



VIRTUAL ROUNDTABLE

What Happened to Formal Methods for Security?

Kim Schaffer and Jeffrey Voas, National Institute of Standards and Technology

A panel of seven experts discusses the state of the practice of formal methods (FM) in software development, with a focus on FM's relevance to security.

In a 1996 article, formal methods (FM) advocate Tony Hoare asked, “How Did Software Get So Reliable without Proof?”¹ Twenty years later, in the same vein, we wondered: How did software get so insecure with proof? Given daily media accounts of new malware, data breaches, and privacy loss, is FM still relevant to security—or was it ever?

To explore whether the application of FM is as suitable for today's “build it, hack it, patch it” mindset as it has been for safety-critical system design, we posed seven questions to a panel of seven experts: Paul E. Black of the National Institute of Standards and Technology (NIST); Connie Heitmeyer

of the US Naval Research Laboratory (NRL); Joseph Kiniry of Galois, Inc.; Karl Levitt of the University of California, Davis; John McLean of NRL; Eugene Spafford of Purdue University; and ICT executive Joseph Williams. See the “Roundtable Panelists” sidebar for more information about the panel members. Their unique personal insights are presented below.

.....

FM Suitability for Security

Computer: FM has been highly successful in safety-critical systems over recent decades. It's claimed that much of that success stems from such systems

being deployed in regulated industries. If you agree with this claim, it begs two questions: is FM well suited to security concerns, and is assurance primarily the result of compliance and self-governance?

Joseph Williams: FM has indeed been successfully applied to safety-critical systems. One reason is the overwhelming evidence that it results in safer systems. The application of FM also satisfies a legal burden of proof regarding due diligence. It's likely that as researchers endeavor to squeeze more accuracy out of their analytical models—for example, for pricing or voice translation—FM will find a niche, but such instances will inevitably be market driven.

Eugene Spafford: Failures in safety-critical systems are costly, so investment in better design and use of tools, including FM, is economically justified. Currently, security issues in most non-safety-critical systems, except some highly regulated ones, don't have the same economic pressures and



constraints. Most consumer software development still stresses time to market rather than quality. The burgeoning Internet of Things is moving even further in this direction.

Market pressure is thus unlikely to have much effect. Regulation and, eventually, higher insurance costs could make a difference. However, putting a monetary figure on security and privacy breaches is difficult, and most losses to date seem to be temporary—few or no significant companies have gone out of business because of poor security.

Paul E. Black: Yes, FM is well suited to security concerns. Modeling and complete evaluation can discover problems that testing or human review wouldn't.

FM will only answer posed questions, and models are limited reflections of reality. Hence Donald Knuth's warning, despite the application of FM, to beware of bugs in code: "I have only proved it correct, not tried it." Nevertheless, methods based on logic and mathematics can greatly increase the security of most software.

FM aims to do the job correctly in the first place by reducing the need for extensive testing and uncertainty in the test schedule. Software with significantly fewer bugs also helps preserve the developer's reputation and minimizes the costs of recalls or patches.

John McLean: Security certainly has a large compliance component: verification can't prevent a hacker from gaining access to a system if users choose easily guessed passwords. However, time-consuming compliance requirements, such as keeping a system's patches up to date, stem from implementation bugs that could have been prevented by formal methods. FM can detect programming mistakes leading to security vulnerabilities, such as buffer overflows, as well as subtle errors that permit side- or covert-channel attacks.

Ironically, what makes FM—or any sort of verification—difficult makes

formal analysis so important: security, unlike functional correctness, isn't always easy to specify. That's why some of the earliest applications of FM to computer security focused on models—formal security specifications and analyses—rather than code verification.

Joseph Kiniry: While a small number of regulated industries have "forced" the application of FM, I believe that the vast majority of FM work is unrelated to aircraft and driverless trains. In fact, virtually all of our work is for customers outside of these industries.

FM is extremely well suited to security concerns. Galois's 50-odd technologists spend most of their time solving our customers' R&D challenges, and many of those solutions result in tools that our customers use henceforth to solve their correctness and security challenges rather than pointwise solutions to their immediate problems.

Traditional compliance and process-centric evaluations are a good means by which to ensure system correctness or security. Organizational use of an ISO 9001-certified process has little to no weight with our customers. What constitutes evidence of a system's correctness and security is a set of concrete technical artifacts that can be evaluated by arbitrary third parties, preferably in an automated fashion. Formal and traceable specifications, theorems, and proofs hold far more weight than checklists and hand-over-heart promises.

Connie Heitmeyer: There are many notable successes in applying FM to safety-critical systems. For example, NASA has effectively applied it to aerospace software.^{2,3} However, the use of FM in developing and evaluating safety-critical software is still an exception and far from the state of the practice. The US and other governments regulate safety-critical systems, such as nuclear power plants, avionics systems, and medical devices such as pacemakers and infusion pumps.

Similarly, they also regulate security-critical systems. Since the 1970s, formal modeling and formal verification have been applied to security-critical systems—in many cases, to satisfy government regulations.⁴⁻⁷ FM is just as well suited to security concerns as it is to safety concerns.

Karl Levitt: FM proofs have been applied successfully to safety-critical systems because such systems lack unnecessary complexity. Systems for which security is important typically are built on complex commercial software. Lightweight FM has been applied for security. Many current attacks could be thwarted through identification of vulnerabilities—for example, buffer overflows could be prevented through code analysis, and SQL injection through formal analysis of SQL queries at runtime with respect to specification of allowable queries.

The *Department of Defense Trusted Computer System Evaluation Criteria*, or Orange Book, posed various levels of certification. The requirement that code be subject to the kind of checking for vulnerabilities offered by various static analysis tools is worth considering. Similarly, protocols could be required to be analyzed.

.....
FM and Time to Market

Computer: Minimizing time to market for apps and consumer, online, and retail market IT systems is a major concern in product releases. To our knowledge, FM doesn't have the reputation of decreasing time to market. Is this a misconception, or is there another viewpoint to consider?

Spafford: This is the general perception. Although the higher quality of software designed with FM arguably means faster development and less time debugging and patching, there's insufficient evidence of that being the case in typical software production environments. Current development

ROUNDTABLE PANELISTS



PAUL E. BLACK is a computer scientist in the Software Quality Group, Systems and Software Division, Information Technology Laboratory, at the National Institute of Standards and Technology. He has nearly 20 years of industrial experience developing software for integrated circuit design and verification, assuring software quality, and managing business data processing and has published in the areas of static analysis, software testing, software configuration control, networks and queuing analysis, FM, software verification, quantum computing, and computer forensics. He also founded and edits the online *Dictionary of Algorithms and Data Structures* (www.nist.gov/dads). Black received a PhD in computer science from Brigham Young University. He's a member of ACM and the IEEE Computer Society and a Senior Member of IEEE. Contact him at paul.black@nist.gov.



CONNIE HEITMEYER heads the Software Engineering Section of the Center for High Assurance Computer Systems at the US Naval Research Laboratory (NRL). Her research focuses on the formal modeling and analysis of critical software systems using software tools. She has published more than 150 technical papers covering a range of software-related research topics including requirements modeling, specification, and validation; formal verification using model checking and theorem proving; invariant generation; model-based test generation; security modeling; and real-time computing. A frequent invited speaker on software topics, Heitmeyer is the chief designer of NRL's Requirements Toolset, a set of tools for modeling, specifying, validating, and verifying critical systems adopted by more than 200 industry, government, and university groups. Heitmeyer received an MA in mathematics from the University of Michigan. She's an IEEE Fellow and a member of ACM. Contact her at constance.heimtaylor@nrl.navy.mil.



JOSEPH KINIRY is the Research Lead at Galois, Inc., of several programs—Rigorous Software Engineering, Verifiable Elections, High-Assurance Cryptography, and Audits-for-Good—as well as the CEO and Chief Scientist of Free & Fair, a Galois spin-off focusing on high-assurance election technologies and services. Prior to joining Galois in 2014, he was a full professor at the Technical University of Denmark, where he headed the Software Engineering section. He has extensive experience in FM, high-assurance software engineering, foundations of computer science and mathematics, and information security. Specific areas that he has worked in include software verification foundations and tools, digital election systems, smart cards, smartphones, critical systems for nation states, and CAD systems for asynchronous hardware. Kiniry received a PhD in computer science from Caltech. He's a member of ACM, IEEE, the IEEE Computer Society, the American Mathematical Society, and USENIX. Contact him at kiniry@galois.com.

depends on legacy software, which wasn't developed with FM, nor is it amenable to retrospective FM. Few developers know FM, and it isn't commonly taught in college. High-assurance development doesn't add obvious value in most vertical processes. Together

these factors suggest that there's little awareness or interest in employing FM more widely.

Levitt: No, this isn't a misconception. Many organizations look askance at any activities that delay the realization

of working code. Of course, we "formal methodists" claim that the early discovery of errors can save much effort downstream in the form of recalls.

McLean: Some studies indicate that extra time spent upfront applying FM



KARL LEVITT has been a professor in the Department of Computer Science at the University of California, Davis, since 1986. From 2005 to 2009, he was also a program director at the National Science Foundation, where he helped manage various security research programs and participated in interagency activities involving security. Levitt was previously at SRI International, where he was director of the Computer Science Laboratory from 1983 to 1986. With UC Davis colleagues and students, he participated in some of the early work on intrusion detection, including the first work on specification-based intrusion detection. His current research is focused on the generalization of intrusion detection to systems and networks that can also respond to detected attacks to minimize the risk to an operational model. He has had a longstanding interest in FM for program verification and for testing supported by symbolic evaluation. Levitt received a PhD in error-correcting codes from New York University. He is a member of ACM and the IEEE Computer Society. Contact him at levitt@cs.ucdavis.edu.



JOHN MCLEAN is superintendent of NRL's Information Technology Division (ITD) and helped create ITD's Center for High Assurance Computer Systems, establishing and heading its Formal Methods Section. As principal investigator for several NRL research projects, he published widely in the areas of FM and formal modeling for computer security. He has also held positions as an adjunct professor of computer science at the University of Maryland, the National Cryptologic School, and Troisième Cycle Romand d'Informatique. McLean received an MS in computer science and a PhD in philosophy from the University of North Carolina at Chapel Hill. He has served as an associate editor of *Distributed Computing*, the *Journal of Computer Security*, and *ACM Transactions on Information and System Security* and is a Senior Member of IEEE. Contact him at john.mclean@nrl.navy.mil.



EUGENE SPAFFORD is a professor in the Department of Computer Science and founder and executive director of the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University. His current research focuses on computer and network security, cybercrime and ethics, technology policy, and the social impact of computing. Spafford received a PhD in information and computer science from Georgia Tech. He's editor in chief of *Computers & Security*; a Fellow of IEEE, ACM, the American Association for the Advancement of Science (AAAS), and the International Information Systems Security Certification Consortium (ISCC)²; and a Distinguished Fellow of the Information Systems Security Association (ISSA). Contact him at spaf@purdue.edu.



JOSEPH WILLIAMS is the ICT sector lead and policy advisor for the State of Washington's Governor Jay Inslee. He's a seasoned technology executive who recently has been working with numerous companies grappling with the business and economics of cloud computing. His research agenda has lately focused on the use of collaboration technology to enable workplace and workforce redesign. Williams received a PhD in business from the University of Texas at Austin. He is a member of IEEE. Contact him at joseph.williams@commerce.wa.gov.

doesn't increase total development time due to the time saved debugging faulty software later. However, FM does lock in a development schedule with a relatively late-appearing testable version of the complete system, reducing flexibility. If I'm

developing an app and learn that my competition is going to release its version next month, I might be willing to release a relatively buggy version this month to be first to market. If I employed FM, that buggy version might not be available.

The NRL's Jim Kirby notes that a fundamental challenge to the US government and the economy in general is growing dependence on software coupled with an inability to quickly and affordably create and sustain quality software, where "quality" refers to low

software defect rates as well as security, robustness, and throughput. Half of cyber vulnerabilities are software defects, and the cost of avoiding and mitigating software errors approaches \$100 billion annually. And this doesn't even begin to address the problem of sustaining software.

Williams: A lamentable aspect of FM is the drag it imposes on time to market. Newer tools can close the gap, but there's no denying the added time cost. So, in a typical context, the race to minimum viable product doesn't really allow for FM. That said, as products and offerings mature, increasing accuracy and system robustness become key differentiators. Mature products and services will also demand the better security practices afforded by FM.

Black: FM likely won't reduce time to market because many releases are driven by preset calendar dates, not by quality targets.

Heitmeyer: The benefit of using FM is greater confidence in a software product's security or safety. The increased cost and time needed for FM can be reduced. First, FM might only be applied to a small portion of the software. In one NRL project,⁸ formal modeling and verification were only applied to the small code segment that implemented the separation kernel, which mitigated each memory access to ensure that data separation wasn't violated. Second, model checkers can significantly reduce the time and effort needed to check for security and safety violations. Third, tests that are systematically generated from a formal model⁹ can eliminate the redundancy often found in conventional software testing, reducing the number of tests and the time needed for testing.

Kiniry: The time to market for a high-assurance product depends on five main factors:

- › What is the technology framing of the product or service? Does it use “reasonable” foundations that lend themselves to formal verification? Was the system built from scratch for high assurance?
- › What level of assurance is necessary? Is it enough to prove that the system will never crash, or must we formally verify how it behaves? What class of adversary must be considered in the system's threat analysis and security requirements?
- › Is new research necessary to accomplish the rigorous engineering of a product or service?
- › Is there a team with the expertise to directly execute a project?
- › With the emergence of machine-checkable assurance evidence, can testing and evaluation done by governing agencies and their proxies—think FIPS (Federal Information Processing Standard) labs and voting system test labs—be made more efficient, improving time to market?

In 15 years of applying FM in R&D, it's our experience that the design and development of high-assurance systems is at least as expedient as alternative approaches, and sometimes more efficient. Note that the team's quality is very much a factor.

FM versus Testing

Computer: Testing currently is the primary means to support claims that specific security policies have been implemented properly. Should FM become the dominant approach? If so, how do we introduce this change, and how should it be combined with testing?

Williams: The state of testing is in disarray, not from a lack of tools or methodology but from the relentless pressure to release and public acceptance of rapid release of patches and updates, which penalizes deep testing.

Obviously, in fields such as safety or security, testing is still a paramount concern. The economic costs of security lapses that lead to breaches will likely drive the increasing application of FM to large-scale software development. Three costs arise from breaches:

- › commercial impact (for example, Target),
- › downtime and distraction (for example, Sony Pictures), and
- › legal liability (for example, any of the many HIPAA breaches).

The rise of sophisticated pricing/risk models is making it clear to data-driven companies that incorporating FM into their testing environment produces superior results.¹⁰

Spafford: I don't see FM replacing testing in general use in my lifetime. I also don't see developers adding FM unless it's shown to be highly cost-effective. That includes learning the techniques, acquiring the tools, and using the tools without any significant cost increase over current methods. FM would also have to be shown to be incrementally effective—adding new features should be quick rather than requiring reiteration of the whole method.

I don't see developers throwing out all that legacy code and redoing it using FM without some very significant forcing function. There needs to be either a suite of FM tools that's fast, easy to use, and applicable to the legacy code and methods in use or a stringent regulatory requirement (and cost-recovery pathway) to motivate adoption.

Gradual FM adoption or use in some vertical processes is possible, especially in “new” areas that don't depend on minimal development time and cost and thus can avoid legacy code reuse and the “penetrate and patch” mindset.

Kiniry: Testing and quality-assurance methods aren't a path toward high assurance. Providing high assurance in the real world, both in terms of a system's correctness and its security,



requires FM. That said, FM feeds testing and vice versa. Testing and FM aren't at odds, but instead complement each other. The goal of testing has less to do with the properties of the system than with the properties of the verification. Model validation—ensuring that formal models accurately reflect valid assumptions about the real world—is the key to formal verification, and the most realistic means to validate models is to compare them against real-world conditions using testing.

FM is becoming the dominant approach to creating high-assurance systems, but it's doing so as part of the integrated development environment, programming language, or software engineering philosophy currently in vogue—what my colleague Dan Zimmerman and I call “secret ninja” FM.¹¹

Levitt: Testing supported by analysis and tools is lightweight FM. To automate testing, assertions can be supplied against which executing code can be evaluated; the assertions can capture security properties, such as access control or acceptable flows. Test cases can be created manually or automated via symbolic evaluation to identify code paths that could affect security assertions and to generate test cases that could cause the security assertions to be false.

Heitmeyer: FM can be applied to a small part of the code. However, testing will continue to play an important role in the development of safety- and security-critical software. Whereas formal modeling and verification can produce proofs that the software satisfies specified security properties, testing and simulation can be used to validate its intended behavior.

Black: FM should be the dominant approach, supported by selected tests. I see three avenues to moving the emphasis to FM:

- › Supply FM claims of security through, for instance, assurance

cases, fully formal proofs, or partially formal arguments. As other developers see how these security claims can be justified, some will follow suit.

- › Support FM through, for example, procurement contracts requiring formal justification of certain parts, insurance discounts for formal justification of security, and promotion of FM use reported at computer science conferences.
- › Publicize test cases that failed to locate security vulnerabilities FM would find. Such case studies would, as a side effect, provide FM arguments for security properties of the target in question, once it's fixed.

Ideally, begin by thoroughly supporting all claims with FM-based arguments and exercises. Then, with the assurance that, in theory, the security policies will be implemented properly, supplement the assurance with testing. Clean-room or combinatorial tests should be run to confirm that there are no serious limitations or errors in the models, reasoning, assumptions, or implementation chain (compilers, libraries, hardware, and so on). In practice, the FM-based approach could be in parallel with testing, but they should be independent.

McLean: Testing might be the dominant approach, but some systems, such as US Department of Defense-grade cryptographic devices, require FM. Edsger Dijkstra's adage that testing can show the presence of bugs but not their absence still applies today, especially for security. Consider a random password generator. No amount of testing can reveal whether the passwords being generated are encryptions of confidential documents; this can be revealed only by analysis of the code itself. Testing might be sufficient for some properties due to the cost of FM, but other properties—for example, real-time response constraints—might

be best addressed by testing. But even so, there are FM approaches for verifying certain temporal properties.¹² FM can also be useful in showing that, for example, certain inputs can't influence the response time.

I'm most concerned about proving properties in intelligent systems whose behavior changes over time as the system learns. One possible solution is a non-bypassable system governor (similar to a reference monitor) that monitors all behavior and can be formally proven to prevent certain behaviors. This wouldn't be an easy application for FM, but I think it would be an even harder application for the testing community, especially in an environment where an adversary could manipulate the learning experiences the system was exposed to.

..... Detecting Unknown Vulnerabilities

Computer: Can FM predict or detect unknown vulnerabilities in existing software products? If this is already being done, can you provide specific examples or educational resources?

Black: It's not likely that FM can predict or detect entirely new classes of vulnerabilities. Most new types of attacks are precisely those that contravene assumptions or models, exercise unforeseen interactions, or involve operations that were dismissed as infeasible. Although FM might turn up surprising attack paths, generally we must limit what we ask of FM for practicality. That said, FM can readily detect unknown instances of vulnerabilities in existing software. A good rule of thumb is that if an FM approach newly applied to a large piece of existing software does not find some problem, the approach probably wasn't applied correctly.

Williams: Prediction and detection of software vulnerabilities is an interesting challenge. Obviously no

software system is 100 percent secure or reliable, but there's real economic value in getting as close as possible. The Barr Group's analysis of Toyota's spaghetti code behind its Camry's unintended acceleration couldn't definitively identify the causative agent for the failure but did predict that problems were likely to arise, so the application of FM would likely have helped.¹³ CIOs are starting to pay attention to the value of provably correct software,¹⁴ and financial regulators are also looking at a modeling approach that could leverage FM.¹⁵

McLean: One of the earliest FM success stories was in the area of crypto-

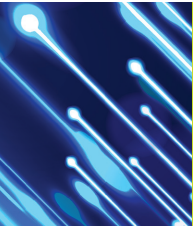
The advantage of such tools, which are increasingly being used to find software bugs, is that ordinary software engineers can use them. Moreover, they scale to million-line code bases. Recently, Astrée was used to prove the absence of runtime errors in the primary flight-control software, implemented in C, of the Airbus A340.¹⁹ Even with "false alarms," warnings about defects that can't occur in the real system, static analysis is still more efficient than many other forms of bug finding, since typically little time is needed to dismiss a spurious software defect.

An important industrial-strength static analysis tool is Microsoft's

static analysis tools. It'll be necessary to augment current static analysis tools to accommodate the specification beyond known classes of vulnerabilities. More conventional FM tools could detect unknown attacks.

Kiniry: FM tools can be used to examine existing software products that weren't developed with FM in mind and have no formal specifications. This situation is becoming more common every day due to three trends:

- › many new FM analysis technologies reason at the binary level without need of source code or other artifacts;
- › theoreticians and tool builders are learning from the past and not assuming users will learn a whole new language and write detailed formal specifications—thus, they're increasingly using elements like programs as specifications; and
- › tools' default specification—and consequently the power of their default reasoning behavior—has strengthened considerably over the years.



There are FM tools that drastically reduce the time for certain types of analysis, but a major breakthrough is needed to make FM tools cost-effective for all software development.

graphic protocol analysis. Richard Kemmerer, Catherine Meadows, and Jon Millen each used FM to find previously unknown flaws in a cryptographic protocol as early as 1994.¹⁶ FM also revealed problems in early designs for an NRL-developed embedded security device.⁸ In addition, there are various approaches for applying FM to object code, for both reengineering the code and finding software bugs, but this remains expensive. That said, although reengineering object code with formal tools is hard, I suspect that reengineering code without them is even harder.

Heitmeyer: Static analysis tools, such as Coverity¹⁷ and Astrée,¹⁸ are FM tools that analyze source code without executing it. They automatically detect vulnerabilities such as buffer and integer overflows, memory leaks, race conditions, and null pointer dereferences in C, C++, and other source code.

Static Driver Verifier (SDV). Microsoft found that bugs in device drivers cause 85 percent of system crashes in Windows XP. To detect such bugs, it developed SDV based on the on-the-fly model checker SLAM, which uses a set of interface rules and an OS model to determine whether a device driver correctly interacts with the OS kernel. During Windows 7 development, Microsoft applied SDV, after all other bug-finding tools, to 140 device drivers and found 270 real bugs.²⁰

Levitt: Starting with MOPS (Model checking Programs for Security properties)²¹ and, more recently, Coverity, there are static analysis tools that can expose unknown vulnerabilities—in known classes of vulnerabilities. As an aside, it would be informative to analyze the systems that have been recently attacked successfully to determine if the exploited vulnerabilities would've been detected by existing

FM and Software Liability

Computer: Software liability waivers, disclaimers, and insurance are proposed ways for companies to mitigate the consequences of data breaches and other security problems that impact shareholders, consumer confidence, and business continuity. What role, if any, can FM play here?

McLean: It's hard to see how software liability (legal or financial) can play a major role in the adoption of FM until

- › customers and the courts start holding software vendors responsible for software faults despite vendor assertions that their products come with no warranty, and

- › FM starts being seen as a necessary component of due diligence in software development and sustainment.

Kiniry: Companies and governments understand legal contracts, not FM. Assurance to a real-world customer has everything to do with culpability, warranties, and financial guarantees, and nothing to do with mathematical proof. Consequently, I expect that liability policy underwriters will be forced to begin to deeply understand the power and implications of the use of FM to make sound judgments about new policy demands. I know that some product companies are beginning to pursue fiscal assurances that are backed by mathematical ones.

Spafford: If collected metrics can show FM lessening the number and severity of failures, insurance companies might offer reduced rates for products developed using FM. This might encourage businesses to adopt FM in their development. It won't be quick, however, either to gather the necessary actuarial data or for customers to be willing to adopt. As it is, insurance in this space isn't yet widely adopted.

Williams: Legal liability arises whenever there's harm. One critical defense a company can assert is due diligence (and the plaintiff would point to a lack of due diligence). The pivot around due diligence is the prevailing or reasonable standard of care. The application of competent FM would provide evidentiary collateral that helps meet the due-diligence standard of care. The flurry of interest in driverless cars will motivate automobile manufacturers to demonstrate that they applied very modern techniques for testing and safety prediction.²²

Levitt: Vendors of safety-critical systems are surely liable for damages associated with attacks that exploit vulnerabilities in their systems. I could imagine a regulatory agency

mandating the use of certain kinds of FM by vendors. The vendors would be absolved of liability for errors that aren't addressed by the mandated FM.

Heitmeyer: While the use of FM isn't mandated, formal evidence can help demonstrate the safety and security of software products. Concerned about the safety of software-intensive medical devices such as infusion pumps and pacemakers, the US Food and Drug Administration and private agencies such as the Association for the Advancement of Medical Instrumentation recently recommended that vendors submit "safety assurance cases" for their devices. A safety assurance case is a systematic, structured method for supporting a stated claim with a top-level claim of safety. Formal models and proofs can provide part of the evidence supporting the claim.

Toward Automated FM

Computer: To our knowledge, FM still requires considerable manual effort; FM isn't as automated as one might hope. If you agree, is that changing?

Williams: Current FM tools are clumsy and difficult to use, and aren't really accessible to researchers without in-depth training. If FM use grows, the tools will inevitably improve, just as they have for data visualization.

Black: FM is far more automated than it once was, and such improvements will continue. However, FM will never be as automated as we would like. When one system becomes easy or routine to use, society will want solutions that demand bigger and more complex systems. As our satisfiability (SAT) solvers and model checkers handle thousands of variables at the push of a button, we'll forge ahead to solve problems involving millions of variables.

Spafford: I'm unaware of recent developments in the field—I don't follow

FM work closely. If there were breakthroughs in making it more automated, I'd expect to see that make a difference in how and where the tools are used.

Kiniry: Modern FM tools are significantly more automated than those of the past. Automation requires rich foundations (like default specifications), making tough decisions (such as trading off soundness for automation), and putting in hard work (such as deciding a tool is going to be complete and automated and thus its tooling is significantly more complicated for myriad theoretical and practical reasons).

Automation is pervasive today. A driver for automation is the mainstream adoption of tooling that, unbeknown to the developer, is using FM; examples include advanced type systems, behavioral refactoring tools, automated test generation, and program synthesis. These technologies and others, especially when incorporated into mainstream development environments, create a feedback cycle for the adoption and impact of FM.

The manual effort, in my experience, is still focused on the creation of novel artifacts: domain models, requirements, designs, assertions—all of which amount to specifications. And, while the automation of specifications has seen some advancement (for example, from the extraction of system architectures to the abstraction of formal specifications), we still have a long way to go before we can wave our hands and say, "... and it should be secure!" After all, even Geordi and Data had to program by hand in *Star Trek: The Next Generation*.

McLean: There are FM tools that drastically reduce the time for certain types of analysis—NRL's Software Cost Reduction toolset (www.nrl.navy.mil/itd/chacs/5546/scr) comes to mind—but a major breakthrough is needed to make FM tools cost-effective for all software development and sustainment. The trick with such tools now is limiting their use to those sections of code that

you have to formally analyze to guarantee a property.⁸ This approach lowers the cost of production by limiting the application of FM to a manageable component as well as sustainment, since functional aspects of the system can be changed without affecting the security verification. Just as David Parnas advocates modularizing those aspects of a system that might change,²³ one should also modularize those aspects of a system that are important enough to use formal verification.

Heitmeyer: Numerous powerful tools have recently been developed that reduce the effort in applying FM. Certain relatively automatic, easy-to-use static analysis tools are customized to find certain classes of software defects, code vulnerabilities (Coverity and Astrée), and device driver bugs (SDV). Moreover, they don't require users to create models or to formulate assertions, properties that the code is expected to satisfy.

Levitt: Model checkers and SAT solvers provide significant proof automation, as do recent efforts to automate the generation of loop invariants.

FM Resources

Computer: Where are some good resources that would enable novices interested in computer/software/IT security to quickly and efficiently learn about FM?

Kiniry: By recognizing how novices today already apply (hidden) FM, practitioners can get over the fear or worry that goes along with diving into this rich discipline. Examples include new programming languages with a rich type system, such as Scala (www.scala-lang.org), Rust (www.rust-lang.org), TypeScript (www.typescriptlang.org), and Haskell (www.haskell.org); automated specification and reasoning systems that look and feel like programming languages, such as

Microsoft Research's F* (<https://fstar-lang.org>) and Galois's Cryptol (<http://cryptol.net>); and various tools available in the Microsoft .Net, Java virtual machine (JVM), and LLVM software development platforms, such as Code Contracts ([https://msdn.microsoft.com/en-us/library/dd264808\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd264808(v=vs.110).aspx)), the Java Modeling Language (JML; www.eecs.ucf.edu/~leavens/JML), and the Clang Static Analyzer (<http://clang-analyzer.lvm.org>), respectively.

I challenge readers to begin to dig into the foundations of modern FM and work toward understanding how these languages, reasoning systems, and tools operate and can dramatically improve both productivity and product quality.

Williams: With 80+ conferences in 2016 that address FM in whole or part, there's no lack of opportunity to network with researchers and learn about the field. Jonathan Bowen and Mike Hinchey's article "Ten Commandments of Formal Methods... Ten Years Later"²⁴ is a good reference, though it needs an update. There's also a 45-lecture introduction to FM available for free online at http://onlinevideolecture.com/?course_id=1306, as well as numerous useful materials on FM from IEEE and other professional organizations.


McLean: My article on "Security Models" in the *Encyclopedia of Software Engineering*,²⁵ though a bit dated, is still considered to be a good general summary of the application of FM to the analysis of security properties. For a more historical overview, see my oral history for the Charles Babbage Institute,²⁶ which also shows the controversies that can arise when formally analyzing security properties. An interesting discussion of the culture and sociology surrounding FM can be found in Donald Mackenzie's *Mechanizing Proof: Computing, Risk, and Trust* (MIT Press, 2004). Many computer security texts—for example,

Ross Anderson's *Security Engineering: A Guide to Building Dependable Distributed Systems* (2nd ed., Wiley, 2008), Matt Bishop's *Computer Security: Art and Science* (Addison-Wesley, 2002), and Rita Summers's *Secure Computing: Threats and Safeguards* (McGraw-Hill, 1997)—also contain sections on the application of FM to security.

Spafford: I have no idea. If something accessible and concise becomes available, I'll read it and provide it to my students.

Black: Perhaps we can prevail upon Donald Knuth to make FM the subject of a new volume of *The Art of Computer Programming*.

Levitt: Currently popular and robust verification tools are well documented and suitable for classroom use. There are several excellent tutorials on the current theorem provers, including the University of Texas's ACL2 (www.cs.utexas.edu/users/moore/acl2) and SRI International's Prototype Verification System (PVS; www.csl.sri.com/projects/pvs).

We thank the panelists in our roundtable discussion for sharing their expertise and for their candor. What do you think: Is FM relevant to security? And, if so, what's the best way to incorporate FM into software development and delivery? 

References

1. C.A.R. Hoare, "How Did Software Get So Reliable without Proof?," *FME'96: Industrial Benefit and Advances in Formal Methods*, M.-C. Gaudel and J. Woodcock, eds., LNCS 1051, 1996, Springer, pp. 1-17.
2. K. Havelund, M. Lowry, and J. Penix, "Formal Analysis of a Spacecraft Controller Using SPIN," *IEEE Trans. Software Eng.*, vol. 27, no. 8, 2001, pp. 749-765.



3. G.J. Holzmann, "Mars Code," *Comm. ACM*, vol. 57, no. 2, 2014, pp. 64–73.
4. L. Robinson and K.N. Levitt, "Proof Techniques for Hierarchically Structured Programs," *Comm. ACM*, vol. 20, no. 4, 1977, pp. 271–283.
5. C.E. Landwehr, C.L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems: Retrospective," *ACM Trans. Computer Systems*, vol. 2, no. 3, 1984, pp. 198–222.
6. C. Meadows, "Analysis of the Internet Key Exchange protocol Using the NRL Protocol Analyzer," *Proc. IEEE Symp. Security and Privacy (SP 99)*, 1999, pp. 216–231.
7. D. Greve, M. Wilding, and W.M. Vanfleet, "A Separation Kernel Formal Security Policy," *Proc. 4th Int'l Workshop ACL2 Theorem Prover and Its Applications (ACL2 03)*, 2003; www.cs.utexas.edu/users/moore/acl2/workshop-2003/contrib/greve-wilding-vanfleet/security-policy.pdf.
8. C.L. Heitmeyer et al., "Applying Formal Methods to a Certifiably Secure Software System," *IEEE Trans. Software Eng.*, vol. 34, no. 1, 2008, pp. 82–98.
9. A. Gargantini and C. Heitmeyer, "Using Model Checking to Generate Tests from Requirements Specifications," *Software Engineering—ESEC/FSE'99*, O. Nierstrasz and M. Lemoine, eds., LNCS 1687, Springer, 1999, pp. 146–162.
10. W. Lobb, "Business Perspectives on Provably-Correct Software," presentation, Bits&Chips Conf., 2015; www.researchgate.net/publication/297732321_Business_Perspectives_on_Provably-Correct_Software.
11. J.R. Kiniry and D.M. Zimmerman, "Secret Ninja Formal Methods," *Proc. 15th Int'l Symp. Formal Methods (FM 08)*, 2008, pp. 214–228.
12. N.S. Bjørner et al., "Verifying Temporal Properties of Reactive Systems: A STeP Tutorial," *Formal Methods in System Design*, vol. 16, no. 3, 2000, pp. 227–270.
13. M. Barr, "An Update on Toyota and Unintended Acceleration," blog, 26 Oct. 2013; <http://embeddedgurus.com/barr-code/2013/10/an-update-on-toyota-and-unintended-acceleration>.
14. M. Heusser, "Can New Software Testing Frameworks Bring Us to Provably Correct Software?," *CIO*, 13 Feb. 2013; www.cio.com/article/2388410/agile-development/can-new-software-testing-frameworks-bring-us-to-provably-correct-software-.html.
15. K. Jones, *Regulatory Analytics and Data Architecture (RADAR)*, CIFR Working Paper No. WPO68/2015, Centre for Int'l Finance and Regulation, Aug. 2015; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2628939.
16. R. Kemmerer, C. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis," *J. Cryptology*, vol. 7, no. 2, 1994, pp. 79–130.
17. A. Bessey et al., "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," *Comm. ACM*, vol. 53, no. 2, 2010, pp. 66–75, 2010.
18. P. Cousot et al., "The ASTRÉE Analyzer," *Programming Languages and Systems*, M. Sagiv, ed., LNCS 3444, Springer, 2005, pp. 21–30.
19. D. Delmas and J. Souyris, "Astrée: From Research to Industry," *Static Analysis*, H.R. Nielson and G. Filé, eds., LNCS 4634, Springer, 2007, pp. 437–451.
20. T. Ball, V. Levin, and S.K. Rajamani, "A Decade of Software Model Checking with SLAM," *Comm. ACM*, vol. 54, no. 7, 2011, pp. 68–76.
21. H. Chen and D. Wagner, "MOPS: An Infrastructure for Examining Security Properties of Software," *Proc. 9th ACM Conf. Computer and Comm. Security (CCS 02)*, 2002, pp. 235–244.
22. M. Harris, "Why You Shouldn't Worry about Liability for Self-Driving Car Accidents," *IEEE Spectrum*, 12 Oct. 2015; <http://spectrum.ieee.org/cars-that-think/transportation/self-driving/why-you-shouldnt-worry-about-liability-for-selfdriving-car-accidents>.
23. C.L. Heitmeyer, "Software Cost Reduction," *Encyclopedia of Software Engineering*, J.J. Marciniak, ed., John Wiley & Sons, 2002.
24. J.P. Bowen and M. G. Hinchey, "Ten Commandments of Formal Methods ... Ten Years Later," *Computer*, vol. 39, no. 1, 2006, pp. 40–48.
25. J. McLean, R.R. Schell, and D.L. Brinkley, "Security Models," *Encyclopedia of Software Engineering*, J.J. Marciniak, ed., John Wiley & Sons, 2002.
26. J.R. Yost, "An Interview with John D. McLean," Charles Babbage Inst., Univ. of Minnesota, 22 Apr. 2014; <http://conservancy.umn.edu/handle/11299/164989>.

KIM SCHAFFER is a researcher at the National Institute of Standards and Technology. His research interests include authentication and security validation development. Schaffer received a DSc in information assurance from Capitol Technology University. He is a Certified Information Systems Security Professional and a Senior Member of IEEE. Contact him at kim.schaffer@nist.gov.

JEFFREY VOAS is a computer scientist at the National Institute of Standards and Technology. His research interests include the Internet of Things and fundamental computer science shortcomings. Voas received a PhD in computer science from the College of William and Mary. He is a contributing editor for *Computer's* Cybertrust column and a Fellow of IEEE and the American Association for the Advancement of Science (AAAS). Contact him at j.voas@ieee.org.

DISCLAIMER

Any mention of commercial products or organizations is for informational purposes only; it is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.