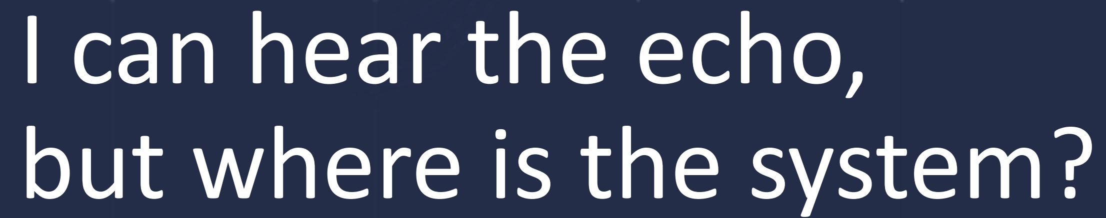
BLUEHALO



I can hear the echo, but where is the system?

Presented by : Renato Levy, D.Sc.

project team: Renato Levy, Ammar Salman, San Sowles, Todd Humiston

Distribution Statement "A" : Approved for Public Release, Distribution Unlimited

Disclaimer

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA)

(contract # W31P4Q-19-C-0050)

"The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government."

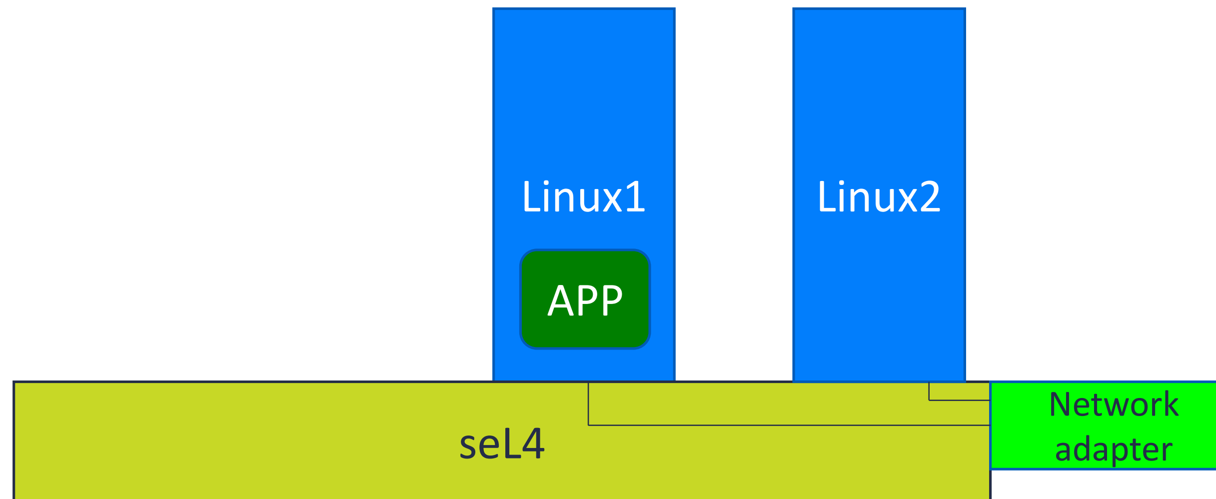
To Do List- seL4 Foundation

- Lower-cost approaches for verifying the non-kernel parts of the trusted computing base, such as device drivers, file and network services, but also the actual applications. So far, verified software is still more expensive to produce than the usual buggy stuff (although life-cycle cost is probably already competitive). Trustworthy Systems' declared aim is to produce verified software at a cost that's at par with traditionally engineered software;
- Proofs of high-level security properties of a complete system (as opposed to "just" the underlying microkernel);
- Design of a general-purpose operating system that is as broadly applicable as Linux, but where it is possible to prove security enforcement.

From Kernel to System

- seL4 is a micro-kernel, hence many system devices are outside its range
- CAmkES can be used for device configuration, but it will not prevent security leaks through these devices
- NOTHING can be implemented with Kernel alone
- Very few system hardware are fully supported
- Using Linux to create an executing environment, reduces security of each environment to Linux level

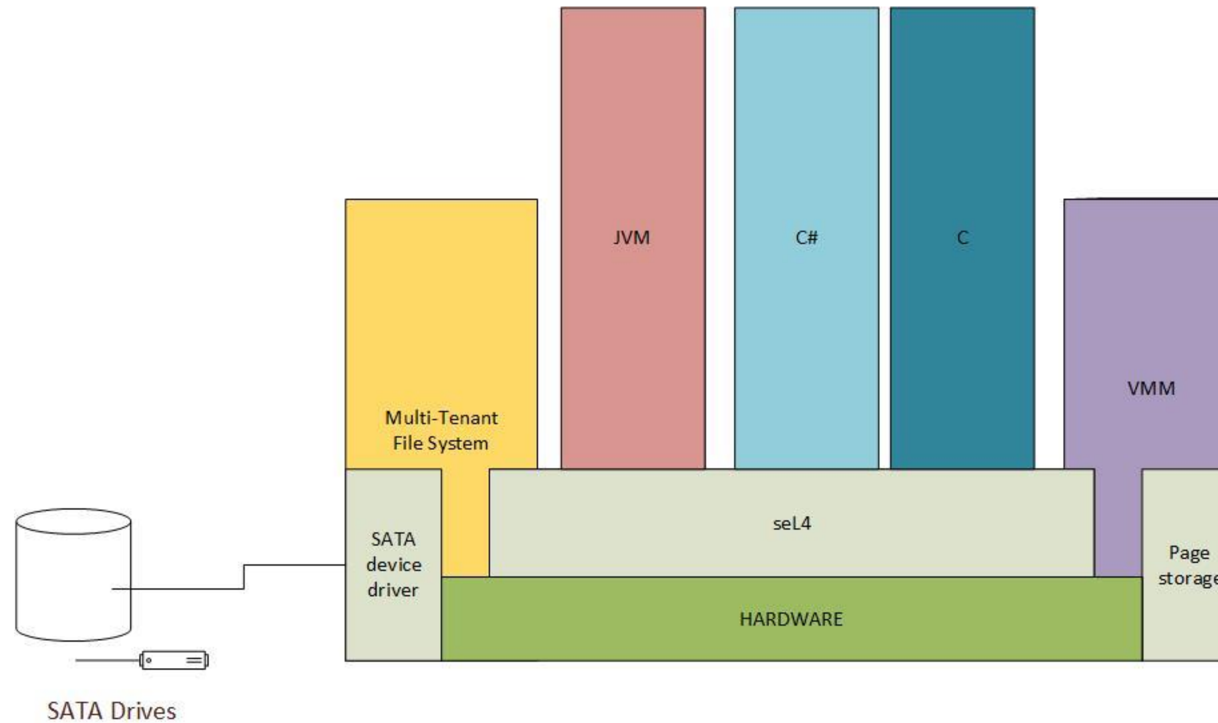
The weakest link



An Architecture for System Security

- Complete isolation from system components using the Kernel capabilities
- Each non-kernel hardware device/service is an independent multi-tenant component with its own pre-defined interface
- Support libraries are built to automatically provide functionality based on component’s interface API (stdlib.h, JVM, C#)
- At the end of its initialization the Kernel launches all device/service components
- All device/service components are seL4 natives! (verification a plus)
- Language based thin environments (docker style)

Isolation Architecture



Language based thin environment

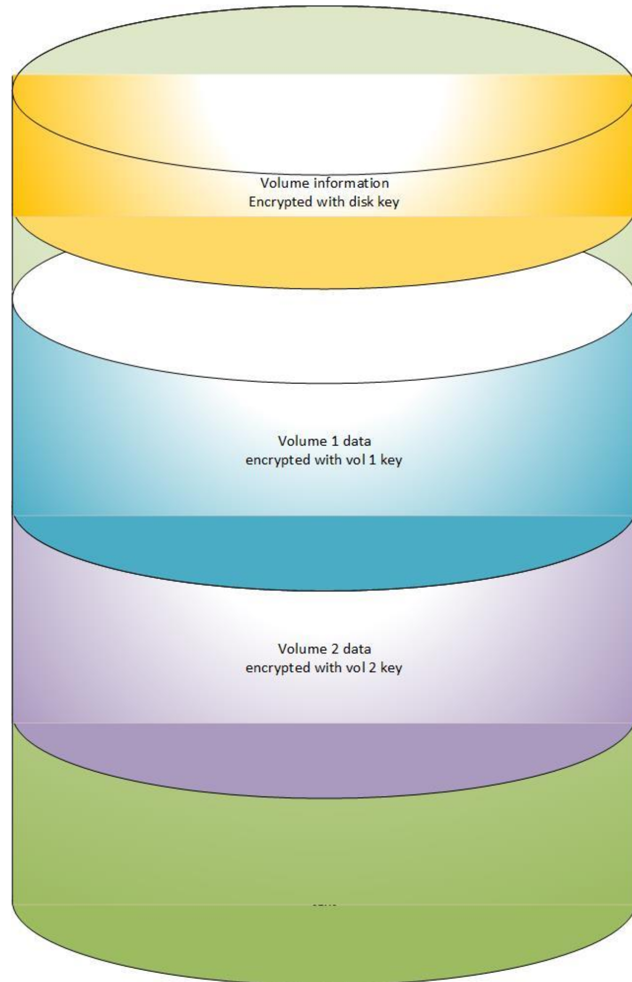
- Allows to launch a program on the language that runs natively on seL4 (just like Dockers)
- Executes programs as native language on seL4
  - (similar to what CAmkES does with C)
- VMs and libraries adapted to use isolation architecture
- Candidates:
  - Java
  - C#
  - Python
  - Rust

Example Component Device

Multi-tenant File System with encryption

- SATA device interface
- Single owner of all devices plugged in the interface
- Logical volumes
- Volume information is encrypted at disk
  (configuration must include key password)
- All volumes are encrypted at rest
- Execution environments must create/mount volumes explicitly
- Mounting volumes require knowledge of name/password

Disk organization



→Volume information on Disk, indicates the volume names, their key, initial directory page, and other critical volume information, encrypted with disk key

→ Volume data

→Volume data

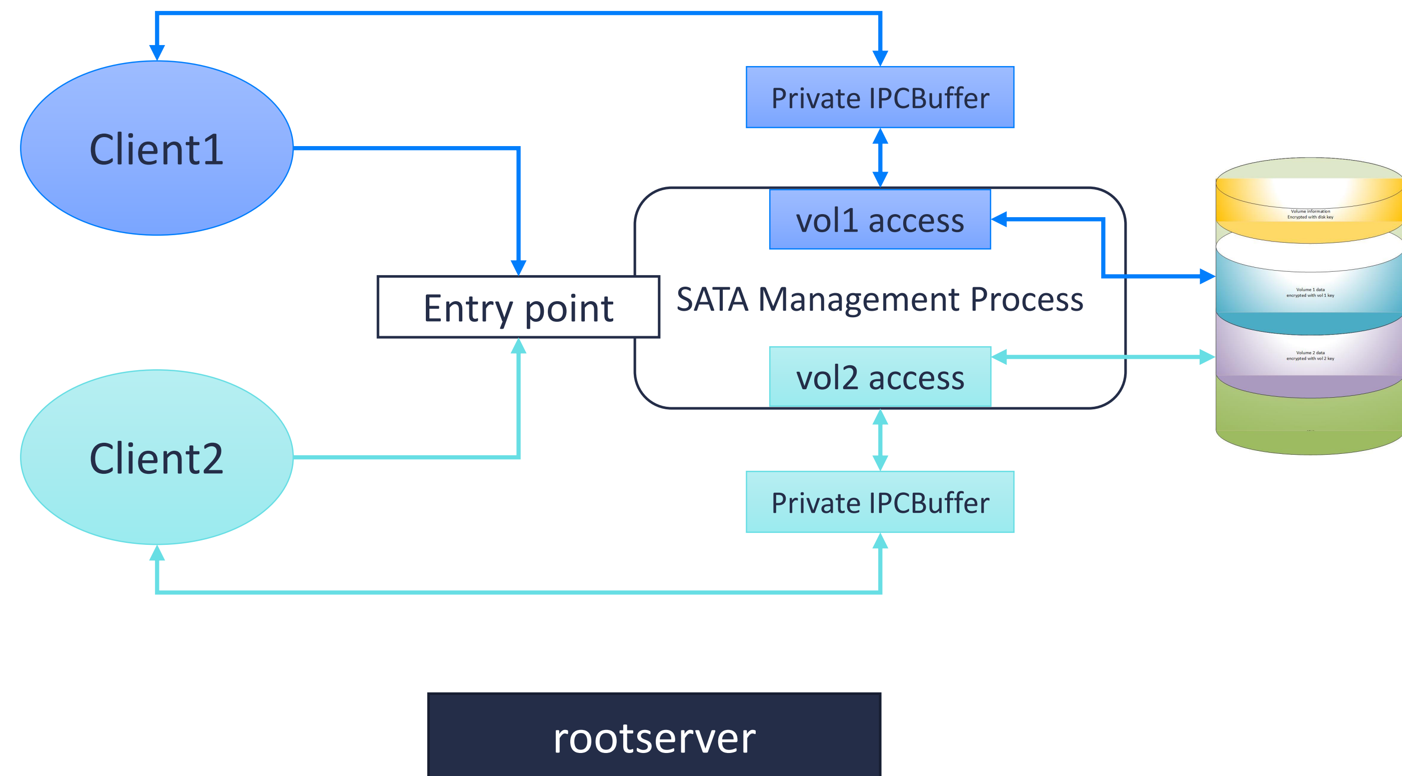
Secondary directory pages and files for each volume, non-continue logical volume with variable size, dispersed across actual disk, allocated in chunks, encrypted with volume key

File System

- No boot on disk
- Similar to ext3, having seL4 processes/environments as users
- Each process mounts volume to r/w, with encryption key as password using published service API
- If volume is mounted successfully, its contents becomes available for the process
- No file write access control in the initial version
- All files are assumed mod 777
- Support libraries for languages must be adapted to use service

Client Access Isolation to SATA

- Encapsulation over entry point, data transfer, and volume access.
- The disk is partitioned into logical volumes.
- If two clients share a volume, then an architecture dependency between the two clients is established, and the system cannot fully isolate one from another



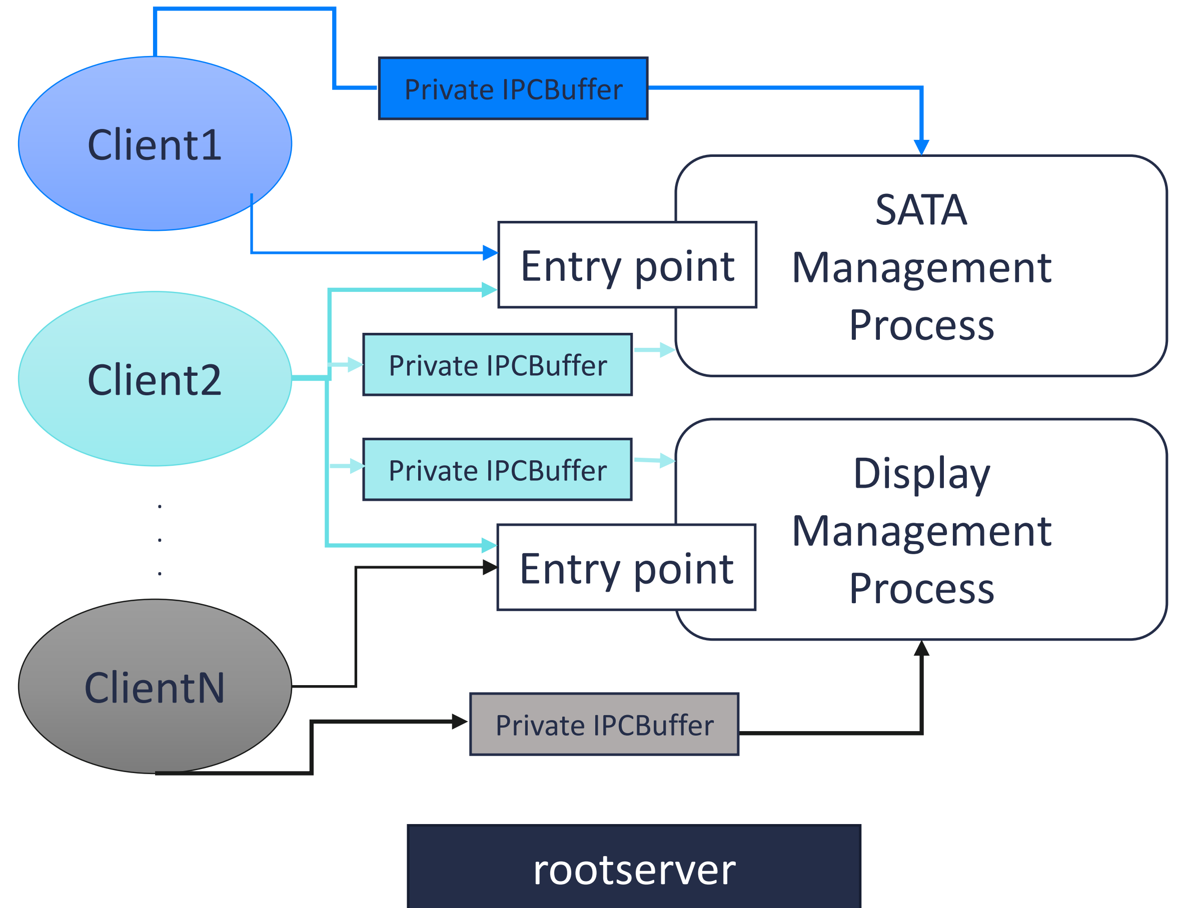
Device Service Framework

- Device drivers are initiated in separate processes
- Each device driver process manages all access to the device
  - Each client receives a private IPCBuffer which operates in fast mode and shared memory mode
    - Fast mode for quick requests
    - Shared memory mode for large data transfers (e.g. storage, display, …, etc.)
- Device processes have known endpoints which clients can hook into
- Client calls are abstracted and managed by the framework:
  - Clients import the framework and use the calls directly

    (E.g.: device_framework_request(DEVICE_SATA, …)
- The framework is integrated into the support libraries and can be linked to with ease.

Client Access to Framework

- Device processes execution logic is provided by the framework.
- Developers just need to specify which device processes are in use.
- Client access rights to devices can be predetermined by the developer.
- Client receives private IPCBuffer for data transfer. **Access can never be shared between clients.**



BLUEHALO

Proposed devices

- File system (SATA)
- Virtual Memory Manager (m2 storage)
- Graphic adapter (OpenGL library)
- Network adapter
- USB

Questions