Isolation as a Security Primitive

Building Real-World Critical Systems on a Formally Verified Microkernel

Martin Dehnel-Wild & Amy Brooks CEng

Chief Scientist; Technical Programme Manager, Kry10

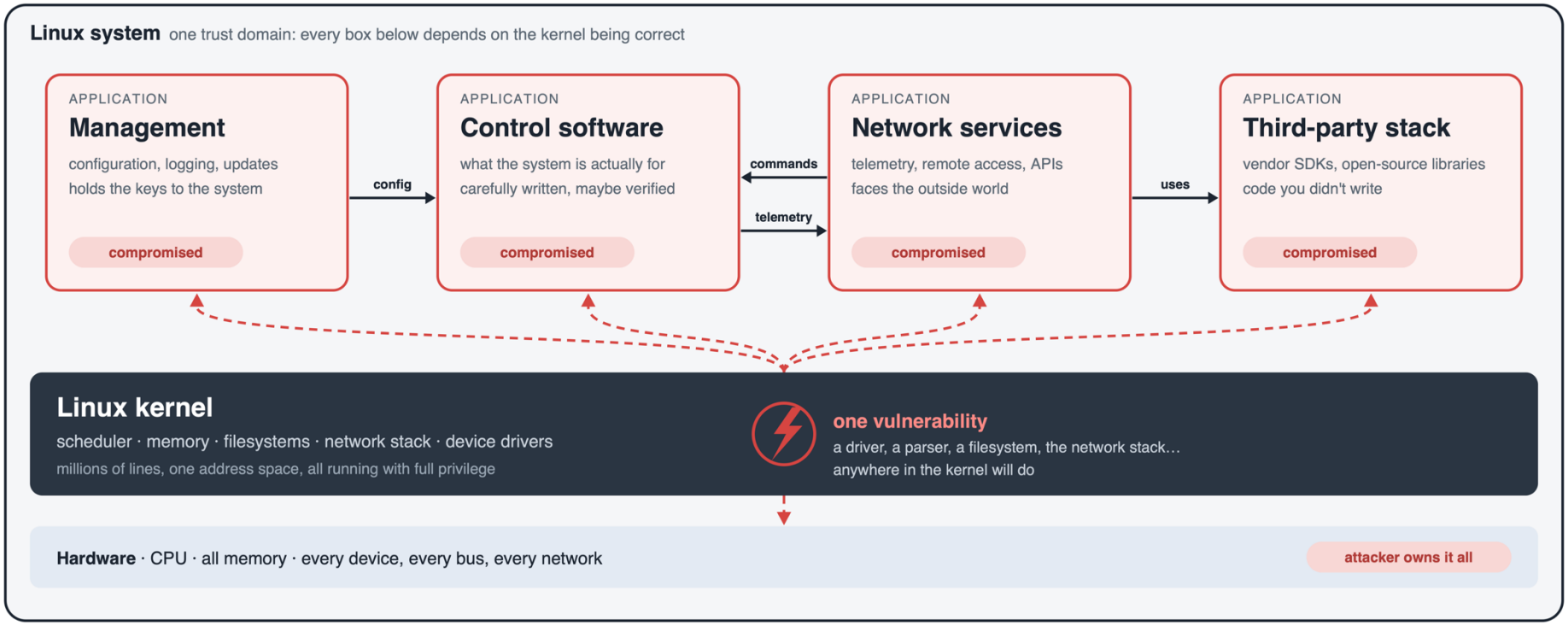
HISC 2026, Birmingham

KRY10

01
The Gap

The monolithic system: one bug, everything compromised

Applications, drivers and the kernel form a single trust domain. Whatever the kernel can touch, a bug in the kernel can touch.



Why one bug is enough. Process isolation, user accounts and file permissions are all enforced by the kernel. Compromise the kernel and nothing is left enforcing anything. The blast radius is the whole system, no matter how good the application code is.

Formal methods promise strong guarantees

But: proving every line of code *doesn't scale*.

What if proof covers the boundaries, instead of *all* the code?

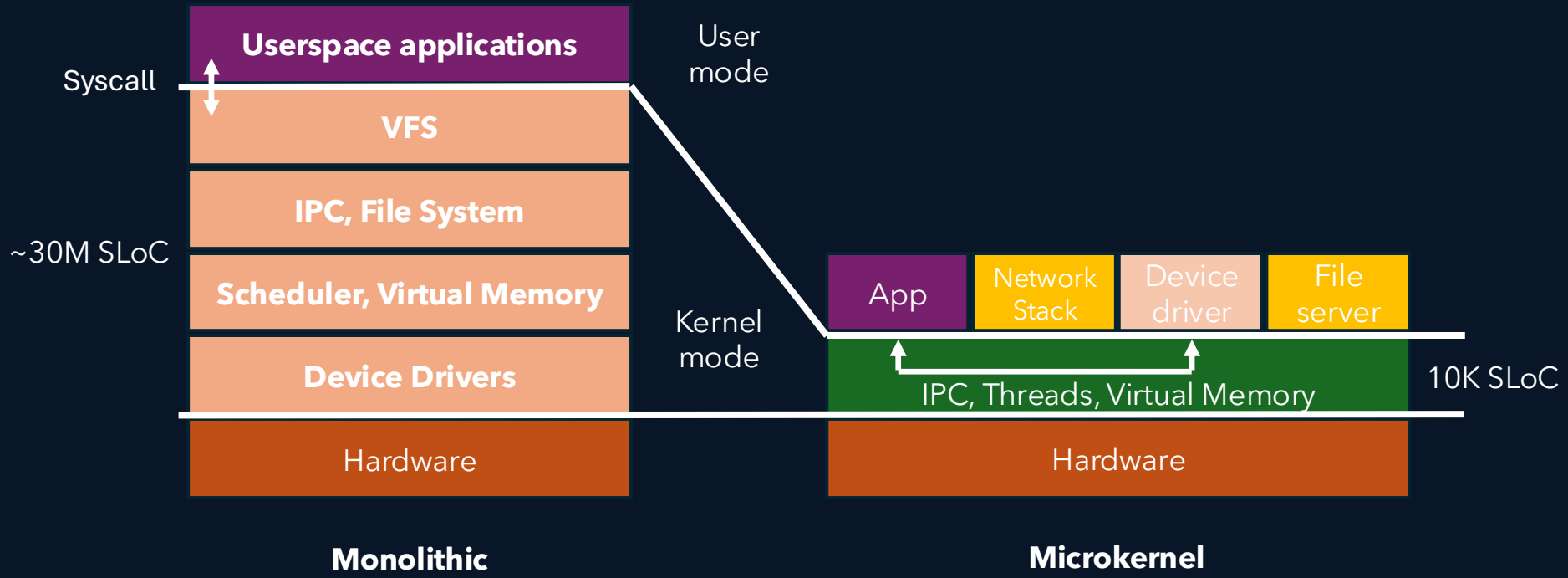
This talk

- Two different projects
- Two different sectors

- **Does this work in practice?**

02
Guarantees and limits

How a Microkernel Works (comparatively)



Source: "The seL4 Microkernel: An Introduction", Gernot Heiser

What is a *verified* microkernel?

- **seL4:** Only 10k SLoC
- Controls all access to all resources
- Any errors / vulnerabilities here would be catastrophic
- How do we build confidence?
- **Prove** the correctness and security of the kernel
- Other microkernels *claim* to enforce separation: seL4 proves it

KRY10

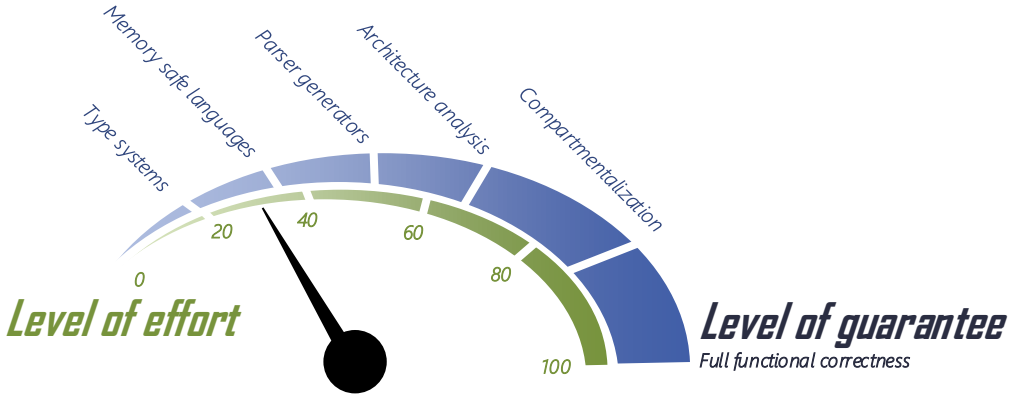
seL4
Security. Performance. Proof.

What does it NOT guarantee?

- Correctness of code inside a component.
- That the design / configuration is correct
- Behaviour of hardware is assumed correct
- Side channels; hardware paths outside the proof scope (e.g. DMA without an IOMMU).

Wide spectrum of formal methods

Rigorous, mathematical approaches that provide guarantees about computer-based systems



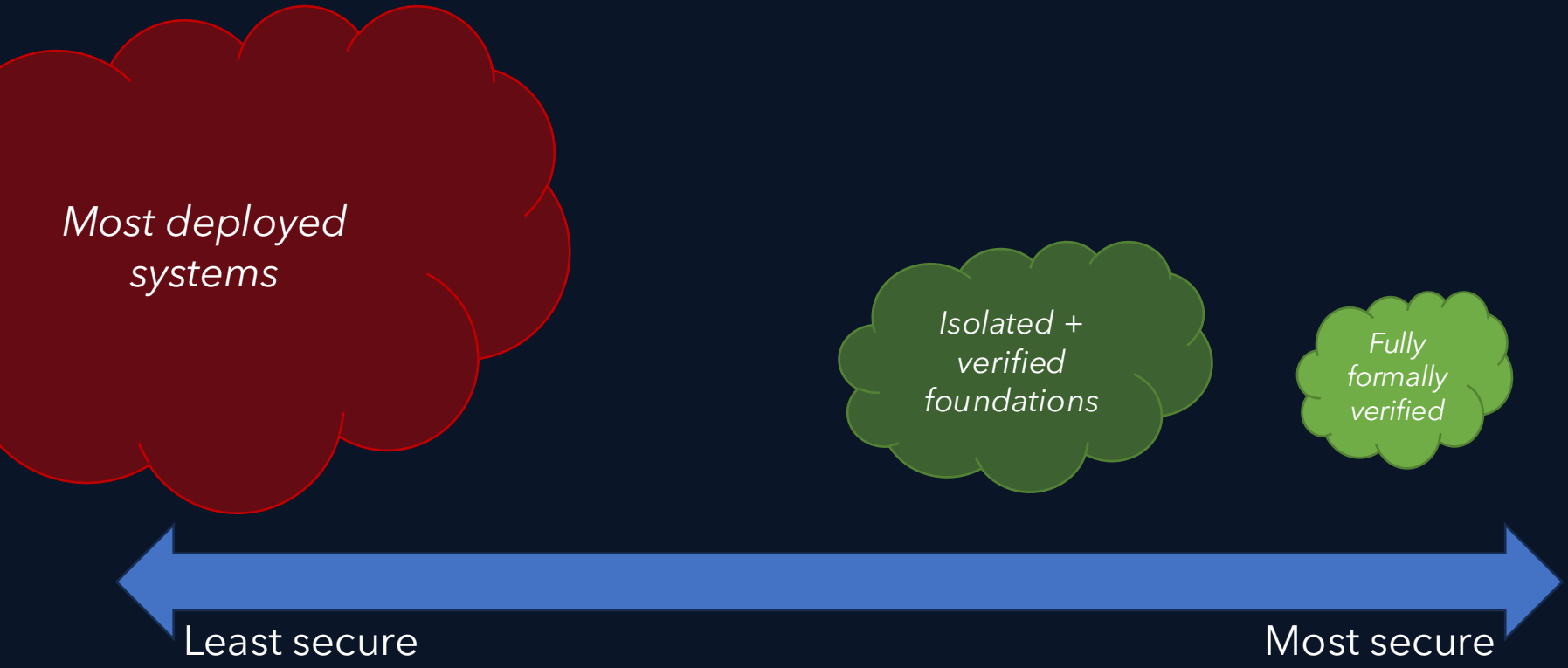
What CAN the system do?
What WILL the system do?
What can the system NEVER DO?

Formal methods

- Apply in many domains and each development phase
- Employ models of relevant parts of a system
- Prove guarantees expressed mathematically
- Sometimes live behind the scenes
- Leverage many various tools

Credit: Kathleen Fisher

Perfect is the enemy of good



KRYO

03

Software you can’t trust, contained


X500 V2

Retrofitting the drone onto KOS: PX4 split across three Linux microVMs

The existing Linux software runs unchanged. KOS decides what each microVM can touch, and what may pass between them.

KOS Kry10 OS on the seL4 microkernel

no path from WiFi to PX4

LINUX MICROVM

PX4

flight stack

estimation · control · mixing

no device access

Parser

native seL4 component

sensor data → PX4

motor outputs ← PX4

MAVLink telemetry

MAVLink RC input

everything else dropped

LINUX MICROVM

Flight I/O

sensor & motor bridge

RC receiver bridge

dashboard server

flight drivers

Parser

native seL4 component

HTTP dashboard

MAVLink ground station

WiFi mode control

everything else dropped

LINUX MICROVM

WiFi

hotspot + radio driver

serves the drone-control dashboard to the operator

untrusted: faces the air

UART

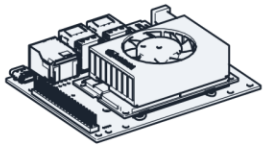
USB-UART

I²C

PCIe

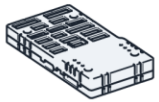
KOS services · static manifest · capabilities · supervision

seL4 microkernel · formally verified



NVIDIA Jetson Orin Nano

runs everything above; each device is passed to one microVM, its DMA fenced by the IOMMU



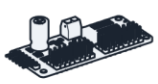
Pixhawk 6C

IMU · GPS · baro



RC receiver

ELRS radio link



PCA9685

I²C → PWM

ESCs



4 × motors

brushless



WiFi card + antennas

operator's phone / laptop

04
Verifying the boundary

"Targeting edge devices has now become a tactic of choice for many cyber threat actors, including state-sponsored actors."

– NCSC, 2026

The parser is the main attack surface

Guarding legacy OT with KOS: CNI Guardian inline in front of the field device

The protected device runs unchanged. KOS decides what each component can touch, and what may pass between them.

KOS Kry10 OS on the seL4 microkernel

no path from field-side NIC to device-side NIC

NATIVE KOS COMPONENT

Device-side NIC

Ethernet driver

forwards permitted traffic to the device, unmodified

only path to the device

NATIVE KOS COMPONENT

Firewall + Policy filter

least privilege on the wire

only approved message types, commands and sources reach each asset

out-of-policy traffic blocked

Parser

formally verified · EverParse

Modbus TCP
DNP3

protocol conformance

malformed traffic dropped

NATIVE KOS COMPONENT

Field-side NIC

Ethernet driver

receives all traffic from the field network

untrusted: faces the network

Ethernet

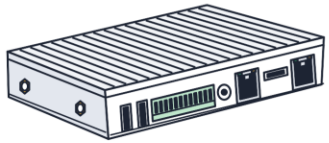
Ethernet

KOS services · static manifest · capabilities · supervision · watchdog

seL4 microkernel · formally verified

Legacy RTU / PLC

unchanged: no firmware, configuration or network changes



CompuLab
IOT-GATE-iMX8PLUS

runs everything above, inline in front of the field device; each NIC is owned by one component, its DMA fenced by the RDC

Field network

cellular · radio · leased line to SCADA / control centre





Credit: CompuLab

05
Patterns

What patterns did we see?

- Untrusted **code** → contain it
- Untrusted **input** → verify the boundaries

- Most real systems need both

- Different sectors , different constraints: network infrastructure vs real-time embedded
  - Both solutions rely on verified isolation

06
Lessons we learned

Lessons we learned

- Assurance by architecture, not re-writing
- Decomposition turns code review into configuration review
- Design for compromise, not against it

07
Takeaways

In conclusion

- Combining verified isolation and verified parsers make an attacker's job really hard
- This scales much better than verifying everything
- Two real-world projects that demonstrate this

- Choose the most critical points for verification effort

- **Resilience comes from containment, not from perfect code.**

KRY10

Resilience comes from containment, not from perfect code.

Thank you!

Martin Dehnel-Wild

(martin@kry10.com)

Amy Brooks CEng

(amy@kry10.com)



KRY10