GENERAL DYNAMICS

Mathematically Verified Software Kernels: Raising the Bar for High Assurance Implementations

Dr Daniel Potts, VP of Engineering, General Dynamics C4 Systems
Rene Bourquin, Member Technical Staff, General Dynamics C4 Systems
Leslie Andresen, Technical Director, General Dynamics C4 Systems

Dr June Andronick, Senior Researcher, NICTA
Dr Gerwin Klein, Senior Principal Researcher, NICTA
Prof Gernot Heiser, Research Group Leader, NICTA

Version 1.0

JULY 2014

Introduction

Recent updates to security guidance documents, such as the US National Institute of Standards and Technology (NIST) Special Publication (SP) 800-53 Rev 4, draw attention to the need for trustworthy *implementations,* in conjunction with the application of associated controls, as part of the development of a comprehensive assured solution. It outlines the importance of addressing not only how you implement, but also what you implement to ensure proper security functionality and security assurance.

Emerging developments in formally verified separation kernels and high assurance architectures significantly strengthen the assurance that can be achieved, and enable wider adoption of this approach. The key tenets of this paradigm are (1) the appearance of more general-purpose yet high-performance separation kernels that support more complex componentized architectures, and (2) the comprehensive formal verification of the *trusted computing base* (TCB). This paper examines specific examples of these techniques, and discusses their foundational applicability to improve system security and resilience.

First, we are building real-world, software-based, componentized, high assurance mobile and critical embedded systems, based on third-generation separation kernels, such as the OKL4 Microvisor and the seL4 microkernel. Both have realized the art of the possible and are now being used to build commercial and defense systems with more compelling, stronger security, safety and reliability postures. For example, the Secure Mathematically-Assured Composition of Control Models (SMACCM) project, funded by DARPA, builds upon the seL4 kernel to demonstrate a complete high-assurance system of significant complexity, ultimately a military UAV.

Second, we have demonstrated the comprehensive formal verification of the seL4 microkernel, with a complete proof chain from precise, formal statements of high-level security and safety properties to the binary executable code.

Finally, we address affordability, which has previously limited the broad adoption of formal methods to 'only as absolutely necessary' applications. Today, the cost of producing formally verified, high-assurance systems is on the order of $200-400/LoC (Lines of Code) for critical code. This cost is much lower than using other traditional (and weaker) techniques to establish high assurance, allowing industry to 'have its cake and eat it too' – affordability is no longer a barrier to security. We discuss this further in Section 3.0.

In summary, we assert that the use of formal methods for software verification, coupled with high assurance architectures, is now the state of the art, and therefore should 'set the bar' for assured solutions. This approach is ready to be applied, in support of emerging security guidance documents, for use in a broad and diverse class of applications, from control systems of drones, to multi-domain commercial-off-the-shelf (COTS) mobile devices.

About the Authors – Our Collaboration

General Dynamics C4 Systems, including its recent acquisition of Open Kernel Labs, and NICTA are long-term, committed partners in the development of trusted systems that offer a greater level of assurance, richness, and functionality than has been available in the past. Through cutting-edge research and architectural development, we evolve it, mature it, make it simple, and ultimately deliver it in products. Our earlier research in separation and micro-kernels has now been deployed in over 2 billion mobile and embedded devices.

We plan to do the same with seL4. By leveraging our complementary IP and key strengths in state-of-the-art systems research, development, and productization of emerging technology, we are committed to

adopting these technologies in our own products, while enabling others to do the same through industry and Government engagement.

1.0 High Assurance Architecture: "Build security in"

In this section, we advocate that high-assurance software-based systems should be built on a componentized architecture, with a minimal *trusted computing base*, and with isolation (or controlled communication) provided by an underlying general-purpose separation kernel.

The architecture has the following characteristics:

1. At its core, a microkernel that minimizes the amount of code with privileged access to the hardware, and that is formally proven to possess precisely defined isolation properties.
2. A minimal set of application-level components that make up the rest of the TCB. Leveraging the isolation properties of the microkernel, these components are formally verified to behave as specified.
3. Precisely defined information and resource access, enforced by the microkernel, ensuring that one fault in the system does not compromise the whole system.
4. The remaining (largest) part of the system being untrusted, contributing to utility of the system without being able to interfere with critical operations..

This implies a drastically shrunk security boundary, small enough to prove it correct, and the ability to maximize the use of untrusted (e.g., COTS and open-source) components for greater overall flexibility, affordability and utility, without compromising security.

These combine to allow the implementation of high-assurance architectures providing significantly improved system resilience, while also reducing costs associated with the need to certify every piece of the entire system with the same scrutiny.

An example of the architecture is given in Figure 1, showing how the General Dynamics OKL4 Microvisor has been successfully applied towards the development of multi-domain and multi-persona mobile devices. The OKL4 Microvisor kernel provides the foundation for isolated and separate componentization. Device drivers and critical encryption functionality, including network VPN and data-at-rest (DAR) flash encryption, is removed from the Android personas. This provides the system with an increased level of defense-in-depth against attack on critical components.

The encryption routines also become non-by-passable from within the Android personas. That is, an exploit or malicious user cannot bypass or disable the network accessible only via the VPN component, nor can it disable the data-at-rest (DAR) component providing full disk encryption. Furthermore, the attack service for these components becomes quite small and easy to validate for correctness and resilience against attacks.

Finally, the OKL4 Microvisor supports access and information-flow controls, enabling the system to implement different policies defined at deployment or run time.

Similar systems are being built on top of seL4 (including the SMACCM project mentioned above, and cross-domain solutions). In these systems, information flow policy enforcement will be formally proved.

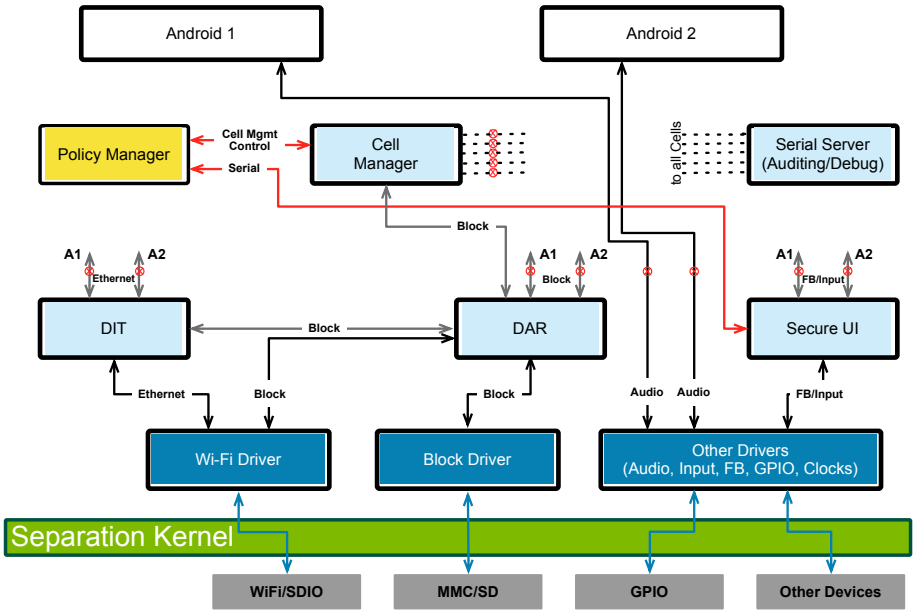

Figure 1: An example multi-persona Android platform built leveraging a separation kernel (OKL4 Microvisor) for componentization. This approach enables stronger policy controls, non-by-passable guaranteed invocation of encryption, and other high-assurance capabilities.

2.0 Formal verification: "Prove that you've built security in"

High-assurance systems need strong, measurable verification of reliability, safety and security. Certification for use requires evidence that these requirements are met. Traditionally, the best achievable evidence that a critical system behaves correctly and enforces some desired security or safety property derives from a combination of testing, code inspection, engineering processes, and formal modeling, augmented by partial verification. This approach is expensive, yet imperfect.

The use of Formal Methods is recognized as the strongest guarantee that can be provided about software behavior. For instance, the highest levels of certification under the Common Criteria security certification scheme demand formal models of the system and formal proofs relating the models to security requirements [1]. In the avionic space, the latest certification scheme DO-178C introduces a supplement providing guidelines on how formal methods should be used to complement testing [2].

Until recently, however, there was a lack of evidence that a comprehensive use of Formal Methods was feasible, effective and affordable. Furthermore, an overloaded and overly broad use of the notion of “formal verification” created confusion and undermined confidence in the approach. This has slowed adoption, and created a reluctance of certification bodies to impose more stringent requirements on the use of Formal Methods.

Here we show that *formal verification, of strong properties, down to the machine code, of real-world systems, is feasible at reasonable cost*: seL4 is a real-world, general-purpose operating system, with a guarantee that its machine code is correct with respect to its specification and that it enforces critical security properties such as information flow control, integrity and confidentiality. Moreover, we establish that an assurance level beyond the highest existing security certification standards can be achieved at a cost that is lower than with traditional approaches [3]. We therefore claim that nothing less than comprehensive formal verification should be demanded for the highest level of assurance for critical systems.

We first comment on present limitations and inhibitors to more widespread adoption of Formal Methods, and show how to overcome them.

Testing and inspection can never exhaustively cover all behaviors and inputs of a non-trivial system, and thus cannot show the absence of defects or vulnerabilities. In contrast, Formal Methods build on a mathematical model of the system, on unambiguous and precise statements of the desired properties, and on mathematical proof that the properties are satisfied by the model. Reasoning about all possible behaviors of a system, and proving correctness and absence of defects, is then possible.

All formal verifications are based on assumptions, which scope what is formalized and verified:

(a) The assumption that the model is a faithful representation of the system: how much does a guarantee about the model say about the system itself?

(b) The assumption that the formal statement of the properties faithfully represents the desired behavior of the system. The properties range from deep statements about the reliability, security and safety of the system (such as the typical certification requirements of functional correctness, confidentiality, integrity, etc), to shallow properties concentrating on checking for patterns, such as absence of runtime errors or compliance to coding conventions.

(c) The assumption about the soundness of the tool/framework used: how much can we trust the tool used to produce the proof?

The term "verified" has been used for a wide range of approaches of varying strength, often without explicit statement of the limitations and assumptions. This ranges from using an unsound tool to check one specific property about a very abstract model of a system, to using a sound tool to proof full functional correctness of the binary code of the system.

Recommendation 1: *Every claim of a verified system should state precisely what has been verified, and explicitly state all limitations and assumptions under which the guarantee holds.*

It seems clear that the highest assurance levels should require the strongest certification, i.e., specifically using formal mathematical proof. To date this is not mandated, as it was, until recently, considered infeasible. Here we describe how the latest progress in formal tools and techniques led to a demonstration of feasibility of providing strong, measurable guarantees for real-world systems.

Regarding faithfulness of the formal model, Assumption (a) above, the closer the model is to the machine code that is executed on the running system, the stronger the guarantees can be that are derived from the model. Latest research provides formally verified compilers [4] and independent, automated verification of the correctness of the compiler-generated code [5]. The latter was used to prove that seL4 machine code is a correct refinement of its high-level, formal specification, reducing the assumptions to the correctness of the model of the hardware. In contrast, Common Criteria's highest evaluation level (EAL7) requires formal refinement proof down to low-level design model only, with informal links between the model and the source code, and no requirement to verify compilation correctness.

Recommendation 2: *Critical systems should be required to provide guarantees down to the binary level*.

There is a big gap between this low-level correctness requirement and assumption (b) above, which asks for strong guarantees of deep, yet high-level properties, whose formal statement should correctly express the intuitive desired behavior. Assumption (b) calls for (i) an expressive high-level language to denote the desired property clearly and unambiguously, and (ii) verification to be performed at an abstract level, where reasoning is practical and scalable.

To reconcile the need for abstraction with the need for machine-level guarantees, we require a formal proof of full functional correctness of the source and machine code against a high-level abstract specification, as was done for seL4 [3], which then allows carrying out verification of high-level properties, such as integrity [6] and confidentiality [7], at the abstract specification level. Full functional correctness means that the abstract specification is a complete and precise description of all the possible functional behaviors of the system, and the refinement proof shows that the behavior of the binary implementation is fully captured by the abstract specification. This is a strong property in itself, and additionally allows proving any further properties that are preserved by refinement on the significantly simpler abstract specification, while still getting the guarantee down to the binary level. Proof of further high-level properties is also a way to increase the confidence in that the specification represents the intuitive expected behavior.

Assumption (c), about the trust in the verification tool, calls for use of sound or foundational tools. For instance, the seL4 verification work used the Isabelle theorem prover, where soundness-critical code is concentrated in a relatively small proof kernel, and statements are then defined and proved from first principles (not axiomatized). Additionally it can produce external proof representations that can be independently checked by a small, simple proof checker [8]. More generally, certification schemes usually require evidences of soundness of the formal methods techniques and tools used to produce proof artifacts.

3.0 Keep it Affordable

So far, we have provided arguments supporting the feasibility of formal verification of strong properties at machine level of real-world systems. We now have to show that cost does not have to be an obstacle. The most compelling evidence lies in the costs associated to seL4 verification [3]: $362 per line of code for the correctness proof plus $78 per line for the security proofs. This is a very low figure for the strongest assurance ever produced about the security of a general-purpose OS kernel. It is, by comparison, much lower than the industry rule-of-thumb of $1,000 per line of code for designing, implementing and certifying a system under Common Criteria EAL6, where only semi-formal model of the design are required, with informal mapping to the code, leading to much weaker guarantees than seL4 proofs.

It should be noted that the above cost included a significant amount of learning and development of new techniques; as such, we estimate that a similar verification effort could now be done at half the cost, or around $200/LOC. This is then comparable to the cost of producing low-assurance software of a similar nature [3].

In addition, formal verification offers auxiliary benefits in terms of more focused testing. As formal verification requires making all assumptions explicit, testing is only required to establish that the assumptions are satisfied. The recognition of the complementary nature of testing and formal verification is slowly starting to make its way into certification schemes (e.g., in DO-178C), but more needs to be done to avoid duplication of effort, where formal proofs make some testing redundant. If incorporated into

the development process, with iteration between design, verification and coding, this approach also leads to early design validation and bug detection [9], leading to further cost reduction opportunities.

Another advantage of formal verification at code level is in terms of system maintenance and evolution: re-running the proof on a variant of the code gives immediate feedback if the changes break any of the previously established security guarantees, in stark contrast to current approaches, which require expensive (informal) re-validation after every change. This should be exploited in reducing effort and costs in re-certification of variants of certified systems.

This is further supported by previous efforts such as those of the Trusted Computer System Evaluation Criteria (TCSEC) [10], also known as the Orange Book. The Rating Maintenance Phase (RAMP) process from the TCSEC would work today as it focuses on the changes made and how they would affect the overall system. More so, it specifies that at A1 level, formal verification was required. As we contend, such a process is compelling as it reduces the burden and effort to develop and maintain a formally verified system.

Finally, with the componentized architecture discussed in the previous section, only the Trusted Computing Base needs to be verified. If "built right", this allows for large (million-line) systems to be developed - by investing only in the verification of a microkernel and few trusted components, in total comprising a few tens of thousands of lines.

To illustrate further the benefit of our approach, we show how it would apply to the high assurance architecture presented in the previous section. We have been advocating that high assurance systems should be built on a componentized architecture, with a minimal TCB and with isolation (or controlled communication) provided by an underlying kernel. To prove that this approach produces a system with the desired properties, we need to:

(i) Prove that the system is set up correctly: Prove that the components are created according to the high-assurance architecture and that the authority they are provided with is conformant to the architecture specification (see for instance [11]).

(ii) Prove that the TCB is trustworthy: Prove that even if parts of the system are compromised, the kernel and all the components identified as "trusted" in the architecture are not violating the desired property. This requires verification of the (small) TCB, including functional correctness of the kernel. For some TCB component, proving weaker properties than full functional correctness may be sufficient.

(iii) Prove that untrusted components can be ignored: prove that whatever their behavior is, the authority provided to them at set-up indeed does not allow them to violate the desired property. This simply requires proving that the desired property holds under *no* assumption about their behavior (in other words, such components are assumed to be malicious). The only assumption made is that they cannot perform operations not allowed by their authority. For this we need to provide a proof that their authority is enforced by the kernel [7] [6].

The purpose of the formal proofs is to validate the careful design of the architecture, where the architecture in turns aims at reducing the cost of such verification.

Conclusion

The state-of-the-art in formal verification shows that it is now feasible and cost-effective to demand strong, code/machine level guarantees as table stakes for highly critical systems. Leveraging separation kernels to create high assurance architectures with formally verified software components, as outlined in this paper, provides a means to implement such systems with measurable security enhancements.

Bibliography

[1] US National Institute of Standards, *Common Criteria for IT Security Evaluation*., 1999.

[2] Radio Technical Commission for Aeronautics (RTCA), *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*., Dec 2011.

[3] Gerwin Klein et al., "Comprehensive Formal Verification of an OS Microkernel," *ACM Transactions on Computer Systems*, vol. 32, no. 1, pp. 2:1-2:70, Feb 2014.

[4] Xavier Leroy, "Formal verification of a realistic compiler," *Communications of the ACM*, vol. 52, no. 7, pp. 107-115, 2009.

[5] Thomas Sewell, Magnus Myreen, and Gerwin Klein, "Translation Validation for a Verified OS Kernel," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Seattle, Washington, USA, Jun 2013, pp. 471-481.

[6] Thomas Sewell et al., "seL4 Enforces Integrity," in *International Conference on Interactive Theorem Proving*, vol. 6898, Nijmegen, The Netherlands, Aug 2011, pp. 325-340.

[7] Toby Murray et al., "seL4: from General Purpose to a Proof of Information Flow Enforcement," in *IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2013, pp. 415-429.

[8] Stefan Berghofer, "Proofs, Programs and Executable Specifications in Higher Order Logic," Institut für Informatik, Technische Universität München, Ph.D. dissertation 2003.

[9] Mark Staples et al., "Formal Specifications Better Than Function Points for Code Sizing," in *International Conference on Software Engineering*, San Francisco, USA, May 2013, pp. 1257-1260.

[10] US Department of Defense, *Trusted Computer System Evaluation Criteria*., Dec 1985.

[11] Andrew Boyton et al., "Formally Verified System Initialisation," in *International Conference on Formal Engineering Methods*, Queenstown, New Zealand, Oct 2013, pp. 70-85.