Whatever Happened to Formal Methods for Security?

(IEEE *Computer* Magazine, August 2016 publication on Supply Chain Security, regular paper in the *Perspectives* section)

J. Voas and K. Schaffer

We asked 7 experts 7 questions to find out what has occurred recently in terms of applying formal methods (FM) to security-centric, cyber problems. We are continually reminded of the 1996 paper by Tony Hoare "How did Software Get So Reliable Without Proof?" [1]

In that vein, how did we get so insecure with proof? Given daily press announcements concerning new malware, data breaches, and privacy loss, is FM still relevant or was it ever? Our experts answered with unique personal insights. We were curious as to whether this successful methodology in "safety-critical" has succeeded as well for today's "build it, hack it, patch it" mindset. Our experts were John McLean (Naval Research Labs), Paul Black (National Institute of Standards and Technology), Karl Levitt (University of California at Davis), Joseph Williams (CloudEconomist.Com), Connie Heitmeyer (Naval Research Labs), Eugene Spafford (Purdue University), and Joseph Kiniry (Galois, Inc.). The questions and responses follow.

1) Most are aware that FM has been highly successful in safety-critical systems over the past decades. Much of that success stems from those systems being deployed in regulated industries. If you agree with this claim, it begs two questions: (1) Is FM as well suited to security concerns, and (2) if assurance is more compliance and self-governance

Joseph Williams

Formal methods have been successfully applied to safety-critical systems. One reason is the overwhelming evidence that formal methods do result in safer systems. The application of formal methods also satisfies a legal burden of proof regarding due diligence. It is likely that as researchers endeavor to squeeze more accuracy out of their analytical models (e.g., pricing models, or voice translation systems) that formal methods will find their niche, but these are inevitably market-driven.

Gene Spafford

Safety-critical systems are more regulated, and failures costly. Thus, investment in better design and use of tools (including formal methods) are justified by economics. Currently, security issues in most systems (except some highly-regulated ones) do not have the same kinds of economic pressures and constraints. Most consumer development still stresses time to market rather than quality. The burgeoning field of "Internet of things" is even more in this direction.

Market pressure is thus unlikely to have much effect. Regulation, and possibly (eventually) insurance costs may make a difference. However, security and privacy issues do not have good costing information associated with them, and most losses to date seem to observers to be temporary (i.e., few or no significant companies have gone out of business because of poor security).

Paul Black

Yes, FM is well-suited to security concerns. The modeling and complete evaluation that can accompany formal methods may discover problems that testing or human review would not.

FM will not answer questions which are not posed and models are limited reflections of reality. Hence, Knuth's warning 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 is used to get the job done correctly in the first place. FM reduces the need for extensive testing and greatly reduces the uncertainty in the schedule that accompanies testing. Software with significantly fewer bugs also helps preserve an entity's reputation and minimizes costs of recalls or patches.

John McLean

Security certainly has a large compliance component. Verification cannot 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. These methods can detect, programming mistakes leading to security vulnerabilities (eg, buffer overflows), and also subtle errors that permit side channel or covert channel attacks. Ironically, the very feature of security that makes it hard for formal methods (or any sort of verification) is the very feature that makes formal analysis so important: security, unlike functional correctness, is not always easy to specify. That's why some of the earliest work on applying formal methods to computer security focuses on security models — i.e., formal specifications and analyses of security, rather than verifications of code.

Joseph Kiniry

While a small number of regulated industries have "forced" the use of applied 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 the R&D challenges of our customers, and many of those solutions result in tools that are used by our customers 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 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 formal methods in developing aerospace software (see, e.g., [2], [3]*). However, the use of formal methods in developing and evaluating safety-critical software is still an exception and far from the state of the practice. The U.S. and other governments have regulated safety-critical systems, such as nuclear power plants, avionics systems, and medical devices e.g., pacemakers and infusion pumps. Similarly, they have also regulated security-critical systems. Since the 1970s, formal methods, such as formal modeling and formal verification, have been applied to security-critical systems, in many cases, to satisfy government* *regulations. Among the many examples of practical systems where formal models and proofs have been used are* [4], [5], [6] *and* [7]. *Formal methods are just as well suited to security concerns as they are to safety concerns.*

Karl Levitt

FM has been applied successfully to safety-critical systems and the lightweight formal methods can be applied for security. Many of the current attacks (still) involve buffer overflows or SQL injection which could be prevented through identification of the vulnerabilities. Buffer overflows through code analysis and SQL injection through formal analysis of SQL queries at runtime with respect to a specification of allowable queries.

The 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.

(2) Minimizing time-to-market for apps and other consumer, on-line, and retail market IT systems is a major concern in getting products released. FM does not have a reputation of decreasing time-to-release to our knowledge. Is this a misconception or is there another viewpoint to consider?

Gene Spafford

This is the general perception with no significant evidence to refute it. Although it can be argued higher quality of software designed with formal methods (FM) means faster development and less time debugging and patching, there is insufficient evidence of that being the case in typical software production environments.

Current development depends on legacy software, which was not developed with, nor is amenable to retrospective FM. Few developers know FM, and I don't see it as commonly taught in college curricula. High assurance development does not add obvious value in most verticals. All of these together mean there is little awareness or interest in employing FM more widely outside those verticals.

Karl Levitt

This is not a misconception. Many organizations look askance at any activities that delay the realization of working code, e.g., the stages in the software lifecycle. Of course, we formal "methodists" claim that the early discovery of errors can save much effort downstream in the form of recalls.

John McLean

Some studies indicate extra time spent applying formal methods up front does not increase total development time due to the time saved debugging faulty software later. However, the use of formal methods does lock in a development schedule where a testable version of the complete system appears relatively late in the process. This reduces flexibility. If I'm developing an app and learn that my competition is going to release their version next month, I may be willing to release a relatively buggy version this month to be first on the market. If I employed formal methods, that buggy version may not be available.

Jim Kirby, notes that a fundamental challenge facing Defense, government, and the economy is growing dependence on software coupled with an inability to quickly and affordably create and sustain quality software, where "quality" refers both to low software defect rates and security, robustness, and throughput. Half of cyber vulnerabilities are software defects and the cost of avoiding and mitigating software errors approaches $100B annually. And this doesn't begin to address the problem of sustaining software.

Joseph Williams

One of the lamentations regarding formal methods is the drag it imposes on time-to-market. Newer tools can close the gap, but there is no denying the time costs of formal methods. So in a typical context, the race to MVP (minimum viable product) does not really allow for the application of formal methods. That said, as products and offerings mature (e.g., Zillow), increasing accuracy and system robustness become key differentiators. As these product and services mature they also will demand the better security practices that can be designed by the incorporation of formal methods into the development process.

Paul Black

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

Connie Heitmeyer

The benefit of using formal methods is increased confidence in the security (or safety) of a software product. The increased cost and time need to apply formal methods can be reduced. First, the formal methods may only be applied to a small portion of the software. In the project described in [8]*, formal modeling and formal verification were only applied to the small code segment that implemented the separation kernel, which mitigated each memory access to ensure that data separation, was not violated. Second, model checkers can significantly reduce the time and effort needed to check for security and safety violations. Third, tests which are systematically generated from a formal model (see, e.g.,* [9]*, can eliminate the redundancy often found in conventional software testing, thus reducing the number of tests and the time needed for testing.*

Joseph Kiniry

The time-to-market for a high-assurance product depends on five main factors. (1) What is the technology framing of the product or service? Does it use "reasonable" foundations that lend themselves to formal verification? Was it built from scratch for high-assurance? (2) 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 product's threat analysis and security requirements? (3) Is new research necessary to accomplish the rigorous engineering of a product? (4) Is there a team with the expertise to directly execute a project? (5) With the emergence of machine-checkable assurance evidence, can testing and evaluation done by governing agencies and their proxies (think FIPS labs and VSTLs) be made more efficient, improving time to market. If the answer to these questions is within past R&D in applied FM, then the design and development of high-assurance systems is at least as expedient as alternative approaches, and is sometimes more efficient.

Note that the quality of the team and its performers is very much a factor.

(3) Testing currently is the dominant approach to support claims that specific security policies have been implemented properly. If true, should FM become the dominant approach and how do we introduce this change? And how should testing and FM be combined for assurance?

Joseph Williams

The state of testing is in disarray, not from a lack of tools or methodology, but from the relentless pressure to release and the public acceptance for rapid release of patches and updates that penalizes deep testing. Obviously, in fields such as safety or security, testing is still a paramount concern. The economic costs of security lapses will likely drive the increasing *application of formal methods to large-scale software developments. Three costs arise from breaches: commercial impact (e.g., Target), downtime and distraction (e.g., Sony Pictures), and legal liability (any of the many HIPAA breaches). The rise of sophisticated pricing / risk models is making it clear to data-driven companies incorporating formal methods into their testing environment produces superior results* [10].

Gene Spafford

I do not see testing being replaced by FM, in general use, in my lifetime. I also do not see FM being added unless shown to be highly cost-effective. That includes developers learning the techniques, acquiring the tools, and using to tools without any significant cost increase over current methods. It would also have to be shown to be incrementally effective – adding new features should be quick rather than requiring reiteration of the whole method.

There is the problem of legacy code and reuse. I simply don't see developers throwing out all that code and redoing it using formal methods unless there is some very, very significant forcing function. Either there needs to be some suite of FM developed that is fast, easy to use, and applicable to the legacy code and methods in use, or else there must be some stringent regulatory requirement (and cost recovery pathway) to force adoption.

Gradual adoption or use in some new verticals is possible, especially if they are "new" areas that don't depend on minimal develop time and cost, and thus can avoid legacy code and embracing the "penetrate and patch" mindset.

Joseph Kiniry

Testing and Q/A methods are not a path towards high-assurance. To provide high-assurance in the real world, both in terms of a system's correctness and its security, requires FM. That being said, FM feeds testing and vice versa. Testing and FM are not at odds with each other, but instead complement one another. The goal of testing has little to do with the properties of the system and is entirely to do with the properties of the verification. Model validation, ensuring that formal models accurately reflect valid assumptions about the real world, is the key when it comes to formal verification. And the most realistic means by which to validate models is to compare them against the real world using testing.

FM is becoming the dominant approach to creating high-assurance systems, but as part of the IDE, or the programming language, or the way of thinking that is suddenly in vogue, all of which are what my colleague Dan Zimmerman and I call "Secret Ninja Formal Methods".

Karl Levitt

Testing supported by analysis and tools is lightweight formal methods. 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 by the use of symbolic evaluation to identify paths in the code that could affect security assertions and to generate test cases that could cause the security assertions to be false.

Connie Heitmeyer

Formal methods can be applied to a small part of the code. However, testing will continue to play an important role in the development of safety-critical and security-critical software. While formal modeling and verification can produce proofs that the software code satisfies specified security properties, validation is also needed that the software behaves as intended. While formal verification can demonstrate that the software behavior is right, i.e., satisfies a set of critical properties, testing and a related technique, simulation, can be used to gain confidence that the software has the intended (i.e., right) behavior.

Paul Black

Yes, FM should be the dominant approach, supported by selected tests. I see three avenues to moving the emphasis to FM: FM supply (push), FM demand (pull), and test limitation exposure. The first approach is to supply FM claims of security, for instance through assurance cases, fully formal proofs, or partially formal arguments. As people see these how security claims can be justified, there will be some tendency to follow. The second approach is demanding FM support. Examples might include, procurement contracts requiring a level of formality or FM justification for certain parts; insurance industry providing discounts with formal justification of security; or following computer science conferences and acknowledging the use of FM. The last approach is shaming of test-only approaches by publicizing cases which failed to find security vulnerabilities that FM would find. Such case studies and examples would, as a side effect, provide FM arguments for security properties of the target in question, once it is fixed.

Ideally, one begins by thoroughly supporting all claims with FM-based arguments and exercises. When one has the assurance that, in theory, the security policies are (will be) implemented properly, then one supplements the assurance with testing. Clean Room or combinatorial type 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, etc.). In practice the FM-based approach could be parallel with developing tests and test plans, however, they should be independent.

John McLean

Although testing may be the dominant approach, for some systems, such as the type 1 cryptographic devices require formal methods. Dijkstra's adage that testing can be used to show the presence of bugs but not their absence is still applicable today. This is especially true 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 an analysis of the code, itself. Testing may be perfectly sufficient for some properties due to the cost of formal methods, but other properties -- for example, real-time response constraints -- may be best addressed by testing. But even here, there are some formal method approaches for verifying certain temporal properties [11]. *Formal methods can also be useful, for example, in showing that certain inputs cannot influence the response time.*

I'm most concerned about proving properties in intelligent systems whose behavior changes over time as the system learns. One approach to this might be a non-by-passible system governor (similar to a reference monitor) that monitors all behavior and which can be formally proven to prevent certain behaviors. This would not be an easy application for formal methods, 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.

(4) Can FM predict or detect unknown vulnerabilities in existing software products? If this is already being done, can you point the readers to a handful of suggested educational resources or specific papers?

Paul Black

*It is not likely that FM can predict or detect entirely new classes of vulnerabilities. Most new classes of attacks are precisely those that contravene assumptions or models, exercise unforeseen interactions, or operations that were dismissed as infeasible. Although some 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 some FM approach newly applied to a large piece of existing software does *not* find some problem, the approach was probably not applied correctly.*

Joseph Williams

Prediction and detection of software vulnerabilities is an interesting challenge. Obviously no software system is 100% secure or 100% reliable, but there is 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 problem could not definitely identify the causative agent for the failure, but the analysis did predict that problems were likely to arise. So the application of formal methods would likely have helped here [12]. *CIOs are starting to pay attention to the value of provably correct software* [13] *And financial regulators are also looking at an approach to modeling that could leverage formal methods* [14].

John McLean

One of the very early success stories was in the area of cryptographic protocol analysis. Richard Kemmerer, Catherine Meadows, and Jon Millen each used formal methods to find unknown flaws in a cryptographic protocol as early as 1994 [15]. *Formal methods also revealed problems in early designs for an NRL developed embedded security device* [8]. *There are also approaches for applying formal methods to object code, both for re-engineering the code and for finding software bugs, but this is still expensive. That said, although re-engineering object code with formal tools is hard, my suspicion is that re-engineering code without them is even harder.*

Connie Heitmeyer (NRL)

Static analysis tools, such as Coverity [16] *and Astrée* [17], *are formal methods tools that analyze source code without executing it. Such tools automatically detect "code vulnerabilities," such as buffer overflows, integer overflows, memory leaks, race conditions, and null pointer dereferences in C, C++, and other source code. Such tools are increasingly being used by software practitioners to find software bugs. The advantage of static analysis tools is that they can be used by ordinary software engineers. Moreover, they scale to million-line code bases. Recently, Astrée was used to prove the absence of run-time errors in the primary flight control software, implemented in the C language, of the Airbus A340* [18]. *Even with "false alarms," warnings about defects that cannot 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.*

Another important static analysis tool is Microsoft's Static Driver Verifier (SDV). Microsoft found that bugs in device drivers cause 85% of the system crashes in their operating system Windows XP. To detect such bugs, Microsoft developed SDV, an industrial-strength static analysis tool, based on the on-the-fly model checker SLAM, which uses a set of interface rules and an operating system model to determine whether a device driver correctly interacts with the operating system kernel. During the development of another Microsoft operating system, Windows 7, SDV was applied, after all other bug-finding tools, to 140 device drivers, and yet found 270 real bugs [19]*!*

Karl Levitt

Yes. Starting with the MOPS static analysis system and more recently the analyzers developed by Coverity, there are 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 have been detected by existing static analysis tools. It will 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.*

Joseph Kiniry

FM tools can be used to examine existing software products which were not developed with FM in mind and have no specifications. This delightful situation is becoming more common every day due to three trends. (1) Many new FM analysis technologies reason at the binary level without need of source code or other artifacts. (2) Theory and tool builders are learning from the past and not assuming users will learn a whole new language and write detailed formal specifications, so they are increasingly using things like programs as specifications. (3) The default specification—and consequently the power of the default reasoning behavior—of tools has strengthened considerably over the years.

(5) Software liability and software insurance are sometimes mentioned as ways that companies mitigate the consequences of data breaches and other problems that impact shareholders, consumer confidence, and business continuity. What role can FM play here, if any?

John McLean

It's hard to see how software liability (legal or financial) can play a major role in the adoption of formal methods until (1) customers and the courts start holding software vendors responsible for software faults despite vendor assertions that their products come with no warranty, and (2) formal methods start being seen as a necessary component of due diligence in software development and sustainment.

Joseph Kiniry

Companies and government understand legal contracts, not formal methods. Assurance to a real world customer has everything to do with culpability, warranties, and financial assurance, and nothing to do with mathematical proof. Consequently, I expect that policy writers will be forced to begin to deeply understand the power and implications of the use of formal methods to make sound judgements about new policy demands. I know that some product companies are beginning to pursue fiscal assurances that are backed by mathematical assurances.

Gene Spafford

If collected metrics can show FM lessening the number and severity of failures, insurance companies may offer reduced rates for products developed using FM. This might encourage businesses to adopt FM in their development. It will not 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 is not yet widely adopted.

Joseph Williams

Legal liability arises wherever there is a 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 formal methods would provide evidentiary collateral that helps meet the due diligence standard of care. The flurry of interest in driverless cars will (pardon the pun) drive automobile manufacturers to demonstrate that they applied very modern techniques for testing and safety prediction [20].

Karl Levitt

Harking back to safety-critical systems, vendors of such systems are surely liable for damages associated with attacks that exploit vulnerabilities in their systems. I could imagine that a *regulatory agency could mandate the use of certain kinds of FMs by vendors. The vendors could be absolved of liability for errors that are not addressed by the mandated FMs.*

Connie Heitmeyer

While the use of formal methods is not mandated, formal evidence can support approaches to demonstrating the safety and security of software products. Recently, the U.S. Food and Drug Administration and private agencies such as the AAMI (Association for the Advancement of Medical Instrumentation), concerned about the safety of software-intensive medical devices such as infusion pumps and pacemakers, recommended vendors submit "safety assurance cases" to demonstrate the safety of 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.

(6) To our knowledge, FM still requires a decent amount of manual effort; FM is not as automated as one might hope. If you agree, is that changing?

Joseph Williams

Current tools for formal methods are clumsy and difficult to use, and these tools are not really accessible to researchers without in-depth training. If formal methods grow in application, it is inevitable that the tools will improve – just as they have for data visualization.

Paul Black

FM is far more automated than it once was, and I believe that such improvements will continue. However, FM will never be as automated as one might hope. As soon as one level of use has become easy or routine, society will want solutions that demand bigger and more complex systems. As our SAT solvers and model checkers handle thousands of variables at the push of a button, we will forge ahead to solving problems that use millions of variables.

Gene Spafford

I am unaware of recent developments in the field — I do not follow FM work closely. If there were breakthroughs in making it more automated, I would expect to see that make a difference in how and here the tools are used.

Joseph Kiniry

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

Automation is pervasive today. A driving vehicle for automation is the mainstream adoption of tooling that, unbeknownst to the developer, is using formal methods; 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 (e.g., from the extraction of architectures from systems to the abstraction of formal specifications from source), 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.

John McLean

There are formal methods tools which drastically reduced the time for certain types of analysis (Connie Heitmeyer's work on NRL's Software Cost Reduction toolset comes to mind here), but I have yet to see the major breakthrough that makes them cost effective for all software development and sustainment. The trick for using them now is to limit those sections of code that you have to formally analyze to guarantee a property. This approach lowers production cost by limiting the application of formal methods to a manageable component and lowers the cost of 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 may change, one should also modularize those aspects of a system that are important enough to use formal verification.

Connie Heitmeyer

A number of powerful tools have been developed recently, reducing the effort in applying formal methods. What makes certain static analysis tools relatively automatic and easy to use is that they are customized to find certain classes of software defects, code vulnerabilities in the case of Coverity and Astree, and device driver bugs in the case of SDV. Moreover, they do not require users to create models or to formulate assertions, properties that the code is expected to satisfy.

Karl Levitt

Yes, there is a definite change.

(7) Where are the better resources from which to quickly and efficiently learn about formal methods for novices interested in computer/software/IT security?

Joseph Kiniry

By recognizing some of what novices do today as applied (but hidden) formal methods, practitioners can get over the fear or worry that goes along with diving into this rich discipline. For example, new programming languages with a rich type system, such as Scala, Rust, Typescript, or Haskell. The automated specification and reasoning systems that looks and feels like programming, such as Microsoft Research's F or Galois's Cryptol. The tools available in the .Net, JVM, and LLVM platforms, such as Code Contracts, JML, and the Clang Static Analyzer, respectively.*

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

Joseph Williams

With 83+ conferences in 2016 that address in whole or in part the field of formal method, there are no lack of places to go to network with researchers and learn about formal methods. It needs an update, but Bowen & Hinchey's "Ten Commandments of formal methods… ten years later" article in Computer (Jan 2006) [21] is a good reference. There is also a 45 lecture online introduction to formal methods available for free online (http://onlinevideolecture.com/?course_id=1306), as well as numerous materials from IEEE and other professional organizations.

John McLean

If one is interested in a general overview for their application in security my article on "Security Models" in the *Encyclopedia of Software Engineering* (Wiley & Sons), though a bit dated, is still considered to be a good general overview of the application of formal methods to the analysis of security properties. For a more historical overview, see my oral history for the Babbage Institute [22], it also shows the controversies that can arise when formally analyzing security properties. An interesting discussion of the culture and sociology surrounding formal methods can be found in *Mechanizing Proof: Computing Risk and Trust* [23]. Many computer security texts – e.g. *Computer Security Art and Science* [24], *Security Engineering* [25], or *Secure Computing* [26] — also contain sections on the application of formal methods to security.

Gene Spafford

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

Paul Black

It's coming out as volume 8 of Knuth's The Art of Computer Programming :-)

Summary

We thank out experts for their candidness, if they appeared to have more to say, they did; however, we've exceeded our space limitations. Now, we let you decide: are formal methods relevant to security-centric problems and if so, how can they best be incorporated into the development and delivery processes?

References

[1] C. A. R. Hoare, "How did software get so reliable without proof?," in *FME'96: Industrial Benefit and Advances in Formal Methods*, 1996, pp. 1–17.

[2] K. Havelund, M. Lowry, and J. Penix, "Formal analysis of a space-craft controller using SPIN," *Software Engineering, IEEE Transactions on*, vol. 27, no. 8, pp. 749–765, 2001.

[3] G. J. Holzmann, "Mars code," *Communications of the ACM*, vol. 57, no. 2, pp. 64–73, 2014.

[4] L. Robinson and K. N. Levitt, "Proof techniques for hierarchically structured programs," *Communications of the ACM*, vol. 20, no. 4, pp. 271–283, 1977.

[5] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, "A security model for military message systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 198–222, 1984.

[6] C. Meadows, "Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999, pp. 216–231.

[7] D. Greve, M. Wilding, and W. M. Vanfleet, "A separation kernel formal security policy," in *Proc. Fourth International Workshop on the ACL2 Theorem Prover and Its Applications*, 2003.

[8] C. L. Heitmeyer, M. M. Archer, E. I. Leonard, and J. D. McLean, "Applying formal methods to a certifiably secure software system," *Software Engineering, IEEE Transactions on*, vol.

34, no. 1, pp. 82–98, 2008.

[9] A. Gargantini and C. Heitmeyer, “Using model checking to generate tests from requirements specifications,” in *Software Engineering—ESEC/FSE’99*, 1999, pp. 146–162.

[10] W. Lobb, “Business Perspectives on Provably-Correct Software,” 2015.

[11] N. S. Bjørner, A. Browne, M. A. Colón, B. Finkbeiner, Z. Manna, H. B. Sipma, and T. E. Uribe, “Verifying temporal properties of reactive systems: A STeP tutorial,” *Formal Methods in System Design*, vol. 16, no. 3, pp. 227–270, 2000.

[12] M. Barr, “An Update on Toyota and Unintended Acceleration,” 2013. [Online]. Available: http://embeddedgurus.com/barr-code/2013/10/an-update-on-toyota-and-unintended-acceleration/. [Accessed: 09-Mar-2016]

[13] M. Heusser, “Can New Software Testing Frameworks Bring Us to Provably Correct Software?,” 2013. [Online]. Available: http://www.cio.com/article/2388410/agile-development/can-new-software-testing-frameworks-bring-us-to-provably-correct-software-.html. [Accessed: 09-Mar-2016]

[14] K. Jones, “Regulatory Analytics and Data Architecture (RADAR),” *CIFR Paper No. WP065/2015*, 2015.

[15] R. Kemmerer, C. Meadows, and J. Millen, “Three systems for cryptographic protocol analysis,” *Journal of CRYPTOLOGY*, vol. 7, no. 2, pp. 79–130, 1994.

[16] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, “A few billion lines of code later: using static analysis to find bugs in the real world,” *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[17] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, “The ASTRÉE analyzer,” in *Programming Languages and Systems*, Springer, 2005, pp. 21–30.

[18] D. Delmas and J. Souyris, “Astrée: From research to industry,” in *In Static Analysis Symposium (SAS*, 2007.

[19] T. Ball, V. Levin, and S. K. Rajamani, “A decade of software model checking with SLAM,” *Communications of the ACM*, vol. 54, no. 7, pp. 68–76, 2011.

[20] M. Harris, “Why You Shouldnâ t Worry About Liability for Self-Driving Car Accidents,” *IEEE Spectrum website*, 2015. [Online]. Available: http://spectrum.ieee.org/cars-that-think/transportation/self-driving/why-you-shouldnt-worry-about-liability-for-selfdriving-car-accidents. [Accessed: 13-Mar-2016]

[21] J. P. Bowen and M. G. Hinchey, “Ten commandments of formal methods... ten years later,” *Computer*, vol. 39, no. 1, pp. 40–48, 2006.

[22] J. R. Yost, “An Interview with John D. McLean,” 2014 [Online]. Available: http://conservancy.umn.edu/handle/11299/164989. [Accessed: 11-Mar-2016]

[23] D. MacKenzie, *Mechanizing proof: computing, risk, and trust*. MIT Press, 2004.

[24] M. Bishop, *Computer security: art and science*, vol. 200. Addison-Wesley, 2012.

[25] R. Anderson, *Security engineering*. John Wiley \& Sons, 2008.

[26] R. C. Summers, *Secure computing: threats and safeguards*. McGraw-Hill, Inc., 1997.