KRY10

Kry10 Operating System

A primer for technical readers

COMMERCIAL
IN CONFIDENCE



PRIMER_SEP 2026

Security isn't a feature, it's *a foundation*



About Kry10

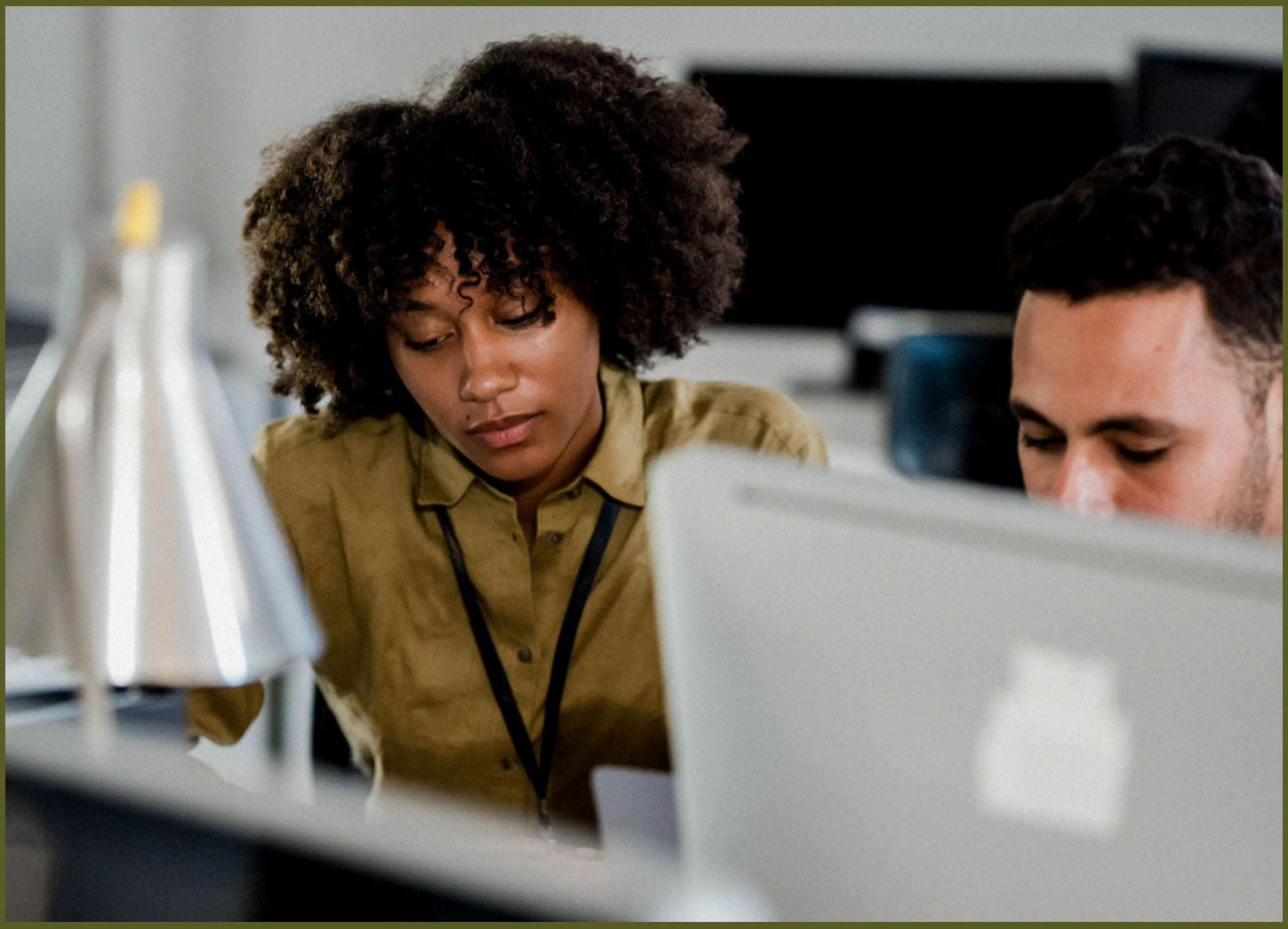
At Kry10, we strengthen the foundations beneath the critical systems that society depends on every day. We bring mathematical assurance to critical embedded and agentic systems, building resilience deep into the technology, right from the foundation.

We work with those responsible for critical systems. From the operators to architects to strategic leaders working across critical national infrastructure, healthcare, defense, telecommunications, transportation and more. All the people who can't afford to get it wrong.

Our focused and growing team across the USA, UK, Australia and New Zealand strengthens what matters for the critical systems society depends on every day.

Our foundations are mathematically proven. Because critical should mean certain.

©2026 KRY10
All rights reserved. Kry10 and associated names, logos, graphics, and product identifiers are trademarks or registered trademarks of Kry10. Any other trademarks referenced remain the property of their respective owners.



The systems that power modern life deserve more than conventional security. Kry10 provides mathematically proven foundations that help keep critical services resilient by design.

The machines that run the physical world — power grids, water treatment plants, hospital equipment, vehicles, industrial control systems — are increasingly controlled by software. While that's nothing new, the scale and speed of the threat presented by AI is reducing the costs for attackers to find and exploit vulnerabilities, and regulators are starting to assign real liability for insecure software. Meanwhile, the software running underneath most of these systems, at the foundational operating system layer, will have been built in the same way that software has always been built, by testing it until it appeared to work.

KOS is an operating system, built on formally verified foundations.

KOS is different. The secure foundation of KOS enables you to run applications right on the seL4® microkernel. There's nothing between the formally proven foundation and your workload. Only your workload, with no intermediary, no unnecessary code, no vulnerabilities. With a small and proven microkernel, you know what you can count on.



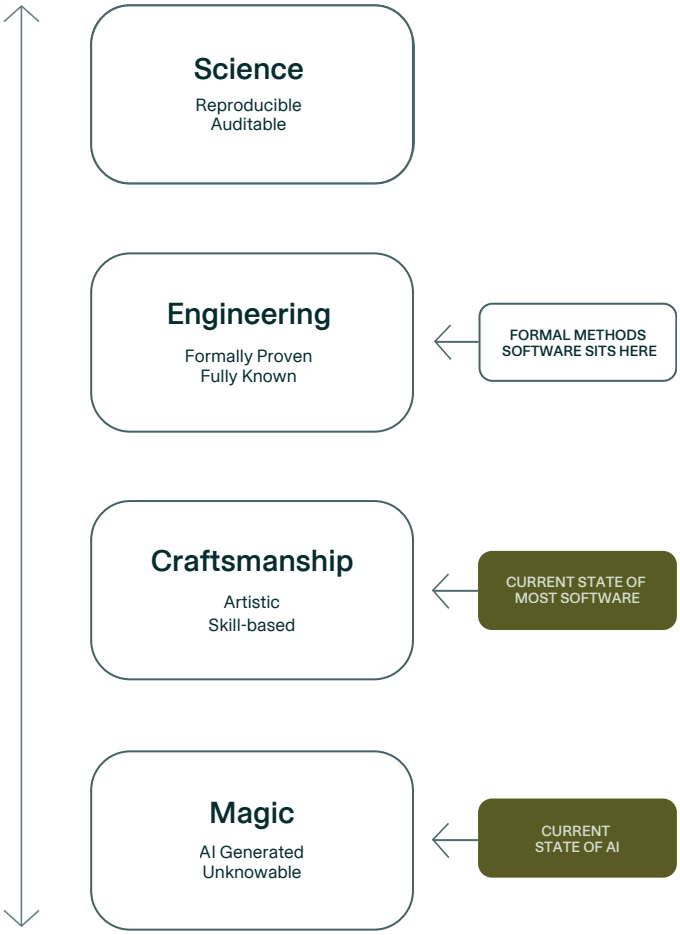
Because the only systems *you can trust have proof*

01 The Creation Continuum: 4 ways to build something

"With AI, we're currently in a world of magic and wizards. To operate with certainty and security, we need to move towards a world of science and engineering."

BOYD MULTERER, CEO, KRY10

THE 'CREATION CONTINUUM'



When considering how anything is built, Kry10's CEO Boyd Multerer speaks about a 'creation continuum'. At one end, science discovers things that are true about the world, independent of intent or application. At the other end, magic uses tools you don't understand. You just have to trust that they work.

In between, engineering takes discoveries from science and uses them to build things with evidence of correctness. Meanwhile, craftsmanship creates solutions through accumulated experience, best practice, and trial and error, without proof that what was built is actually right.

Most of the software industry, including software engineers, practices craftsmanship. They build and review code, utilize best practice, test and hope that the testing catches what matters. If you were to call an API where you haven't read the source or trust an AI model where no one fully understands the internals, that is magic. Almost none of the work done in the industry is engineering in this specific sense, and that's true industry-wide, not a criticism of any particular team, organization or product.

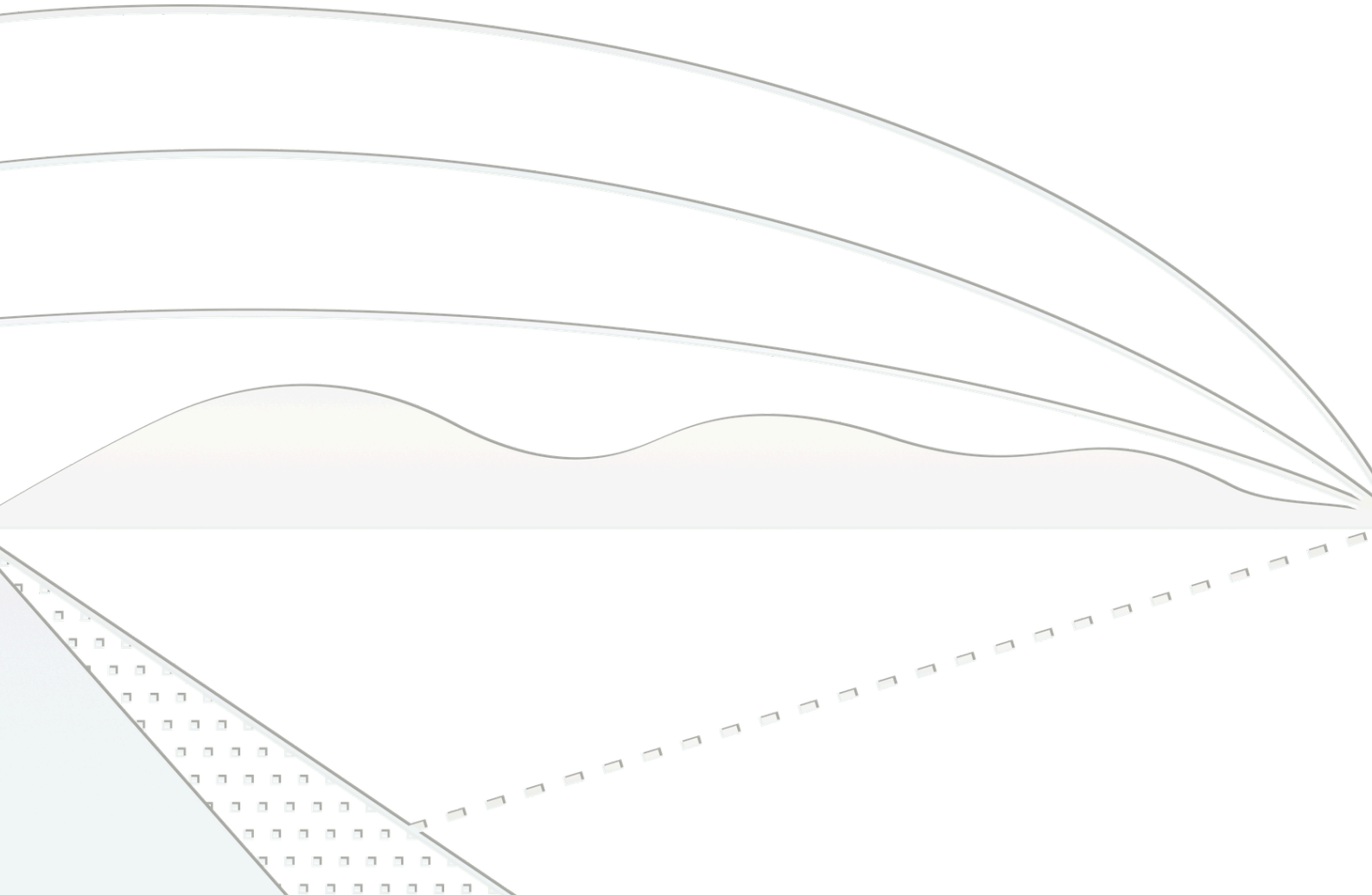
02 Why software never made the jump to engineering

Confidence is good, *proof is better*

Prior to the formal development of engineering, even the most ambitious construction relied on craft knowledge. Once calculus was developed in the 17th century, engineers could model the forces on a structure mathematically and generate evidence that a design would hold. Craft became engineering when an evidence requirement arrived.

Software has never had that moment. The standard approach to software testing can only sample a finite set of executions to check whether they each behave correctly. It can never cover every possible input or state.

In contrast, formal verification mathematically proves a property holds for all possible executions. That's a fundamental difference in approach. No amount of additional testing can turn a sampled result into a proof.



03 Formal Verification

Why formally verify the foundation?

The effort required to formally verify all the software on a device rarely makes economic sense, and most application code changes too often to make it practical. So, which part of the system justifies the cost of a full mathematical proof?

The answer is the kernel and the key pieces that configure it. These fundamental elements run in the processor's privileged mode, arbitrate access to memory and hardware, and mediate every other component's ability to do anything at all. A flaw here isn't contained the way a flaw in an application is, instead it can potentially impact everything that the kernel touches. If the foundation is provably correct, then everything built on top of it benefits from those proven properties, and whole categories of flaws and vulnerabilities are eliminated.

What does "formally verified" actually mean?

At the base of KOS is the seL4 microkernel. This microkernel is stripped to its essential functions, managing memory, running processes, and handling communication between them, implemented in roughly 10,000 lines of code.

A conventional monolithic kernel like Linux runs tens of millions of lines of code in privileged mode. With seL4's much smaller footprint, there is a much smaller attack surface, before proof is even considered.

What makes seL4 different is that its implementation has a machine-checked mathematical proof that it behaves exactly according to a formal specification. In addition, seL4 has proofs that the specification itself provides the isolation properties, confidentiality and integrity, needed to build secure systems on top of it. Access to memory, threads, CPU time, and inter-process communication is governed by fine-grained, capability- based access control. The access-control behavior is also part of what's been proven.

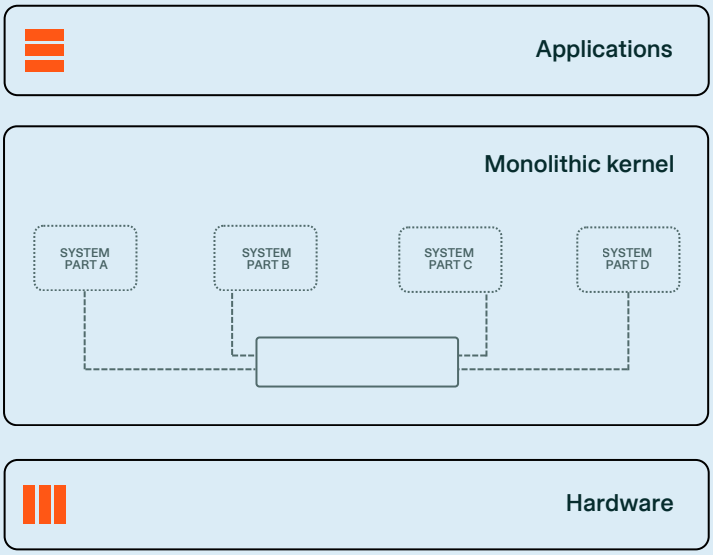
The result is that whole categories of vulnerability, buffer overflows, memory corruption, privilege escalation, undefined behavior, are **ruled out** in the kernel by proof, not **reduced in likelihood** by testing. Whether or not a fuzzer or a red team could find such a flaw in the kernel is irrelevant because the whole category of flaw has been provably eliminated from the code.



Building certainty into an uncertain world

04 Turning proof into an operating system

MONOLITHIC KERNEL ARCHITECTURE

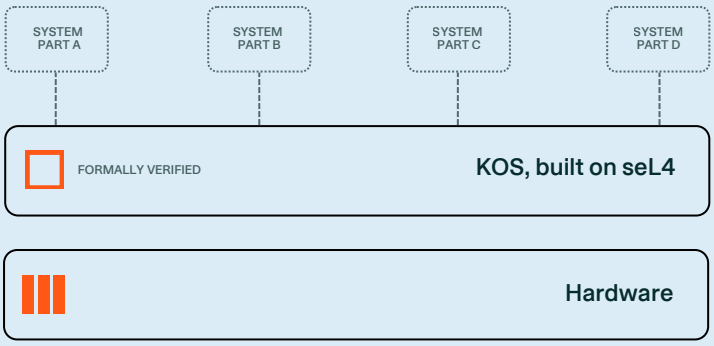


With seL4, most operating-system functionality is forced out of the kernel and into unprivileged, user-level components.

KOS adopts what's called a multi-server design. Each service that would otherwise live inside a monolithic kernel instead runs as its own isolated, address-space-separated component with only the privileges it needs.

By analogy, a conventional operating system, with a monolithic kernel (see diagram), is like an open-plan office. Every part of the operating system's kernel shares the same space and resources. When a threat gets through the 'front door', it can move freely throughout the whole building.

ISOLATED PARTITIONS ON KOS

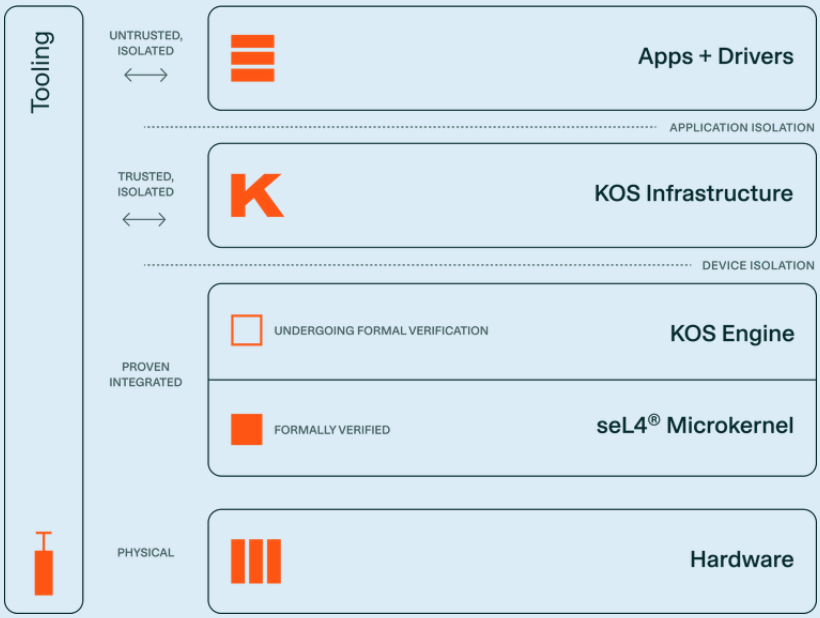


KOS gives every system part its own secured partition, like the cubicles in an office, with access only to what it's explicitly been granted. If one component is compromised or crashes, the breach or failure stays contained inside that partition.

KOS enforces specific configurations, creating a trustworthy SBOM that defines what is **actually** running, rather than a "Software Bill of Hope", that outlines what you **hope** is running. Cyber assessment teams can look at a KOS SBOM and **know** that it is going to work.

05 The benefits of KOS in practice

FULL KOS ARCHITECTURE



- A compromised or buggy driver can't escape its bounds and interfere with the kernel or other components isolated through partitions. In a monolithic kernel, drivers run in privileged kernel space, so that the compromised driver can interfere more broadly.
- Apps are isolated, and can only talk to other pre-approved apps, drivers, or peripherals. Lateral movement isn't a realistic option for attackers due to tightly controlled communication links between components.
- KOS delivers maximum uptime. Any component that crashes can be restarted, according to a configurable policy, rather than taking the whole system down.
- Minimize downtime during upgrades by only changing individual partitions rather than a whole system image, so a change to one component doesn't necessarily reopen certification questions for other parts of the system.

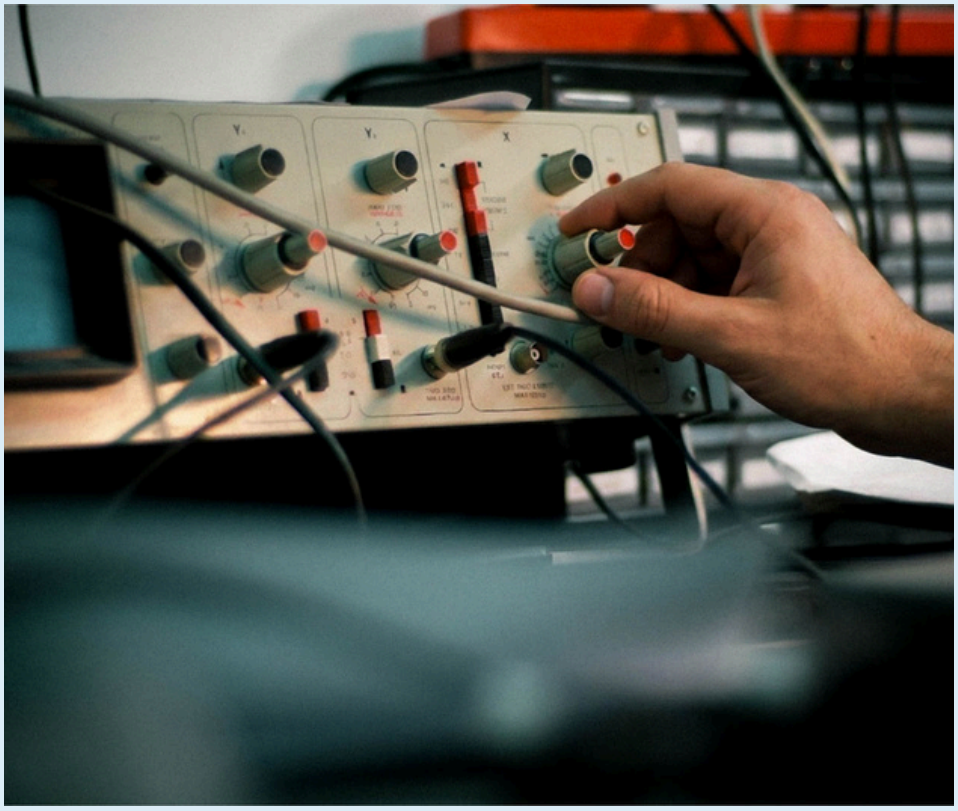
None of this requires the application code itself to be formally verified, it's a property of the foundation underneath it.

06 What’s proven and what isn’t

Formal verification applies to the seL4 microkernel specifically, providing proofs of functional correctness, integrity and confidentiality. Kry10 is currently formally verifying the KOS components that configure your system, run the updates and enforce communication.

Verification stops at the trusted boundary you architect and KOS gives you the means to enforce that boundary. This is not a guarantee that everything inside it is proven.

The same discipline applies to AI workloads. For example, KOS provides a verified sandbox for a model. The model itself is not formally verified, but the container it runs in is.



We're ready to *partner*

Let us help you strengthen what matters

Where to go next

KOS developer documentation is available via an evaluation or SDK license, and a public documentation repository is in its final stages of development.

The most useful next step for a technical reader is a direct conversation with Kry10's engineering team about where your specific system's trust boundary would sit on top of KOS.

KRY10.COM
HELLO@KRY10.COM
USA/UK/AUS/NZ

KRY10