Cybersecurity Skeptics Now Embracing Formal Methods

An Interview with Gernot Heiser and Jim Morris

by Ted G. Lewis

Editor's Introduction

There is new hope for those who despair securing computer systems from external hackers. The recent DARPA HACMS project demonstrated conclusively that “certain pathways for attackers have all been shut down in a way that’s mathematically proven to be unhackable for those pathways.” Continuing research at DARPA and IARPA will eventually shut down all the pathways, and the external hackers will be out of business permanently.

Cybersecurity Skeptics Now Embracing Formal Methods

An Interview with Gernot Heiser and Jim Morris

by Ted G. Lewis

It has perhaps been the majority opinion among most security experts in the past few years that:

- computer systems cannot be made secure from either external hackers or malicious insiders;
- formal methods are useless in helping to provide security for computer systems.

Recent research at DARPA has shed new light on computer security and formal methods. The main point of this interview is to provide evidence that these two negative opinions no longer have credibility. Recent results from the DARPA HACMS program have shown computer systems can be made secure from hackers, and it was largely formal methods that made these new results possible.

Ted Lewis: Hacking by outsiders has been getting worse for years, and now theft by malicious insiders, such as reportedly occurred in the recent CIA hack, seems to be getting worse. Why can't we fix our systems to prevent this hacking?

Jim Morris: We need to build a new generation of simpler, trustworthy systems if we expect to end the data theft and data damage by both external hackers and malicious insiders. Several computer security experts have said as much in the last year or two. For example, Kevin Fu, in a recent Ubiquity [blog](#), said, "... security must be built into IoT devices, rather than bolted on after deployment." David Talbot, in an *MIT Technology Review* article entitled "[Why We're So Vulnerable](#)," wrote, "... the next generation of technology must have security built in from the very start." And Ron Ross, the top cyber-scientist and a Fellow at NIST, [shared](#) the following: "Security does not happen by accident. Things like safety and reliability need to be engineered in from the beginning."

TL: What is a trustworthy system?

Gernot Heiser: A trustworthy system is one that can be shown to live up to its users' expectations on security, safety, reliability and efficiency. Present systems are not trustworthy, as evidenced by all those security and safety failures we hear about all the time. True trustworthiness can only be achieved through appropriate, safety/security-oriented systems design, combined with the application of cutting-edge formal methods, i.e. mathematical proof techniques.

TL: What would be your approach to building a trustworthy system?

GH: Core to design for trustworthiness is a minimal trusted computing base (TCB), i.e. a design that clearly identifies, minimizes and isolates the critical parts of the system, whose failure lead to overall system failure. This approach requires a trustworthy operating system (OS), whose core duty is to provide strong isolation for the critical components, protecting them from failure in non-TCB components. This then allows one to reason about overall security/safety by only examining the TCB components and their interaction, thus making the process of establishing overall system trustworthiness tractable.

TL: What trustworthy operating system would provide the TCB and the isolation properties you would need?

GH: Our approach uses our [seL4 microkernel](#) as the OS that guarantees isolation. seL4 has undergone extensive formal verification, including proof that a mathematical model of the kernel enforces isolation, proof that this model is correctly implemented in C code, and proof that the C code is correctly translated into machine binary code. We are working on proving similarly strong temporal isolation.

TL: You have stated systems need to be much simpler if we are going to make them secure. But computer systems are complex by their very nature. How are you going to build systems that are less complex, but still have the functionality of our current systems?

JM: We need to be more disciplined in how we develop systems for use in critical cyber-physical systems in much the same way we design, verify, and construct bridges. I believe the best way to structure these systems, to make them simpler, is as microservices architecture, and we need to formally verify the correctness and security properties of the systems as much

as we can. The overly complex systems in use today running Windows and *nix are, in the vast majority of cases, structured as a single, giant monolithic architecture. A monolithic architecture is, by its very nature, extremely complex, difficult to maintain and deploy, and impossible to verify formally.

TL: I have heard of microservices, how do they map onto the TCB components Gernot mentioned?

GH: With a microkernel, the natural approach is to structure a system as a collection of strongly encapsulated components, which interact via narrow, kernel-enforced interfaces. At the lowest level, each such component is an address space, which makes them more coarse-grained than the object notion from OO programming, but the concepts are similar. Furthermore, components can be grouped to present a higher-level component, whose API is a small subset of the union of the individual component APIs. This is nothing but applying the classical engineering approach of modularization and abstraction to software systems. A microservice is then a collection of cooperating components, which, together, provide some high-level functionality.

Each microservice has its own notion of a TCB, and if you want to make that microservice trustworthy, you'll need to make the TCB components trustworthy, ideally by proving them correct. Of course, the TCBs of different microservices will overlap, i.e. they share critical components—the critical system services. This sharing is safe if the system components are formally verified.

What makes this fly is components can be verified in isolation, without reference to other components, except the services they explicitly import, but even those services can be represented by their specified functionality (as long as they are provided by trustworthy components), thanks to the isolation and API enforcement provided by seL4. By keeping components small and simple, the verification effort can be kept manageable.

TL: Given the inherent cost of crossing interfaces, how would you break a monolithic system up into components and still have it perform well?

JM: This is an ongoing debate that will not be decided soon. The debate is about granularity. A monolithic system is maximally coarse-grained (which can be thought of as a single "macroservice"). A microservices architecture with 1 million microservices is extremely fined-

grained (ridiculously so, as a matter of fact, but you get the point). The trade-off related to granularity is performance versus all the good features you get from microservices architecture: maintainability, relative simplicity, development team size, and the ability to do continuous deployment. A monolithic system will always give the best performance, but at the expense of all the good features that come with a more fine-grained microservices architecture. On the other hand, if the architecture is too fine-grained, most of the time is spent in inter-microservice communication, and performance suffers. Also, if the number of microservices grows too large, the complexity of the architecture itself becomes a major problem.

TL: Has anyone ever developed practical componentized systems that use a microkernel as the core of its trusted computing base?

GH: There are presently two supported component frameworks for seL4, and both are used in real-world systems. One is the Genode system (from the eponymous company in Dresden, Germany) that supports a number of microkernels, including seL4. It is fully general and dynamic, and has many usable components. In fact, it is now a complete OS. The Genode developers run it on their laptops (although at present on the NOVA hypervisor rather than seL4). I've seen it at work, and it's as snappy as you can expect, so there are no visible performance issues (and seL4 is actually faster than NOVA). Genode's main drawback is it has no real assurance story, and is unlikely to get one in the foreseeable future (partially because it's all done in C++, heavily templated, etc.).

The other one is CAMkES, which is what we've developed and are using, among others, in the DARPA HACMS program. Its main drawback is that it's, at present, strictly static. In the sense that the system architecture, as a collection of components, is defined at build time and cannot change, although we do have ways of restarting components. The step to a truly dynamic system is much larger, though. Which means it's presently mostly useful for embedded systems, not general-purpose computing, but is enough for most security/safety-critical systems.

The main advantage of CAMkES, and the reason we used it in the HACMS project, is it has an assurance story. It's not yet as strong as seL4, but aspects of the C code generated by CAMkES are partially verified. Some of the generation of "glue" between components is verified, and the security-critical system initialization, which configures all access rights, is verified at the model level.

TL: You mentioned HACMS. What is HACMS? What did it achieve, and how does it relate to formal methods?

GH: HACMS (high-assurance cyber-military systems) is a 4.5-year DARPA program that will finish up in 2017. The goal of HACMS is to use formal methods to build real-world critical systems that are highly resilient against cyber attacks.

One of the interesting things about HACMS is most people expected it to be a flop; most in the community seemed to have given up on trying to make systems secure, and on formal methods ever doing anything really useful. HACMS apparently changed a lot of attitudes in the defense and intelligence community, especially when a real-world system (the ULB, Boeing's optionally-piloted helicopter) first flew with seL4 on its mission computer 18 months ago—a great demonstration that formal methods, and formally verified systems, are suitable for real-world use.

Even more importantly, we demonstrated we could retrofit security into existing systems by re-architecting the software, but keeping the majority of the legacy code. An approach dubbed “seismic security retrofit” by DARPA I2O Director John Launchbury. By extracting critical code into separate, isolated components we protected it from the (untrustworthy) other code. We did this on the Boeing ULB as well as autonomous U.S. Army trucks. The article [“How DARPA Is Prepping For The Next Cyberwar”](#) shows we were confident enough to give DARPA's “Red Team” administrator access on a virtual machine running Linux on the same processor as the critical components, and *they could not break out or compromise the critical parts of the ULB's mission software*.

TL: Was all the critical code in HACMS verified?

GH: No, far from it. Some newly written code was verified, some code had certain properties verified through language techniques or model checking, some code generated from high-level specs, some was traditionally (and carefully) engineered code which we protect through isolation. In particular, there was no overall proof of the system's security or safety properties, so there is still much work to be done. Nevertheless, HACMS demonstrated just by building on trustworthy foundations, using a security-oriented design, combined with other validation techniques, overall security can be improved dramatically.

TL: To complete this picture, you'd have to verify much more code than you have done so far. Will the cost of this not be prohibitive?

GH: Experience with seL4 shows that verified code was about a factor of three less expensive than traditional “high-assurance” code (which gives you zero guarantees), but about a factor of three more expensive than industry-standard, no-assurance code. So it’s cost competitive where people really care, but not yet competitive where people aren’t willing to pay for security. We think high-level languages and generation or synthesis from high-level specifications are the way to bridge that gap.

Generating code is less error-prone than writing it manually, but it’s most powerful if the generation is proved to be correct. An example is CompCert, the “certifying” C compiler from INRIA, which does provably correct translation of C into machine code. Another example is CakeML, which is a functional programming language that has a verified compiler, as well as a verified runtime.

If you have provably correct code generation from a high-level programming language with a simple, well-defined semantics and strong typing that guarantees memory safety (among other things), then you can verify your code in the high-level language semantics. There you can focus on the logic of your algorithms, rather than worrying about all the low-level detail we had to deal with in the verification of seL4.

We are doing that with our Cogent language. Cogent is a type- and memory-safe, non-managed systems language (think Rust), from which we compile into C, together with a proof that the C is correct against the Cogent spec. Our file-system case study with Cogent indicates that we’ve already at least half bridged the cost gap.

Another development in the team is the micro-VM, a language-agnostic kernel of a language VM that is designed to be verifiable. It will let you program in a Java-like high-level, type- and memory-safe managed language and verify in the semantics of that language, again with vastly increased productivity, and get a proof that goes to the binary. But that’s still a few years in the future.

TL: Of course you know that deciding whether or not a program is correct/secure is undecidable. Are we sniffing up the same trail that led Dijkstra to a dead-end?

GH: Yes, security is undecidable in general, but that doesn’t mean you can’t prove anything. Similarly, the halting problem is undecidable in general, but that doesn’t mean that there aren’t classes of programs where you can prove termination (seL4 is one of them). It should be provable that a CAMkES system has certain overall security properties, or rather that a suitable statement of security is maintained for the system, but we haven’t done that yet. Things we

have proved in the past include that a suitably set-up seL4-based system will enforce isolation between partitions (other than through timing channels).

TL: The HACMS deliverable was a rather small embedded system. How do you propose to deal with complexity in a much larger general-purpose system?

GH: An autonomous helicopter or truck isn't really a "small system", but the architecture of the HACMS vehicles was relatively simple. However, the idea of microservices can help to make overall complexity manageable. If the overall system architecture consists of largely independent microservices, then the isolation guarantees provided by the underlying seL4-based OS allow you to treat them mostly in isolation, dramatically reducing the difficulty of the task of getting each one correct. If each microservice is itself well architected, with a minimal TCB (as discussed above), then formal verification of the critical parts becomes tractable. It's basically the time-honored divide-and-conquer approach.

TL: How about multi-factor biometrics in a trustworthy system? That is already happening today in smartphones.

JM: Yes, multi-factor biometrics is happening on smartphones. That's a good thing, because it is an existence proof that fingerprint recognition, iris scans, and facial recognition are viable technologies that are ready to be adopted in trustworthy systems. It should be obvious by now that passwords alone are never going to provide secure authentication. We need to authenticate humans, not passwords. In a system that authenticates humans, you will be authenticated when you attempt to use a secure system. This authentication might occur as a result of an iris scan or facial recognition or fingerprints, or, more likely, a combination of at least two of these factors. This is called *multi-factor, biometric authentication*, and it will be very important in future trustworthy systems.

TL: I am skeptical that we will rewrite everything.

JM: I agree. Here are some examples of the kind of systems that I think will be required to be trustworthy in the future—although it could be 10–20 years before trustworthy systems are pervasive in the systems—listed below:

- All systems that store "sensitive" data
- Most IoT devices (dishwashers, refrigerators, toasters, water pumps, etc.)

- Military and intelligence systems
- Financial systems
- Important infrastructure systems (e.g. the power grid)
- Universal blockchain maintainer.

In conclusion, recall I said at the beginning of this interview that the negative opinions about computer security and formal methods have now been thoroughly disproved by DARPA, Gernot's team, and the other HACMS partners. For more about how the "Red Team" hackers failed to penetrate the ULB (Unmanned Little Bird) systems, see *Quanta Magazine's* much discussed piece, "[Hacker-Proof Code Confirmed.](#)"

I want to share an excerpt from this article (my emphasis is in boldface):

*When the [HACMS] project started, a "Red Team" of hackers could have taken over the helicopter almost as easily as it could break into your home Wi-Fi. But in the intervening months, engineers [Gernot, his team, and their other DARPA partners] from the Defense Advanced Research Projects Agency (DARPA) had implemented a new kind of security mechanism—a software system that couldn't be commandeered. Key parts of Little Bird's computer system were unhackable with existing technology, its code as trustworthy as a mathematical proof. **Even though the Red Team was given six weeks with the drone and more access to its computing network than genuine bad actors could ever expect to attain, they failed to crack Little Bird's defenses.***

*"They were not able to break out and disrupt the operation in any way," said Kathleen Fisher, a professor of computer science at Tufts University and the founding program manager of the High-Assurance Cyber Military Systems (HACMS) project. **"That result made all of DARPA stand up and say, oh my goodness, we can actually use this technology in systems we care about."***

Biographies

Gernot Heiser is Scientia Professor and John Lions Chair of Operating Systems at UNSW Sydney and Chief Research Scientist at Data61, CSIRO. He has a 20-year track record of building high-performance operating-system microkernels as a minimal basis for trustworthy systems. He is the founder and past leader of Data61's Trustworthy Systems group, which pioneered large-scale formal verification of systems code, specifically the design, implementation and verification of the seL4 microkernel. His former company Open Kernel Labs, acquired by General Dynamics in 2012, marketed the OKL4 microkernel, which shipped on billions of mobile wireless chips and more recently ships on the secure enclave processor of all iOS devices. Heiser is a Fellow of the ACM, the IEEE and the Australian Academy of Technology and Engineering (ATSE).

Jim Morris is a software developer, computer systems architect, businessman, and serial entrepreneur with over 40 years of experience, most recently at FullSecurity Corporation researching solutions for microservices architectures and secure data-storage systems that are immune to external infection by malware and resistant to theft by malicious insiders. He has started and sold a successful technology company, did very early research (late 1970s) in OO programming languages for ten years at the Los Alamos National Laboratory, and was an associate professor of computer science at Purdue. His areas of interest are cryptography, software development, operating systems, and hacking methodology. He has a Ph.D. in computer science and a B.S. in electrical engineering, both from the University of Texas at Austin.

DOI: 10.1145/3081880