GERNOT HEISER

Your system is secure? Prove it!

Gernot Heiser is professor of operating systems at the University of New South Wales (UNSW) in Sydney and leads the embedded operating-systems group at NICTA, Australia's Centre of Excellence for research in information and communication technology. In 2006 he co-founded the startup company Open Kernel Labs (OK), which is developing and marketing operating-system and virtualization technology for embedded systems, based on the L4 microkernel.

gernot@nicta.com.au

NICTA is funded by the Australian government's Backing Australia's Ability initiative, in part through the Australian Research Council.

COMPUTER SECURITY IS AN OLD problem which has lost none of its relevance—as is evidenced by the annual Security issue of *;login:*. The systems research community has increased its attention to security issues in recent years, as can be seen by an increasing number of security-related papers published in the mainstream systems conferences SOSP, OSDI, and USENIX. However, the focus is primarily on desktop and server systems.

I argued two years ago in this place that security of embedded systems, whether mobile phones, smart cards, or automobiles, is a looming problem of even bigger proportions, yet there does not seem to be a great sense of urgency about it. Although there are embedded operating-system (OS) vendors working on certifying their offerings to some of the highest security standards, those systems do not seem to be aimed at, or even suitable for, mobile wireless devices.

Establishing OS Security

The accepted way to establish system security is through a process called *assurance*. Assurance examines specification, design, implementation, operation, and maintenance of a system.

The most widely used assurance process is the international standard called the *Common Criteria for IT Security Evaluation*, or Common Criteria (CC) for short. CC evaluation is performed against a *protection profile* (PP), which represents a standardized set of security properties the system under evaluation is expected to meet. The idea is that purchasers of IT systems can define their security requirements through a PP (or a combination of PPs) and can then select any system that is certified to match that PP.

CC compliance is evaluated to a particular *evaluation assurance level* (EAL). These range from EAL1, the easiest (requiring little more than a demonstration that the system has undergone some testing), to EAL7, the toughest. The goal of a CC evaluation is to obtain certification from an accredited authority that the system satisfies all the required criteria for a particular PP at a certain EAL. A higher evaluation level means a more thorough examination of the system. This does not, however, guarantee more security; it means only that a more thorough and systematic attempt is made to eliminate vulnerabilities.

A number of operating systems have been certified under CC, including Mac OS to EAL3, versions of Windows, Linux, and Solaris to EAL4, and the hypervisor of IBM's z-Series to EAL5. The Green Hills Integrity microkernel is said to be undergoing evaluation to EAL6.

But what does this mean? At the toughest assurance level, EAL7 (which to my knowledge has not yet been achieved by any OS that provides memory protection), CC evaluation is characterized as "formally verified design and tested." In a nutshell, this means two things:

- The system has an unambiguous specification. At EAL7 this must be in the form of a formal (mathematical) model, and there has to be a formal proof that the specification satisfies the requirements of the PP (e.g., that no unauthorized flow of data is possible in the system).
- There is a correspondence between the mathematical model and the actual implementation of the system. This is established by a combination of means, including a formal high-level design, an at least semiformal low-level design, formal or semiformal correspondence between them, a detailed mapping of design to implementation, and *comprehensive independent testing*.

There is also a requirement that the system under evaluation be "simple." This is a reflection of the security principle of least authority (POLA) and economy of mechanisms, which imply that a system's *trusted computing base* (TCB) should be as small and simple as possible.

Testing Required

CC, even at EAL7, relies on *testing*. Although mathematical proofs are required for security properties of the system's API, there is no proof that these properties hold for the actual implementation. This is why testing is still required. Testing, as Dijkstra famously stated, "can only show the presence, not the absence, of bugs." Hence, even a system certified at EAL7 must be suspected to contain security flaws.

Why does CC not go further and require an actual correctness proof of the implementation? After all, formal proofs for computer programs have been around for decades. Presumably the answer is that it was not considered feasible. Formal code proofs, doable for small algorithms, scale very poorly with code size. Systems that are undergoing CC certification at EAL6 or EAL7 are typically *separation kernels*, very simple OS kernels whose sole purpose is to provide strict (static) partitioning of resources among subsystems. A typical separation kernel consists of maybe 4,000 lines of code (LOC), which may be small as kernels go but is huge as far as formal verification is concerned.

The Next Step

So, are we stuck with trusting the security of our computer systems to traditional debugging approaches such as testing and code inspection, enhanced by model checking (a class of formal methods that may be able to prove the absence of certain categories of bugs but not *all* bugs)?

I think not. One of the most exciting developments in this respect is that it now seems feasible to fully verify the implementation of a complete *microkernel*. A microkernel is a much more powerful construct than a separation kernel, as it is a platform on which a general-purpose OS can be implemented. A well-designed microkernel is a superset of a separation kernel, in that it can provide the same functionality, plus more. However, it is inherently more complex: A minimal microkernel that has sufficient functionality to support high-performance systems of (virtually) arbitrary functionality weighs in at some 7,000–10,000 LOC.

In spite of this, complete formal verification of a microkernel is nearing completion at NICTA. In a project that has been running since January 2004, the API of seL4, the latest member of the L4 microkernel family, has been formalized as a mathematical model in a theorem prover. A number of security properties have been proved about this API, with more to come: The aim is to provide a complete set of proofs corresponding to at least one of the CC PPs. The seL4 kernel can then be used as the basis of systems whose TCB is truly *trustworthy*.

The implementation proof is progressing concurrently with the security proofs of the API. It uses the *refinement* approach, which is a multistep procedure involving intermediate representations (between the specification and the code). Each refinement step proves that the lower-level representation has all the relevant properties of the higher level.

In the case of seL4, there are three levels: The formal specification is the highest, and the actual C and assembler code of the kernel implementation is the lowest. The intermediate level (which roughly corresponds to CC's low-level design) has a concrete meaning, too: It corresponds to a prototype of the kernel implemented in the functional programming language Haskell, which serves as an executable specification for porting and evaluation purposes.

The first refinement step is completed; the second (and final) one is in progress and is due for completion during the second quarter of 2008.

This still leaves a gap: It assumes that the correctness of the implementation is established by showing the correctness of the code (C and assembler). Although CC makes the same assumption, this nevertheless leaves the C compiler and the assembler as trusted components in the loop. Given the quality, size, and complexity of a typical C compiler, this is still an uncomfortable level of trust.

The problem could be solved by performing a third refinement step, from C/assembler to actual machine code. This would require a considerable effort, but it is inherently no more difficult (and most likely easier) than the previous refinement steps. However, there is promising work performed elsewhere on compiler verification. A verified compiler could be leveraged to close the gap without a further refinement step on the kernel.

Let's Get Serious About Security!

Security has far too long been treated with insufficient rigor, given what's at stake. CC, despite best intentions, could actually be counterproductive there. By stopping short of the requirement for formal verification at the highest assurance level, CC has the potential to create a false sense of security. After all, a system certified to EAL7 can rightly be claimed to have passed the highest hurdle of security evaluation. The problem is that this is still incomplete, and a potential for security flaws remains.

If complete formal verification is possible, it must become a requirement.

FURTHER READING

The approach taken in designing and implementing seL4 is described by K. Elphinstone et al., "Kernel Development for High Assurance," *Proceedings of the 11th Workshop on Hot Topics in Operating Systems*, San Diego, May 2007, USENIX.

Further information on seL4 can be found on the project Web site, http://ertos.org/research/sel4/, and the Web site of the verification project, http://ertos.org/research/l4.verified/.

The Common Criteria document is available at http://csrc.nist.gov/.