



Container Isolation Mechanisms
Linux POSIX Capabilities

- A process has three sets of bitmaps:
  1. inheritable (***I***) capabilities
  2. permitted (***P***) capabilities
  3. effective (***E***) capabilities
- A capability is implemented as bit in each of these bitmaps
- ***P*** contains all capabilities a process is allowed to use
- ***I*** is used as mask for child processes (exec) against the permitted set
- ***E*** contains the currently activated capabilities of a process
- On fork/clone the child gets an copy of these bitmaps of the parent
- If process uses an privileged operation the kernel checks if the required bit is set in ***E***
- Capabilities can be dropped in the child process before the actual execve() syscall

Container Isolation Mechanisms

Linux POSIX Capabilities dropped in trust|me

```
cap_start_child(const container_t *container)
{
        /* 9 */ C_CAP_DROP(CAP_LINUX_IMMUTABLE);
        /* 14 */ C_CAP_DROP(CAP_IPC_LOCK);
        /* 15 */ C_CAP_DROP(CAP_IPC_OWNER);
        /* 16 */ C_CAP_DROP(CAP_SYS_MODULE); /* forbid module loading out of containers */
        /* 18 */ C_CAP_DROP(CAP_SYS_CHROOT);
#ifndef DEBUG_BUILD
        /* 19 */ C_CAP_DROP(CAP_SYS_PTRACE);
#endif
        /* 20 */ C_CAP_DROP(CAP_SYS_PACCT);
        /* 21 */ C_CAP_DROP(CAP_SYS_ADMIN); /* forbid system administrative commands */
        /* 22 */ C_CAP_DROP(CAP_SYS_BOOT); /* forbid container's to globally reboot*/
        /* 23 */ C_CAP_DROP(CAP_SYS_NICE);
        /* 26 */ C_CAP_DROP(CAP_SYS_TTY_CONFIG);
        /* 28 */ C_CAP_DROP(CAP_LEASE);
        /* 29 */ C_CAP_DROP(CAP_AUDIT_WRITE);
        /* 30 */ C_CAP_DROP(CAP_AUDIT_CONTROL);
        /* 31 */ C_CAP_DROP(CAP_SETFCAP);
        /* 32 */ C_CAP_DROP(CAP_MAC_OVERRIDE);
        /* 33 */ C_CAP_DROP(CAP_MAC_ADMIN);
        /* 34 */ C_CAP_DROP(CAP_SYSLOG);
        ///* 35 */ C_CAP_DROP(CAP_WAKE_ALARM); /* needed by alarm driver */

        /* Use the following for dropping caps only in unprivileged containers */
        if (!container_is_privileged(container)) {
        /* 25 */ C_CAP_DROP(CAP_SYS_TIME);
        }
   return 0;
```

Container Isolation Mechanisms

Linux POSIX Capabilities (CAP_SYS_MKNOD)

- Allows to create device nodes in file system
- Usually this is a security critical capability
- trust|me explicitly allows `CAP_SYS_MKNOD` for unprivileged containers
- CGroups devices are used to restrict device access

⇒ A container can run its own udev daemon to set up `/dev`

Container Isolation Mechanisms
Linux POSIX Capabilities (`CAP_SYS_ADMIN`)

- Some critical operations included in (`CAP_SYS_ADMIN`)
  - Allow mount() and umount(), setting up new smb connection
  - Allow locking/unlocking of shared memory segment
  - Allow setting encryption key on loopback file system Allow administration of the random device
  - Allow examination and configuration of disk quotas
  - Allow mount() and umount(), setting up new smb connection
  - ...
- Android needs mount syscall for its storage subsystem

⇒ trust|me introduces `CAP_SYS_MOUNT` to safely drop `CAP_SYS_ADMIN`

Fraunhofer AISEC

Container Isolation Mechanisms

Linux POSIX Capabilities (`CAP_SYS_TIME`)

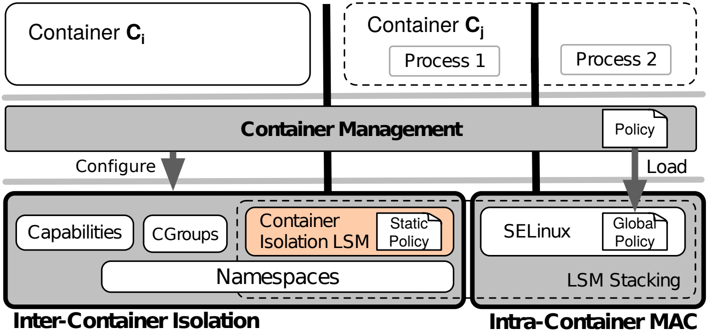
- Allows to manipulate system clock and real time clock
- Android does not use the intended mechanisms to set the system time
- The `/dev/alarm` driver is used to set the time and timezone
- No capability check is implemented!

⇒ trust|me introduces an security hook in the Android alarm driver

- Container Isolation LSM prevents time setting

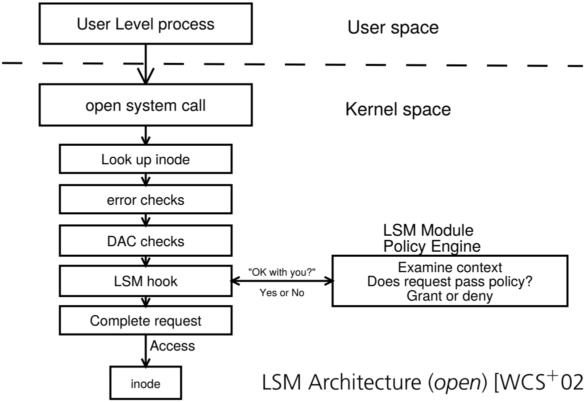
Container Isolation Mechanisms

Linux Security Module



Container Isolation Mechanisms
Linux Separation Mechanisms for Containers

- Linux Security Modules (LSM)
- A LSM can implement further security checks to access on kernel data structures



LSM Architecture (*open*) [WCS+02]

- Several security hooks in Kernel for:
  - Task
  - Program Loading
  - File System
  - IPC
  - Module
  - Network

Fraunhofer AISEC

Container Isolation Mechanisms
trust|me Container Isolation LSM

- Privileged vs. unprivileged containers
  - Securityfs-based userland interface to drop privileges of a pid namespace
  - Flag inside the pidns struct
  - *cmld* sets the privilege flag for containers
  - C_0 is the only privileged container
- Netlink socket protection
  - **socket** hooks to deny global uevent listening for containers on netlink socket
  - Filter of uevents is done by *cmld* and provided through UNIX domain socket
- Time management protection
- Path-based access protection
- Mount white listing

Container Isolation Mechanisms

trust|me Container Isolation LSM (time)

```
// trustme/hooks.c
static int trustme_android_alarm_set_rtc(void)
{
        if (trustme_pidns_is_privileged(task_active_pid_ns(current)))
                return 0;
        return -1;
}
// [...]
static struct security_hook_list trustme_hooks[] = {
    // [...]
    LSM_HOOK_INIT(android_alarm_set_rtc, trustme_android_alarm_set_rtc),
    // [...]
};

// alarm-dev.c
static long alarm_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
  // [...]
        case ANDROID_ALARM_SET_RTC:
                if (security_android_alarm_set_rtc() < 0) {
                        return -EPERM;
                }
  // [...]
}
```

- **Alarm** hook is checking if container is privileged
- Denies all unprivileged containers to manipulate system time

Container Isolation Mechanisms

trust|me Container Isolation LSM (path-based)

```
static struct security_hook_list trustme_hooks[] = {
    // [...]
    /* path and file */
    LSM_HOOK_INIT(path_unlink, trustme_path_unlink),
    LSM_HOOK_INIT(path_mkdir, trustme_path_mkdir),
    LSM_HOOK_INIT(path_rmdir, trustme_path_rmdir),
    LSM_HOOK_INIT(path_mknod, trustme_path_mknod),
    LSM_HOOK_INIT(path_truncate, trustme_path_truncate),
    LSM_HOOK_INIT(path_symlink, trustme_path_symlink),
    LSM_HOOK_INIT(path_link, trustme_path_link),
    LSM_HOOK_INIT(path_rename, trustme_path_rename),
    LSM_HOOK_INIT(path_chmod, trustme_path_chmod),
    LSM_HOOK_INIT(path_chown, trustme_path_chown),
    LSM_HOOK_INIT(path_chroot, trustme_path_chroot),
    LSM_HOOK_INIT(file_open, trustme_file_open),
    LSM_HOOK_INIT(file_ioctl, trustme_file_ioctl),
    LSM_HOOK_INIT(file_fcntl, trustme_file_fcntl),
    LSM_HOOK_INIT(inode_getattr, trustme_inode_getattr),
    // [...]
};
```

- **Path-based** hooks call
  `static int trustme_path_decision(struct path *path)`
- This checks if the path is allowed in white list and is not blacklisted
- "/sys/devices/leds-*" is included in blacklist ⇒ LED cannot be set by containers

Fraunhofer AISEC

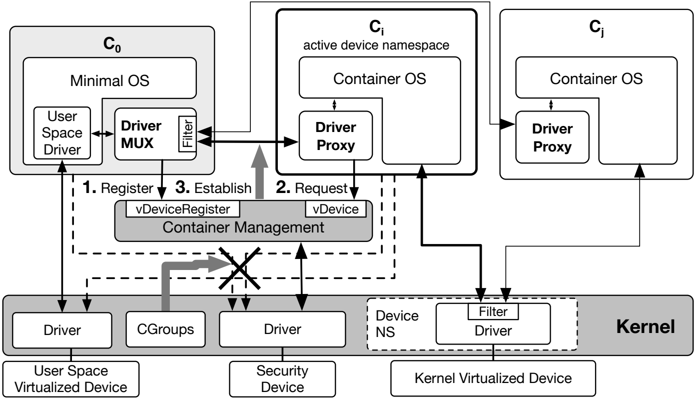
Container Isolation Mechanisms

trust|me Container Isolation LSM (mount)

```
static struct security_hook_list trustme_hooks[] = {
    // [...]
    /* superblock */
    LSM_HOOK_INIT(sb_mount, trustme_sb_mount),
    LSM_HOOK_INIT(sb_umount, trustme_sb_umount),
    // [...]
};
```

- **Mount** hooks are checking the mount white list
- The white list only allows for pseudo file systems (proc and sysfs) to the specific location
- Otherwise the path-based security approach above would not work
- Fuse mounts for emulated storage is allowed
- CGroups are not allowed to be mounted, except the cpuacct subsystem
- Otherwise the container's root processes would be able to manipulate the RAM limits or scheduler

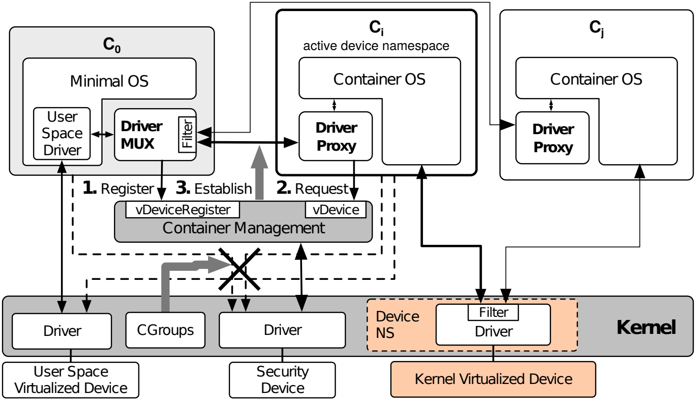
Secure Device Virtualization



- Security & non-virtualized devices: only for CM or C_F (enforced by cgroups)
- Kernel virtualized devices: device namespace (*active* namespace)
- User space virtualized devices: Driver MUX & Proxy to access user space drivers. Socket pair set up by CM (`socketpair`)

Secure Device Virtualization

Kernel space virtualized device



Fraunhofer AISEC

Secure Device Virtualization
Kernel space virtualized device

- Some device subsystems can be virtualized inside the kernel
- trust|me uses so called **device namespaces** (not mainline)
- Formerly introduced by the Columbia University with their *cells* [ADH+11] approach
- Introduces the **active** state
- Needs adoption of the corresponding drivers
- Isolate some driver states in specific driver device namespace struct
- Replace access to former global states to namespace struct, e.g., during IOCTLs
- Used for alarm, binder, logger
- Only allow access to the active namespace
- Used for input (evdev, mousedev), backlight lcd, graphics

Secure Device Virtualization

Kernel space virtualized device (binder)

```
\\ [...]
static HLIST_HEAD(binder_procs);
\\ [...]
static struct binder_node *binder_context_mgr_node;
static uid_t binder_context_mgr_uid = -1;
static int binder_last_id;
static struct workqueue_struct *binder_deferred_workqueue;
\\ [...]
```

↳

move static (device global) variables to namespace structure

```
\\ [...]
static struct workqueue_struct *binder_deferred_workqueue;
struct binder_dev_ns {
        struct binder_node      *context_mgr_node;
        uid_t                   context_mgr_uid;
        int                     last_id;

        struct hlist_head       procs;
        struct hlist_head       dead_nodes;

        struct dev_ns_info      dev_ns_info;
};
static void binder_ns_initialize(struct binder_dev_ns *binder_ns)
{
        INIT_HLIST_HEAD(&binder_ns->procs);
        INIT_HLIST_HEAD(&binder_ns->dead_nodes);

        binder_ns->context_mgr_uid = -1;
}
\\ [...]
```


Fraunhofer AISEC

Secure Device Virtualization

Kernel space virtualized device (binder)

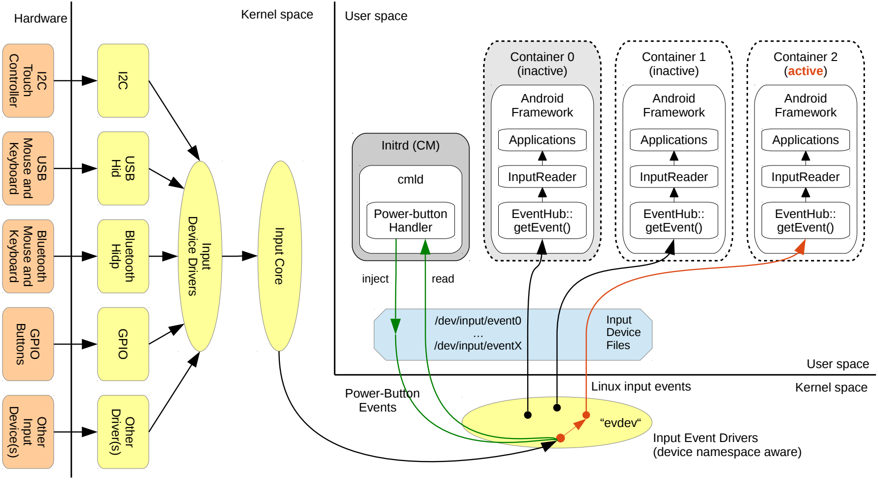
- When the driver is opened, `binder_open()` gets called

```
static int binder_open(struct inode *nodp, struct file *filp)
{
    // [...]
    binder_ns = get_binder_ns_cur();
    // [...]
    proc = kzalloc(sizeof(*proc), GFP_KERNEL);
    // [...]
    proc->binder_ns = binder_ns;
    // [...]
    filp->private_data = proc;
    // [...]
}
```

- The IOCTL request then uses the private data field of the file descriptor to point to the namespace aware `struct binder_proc`

```
static long binder_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    // [...]
    struct binder_proc *proc = filp->private_data;
    // [...]
}
```

Secure Device Virtualization
Kernel space virtualized device (input)



Fraunhofer AISEC

Secure Device Virtualization

Kernel space virtualized device (graphics)

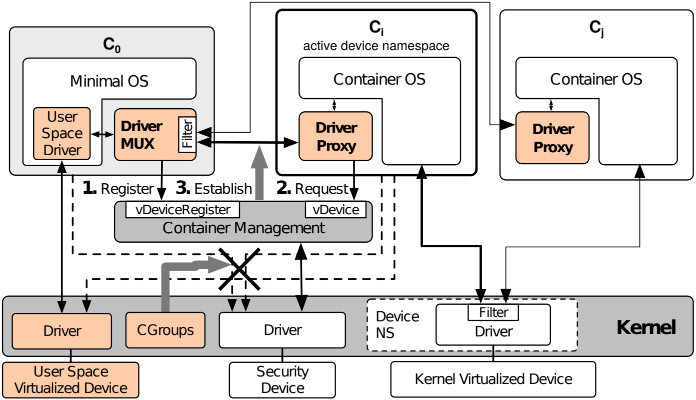
- This case is special as graphics virtualization usually is complex
- trust|me does not duplicate the data structures of the framebuffer driver
- Simple check of the active device namespace in the generic `fb_ioctl` handler
- Background container C_B get an access denied
- Switching procedure takes care about suspending the graphics subsystem

```
// fbmem.c
static long fb_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
        struct fb_info *info = file_fb_info(file);
        if (!info)
                return -ENODEV;

        if (!is_active_dev_ns(current_dev_ns()))
                return -EPERM;

        return do_fb_ioctl(info, cmd, arg);
}
```

Secure Device Virtualization
User space virtualized device



Secure Device Virtualization
User space virtualized device

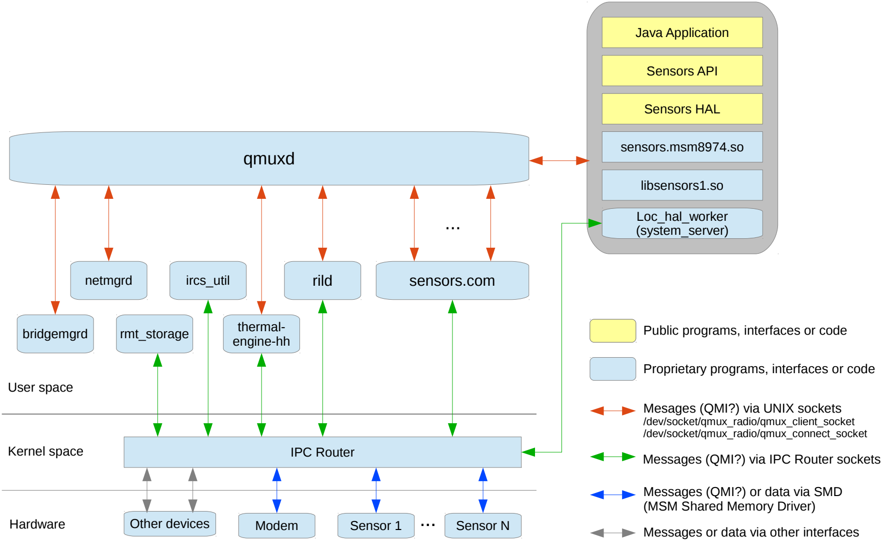
Why do we need user space virtulized drivers?

- Mobile phones use proprietary user space components
- Nexus 5 uses a Quallcom SoC
- Quallcom uses special user space components to communicate between several subsystems
- E.g., sensors, gps, thermal management, power management and the modem

⇒ its not feasible to virtualize them in the kernel

- trust|me runs these user space drivers unmodified in C_0
- C_0 provide a Driver MUX including a Filter
- In C_i the corresponding Driver Proxy is implemented as library inside the *system_server*
- CGroup devices subsystem prohibits access to the corresponding kernel driver in C_i

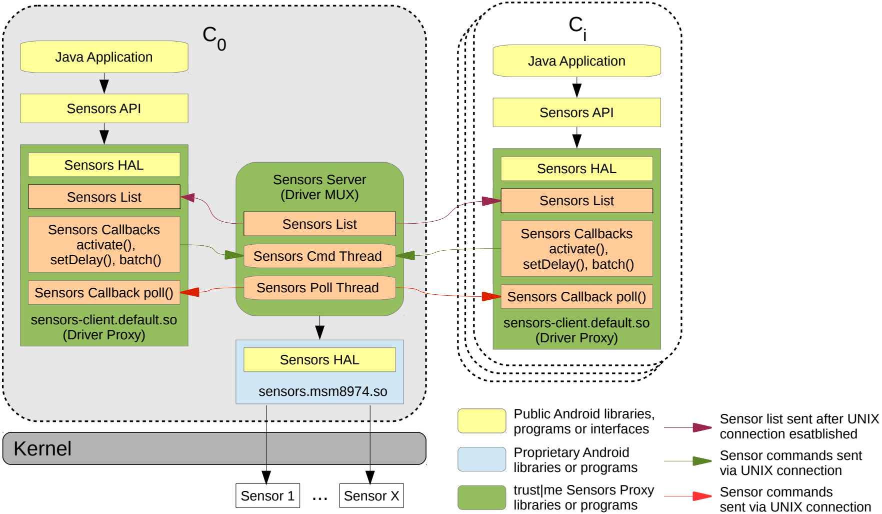
Secure Device Virtualization
User space virtualized device (sensors)



Fraunhofer AISEC

Secure Device Virtualization

User space virtualized device (sensors)

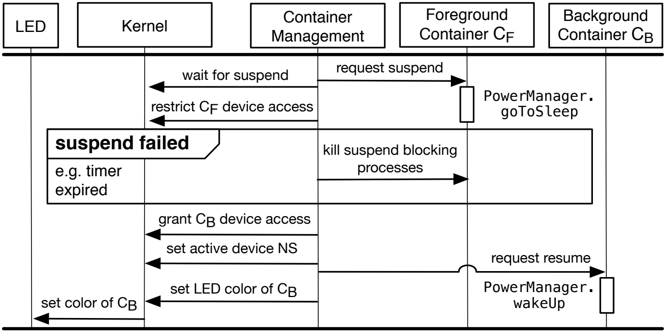


Fraunhofer AISEC

Secure Container Switch

Switching

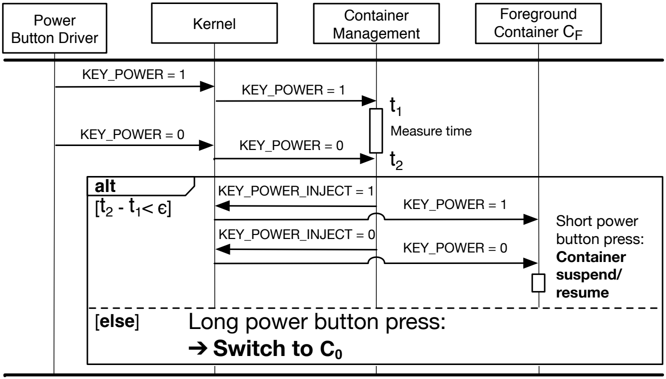
Secure Container Switch
Switching Procedure



- Cgroups devices dynamic reconfiguration
- Demand container to suspend, otherwise suspend forcefully
- CM sets LED color to identify C_F

Fraunhofer AISEC

Secure Container Switch
Secure Switch Initiation



- Switch from C_0 to $C_{1..n}$ via Trusted GUI
- Spoofing C_0 prevented due to LED color setting
- Switch from $C_{1..n}$ to C_0 via power button

Conclusions

- **Secure architecture** for OS-level virtualization
- Provides data confidentiality **at container boundaries**
- **Systematic isolation** of containers. **Minimal functionality** and communication
- Architecture targets **whole platform**, incl. SE, secure device virtualization, switching
- **Fully-functional** implementation, applicable in real-life

Bibliography

[ADH+11] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh.
Cells: A virtual mobile smartphone architecture.
In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 173–187, New York, NY, USA, 2011. ACM.

[HHV+15] Manuel Huber, Julian Horsch, Michael Velten, Michael Weiss, and Sascha Wessel.
A secure architecture for operating system-level virtualization on mobile devices.
In *Proceedings of the 11th International Conference on Information Security and Cryptology (Inscrypt 2015)*, 2015.

[WCS+02] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman.
Linux security modules: General security support for the linux kernel.
In *Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2002. USENIX Association.

Contact Information

Dr. Michael Weiß

Fraunhofer-Institute for
Applied and Integrated Security (AISEC)

Address: Parkring 4
85748 Garching (near Munich)
Germany
Internet: http://www.aisec.fraunhofer.de

E-Mail: trustme@aisec.fraunhofer.de

Fraunhofer
AISEC