On Software Infrastructure: Develop, Prove, Profit?

Sean Peisert

April 22, 2023

Having *infrastructure* survive over very long stretches of time is a non-trivial task. This is because such infrastructure either needs to be built *extremely* well from the outset, or because it requires ongoing maintenance. The former may require prohibitively large initial investment. The latter requires ongoing investment from public agencies over the span of decades or centuries despite the pendulum swings of those governments from contrasting political aims. Without either the very high initial or ongoing investment, infrastructure can fail. Physical infrastructure failure is not inevitable — consider railways in Japan, the Panama Canal, and the U.S. Interstate Highway System.

Infrastructure challenges in the physical world has also led me to think more about the *software infrastructure* that facilities our society. I am not the first to refer to software as infrastructure, of course — we have long had the term *cyberinfrastructure* to encompass the numerous elements of hardware and software involved in computing, and software maintenance has been well established as a critical problem [1]. Nor am I the first to draw parallels between software development and bridge design [2].

Nonetheless, the problem persists. Ozment and Schechter's wonderful "Milk or Wine: Does Software Security Improve with Age?" [3] showed essentially that code that continues to be mostly just refined over time, like OpenBSD, ages (improves) like wine, whereas software that prioritizes adding features to refining functionality will age (sour) like milk. Of course most software vendors *do* prioritize adding features. Would today's software be even be as robust as it is today if roughly 20 years ago, Bill Gates had not issued his "Trustworthy Computing" memo?

Even aside from bugs introduced through unsound software deveopment practices, *IEEE Security & Privacy* Editorial Board members recently wrote on the subject of software supply chains and the degree to which we can trust developers to do the "right technical thing" [4] — inspired by the `node-ipc` sabotage. Massacci and Sabetta noted:

> "If past is prologue, these events might also indicate a new factor of fragility of the overall software infrastructure."

Of course, we cannot always trust developers to do the "right thing." SolarWinds caused raised the awareness of the *software supply chain* problem [5]. This perspective was further amplified with the University of Minnesota researchers' "hypocrite commit" social "experiment" on the Linux kernel maintainers.

We all rely on software developed by others, and almost all software developers rely on code libraries developed by others. Massacci and Pashchenko refer to this reliance as *technical leverage* [6]. Of course technical leverage cuts both ways — what is the risk of using someone else's code vs. producing all needed code from whole cloth, which few organizations would choose to do. So then what does limiting trust in software developers mean for software infrastructure that society relies on, and what are its implications?

Going back to physical infrastructure, experts in electric power systems tell us that the best solution for a reliable power system is not increasing central generation with nuclear reactors or prolonging the combustion of fossil fuels, but a reduced focus on fully-public power grids to focus on local generation and storage via microgrids. Experts in water infrastructure tell us that solving the water shortage problems in the Western United States is not necessarily building giant, electric power-hungry desalinization plants that emit more carbon. Instead, we must address the reality that most water *captured* in reservoirs and wells is maximally purified for human consumption, while most of the water *used* actually requires minimal purification because is not actually consumed by humans. In terms of transportation, the only reason that we keep widening freeways (Interstate 10 in Houston, Texas currently has 26 lanes in certain parts) is due to lack of civic planning.

What can we learn from the problems in our electric power, water, and transportation infrastructure, as well as the other major infrastructure successes listed earlier in this piece? Certainly sustained, ongoing maintenance can help, and in some cases are essential — for example, U.S. Interstate Highways — more on that in a bit, when we discuss public vs. private funding. Although at the same time, such solutions can also only go so far — you can only widen freeways so much.

So what is the software equivalent of microgrids, optimizing water purification choices, civic planning, and the portions of Japanese railways, the Panama Canal, U.S. Interstate Highways that are not simply dependent on ongoing maintenance? For most of these examples, the common theme is *engineering*, though looking carefully, is not simply about engineering more infrastructure but about *more efficient infrastructure*, which in some cases, may even lead to *less infrastructure* or at least infrastructure that is easier to maintain. Perhaps software infrastructure should be viewed through lenses of those themes. The failures in physical infrastructure that we have referred to surely have parallels to the software development practices that give us our "patch and pray" method of software security. This, in contrast to robust software engineering, which, at one end of the spectrum, might even include *formal verification* of full functional correctness.

How is it that software can perform so badly in comparison to other construction? Building codes require precise architectural drawings and engineering plans [7]. Even directing a motion picture should follow a shot list, as noted by filmmaker David Mamet:

> "The work of the director is the work of constructing the shot list from the script. The work on the set is nothing. All you have to do on the set is stay awake, follow your plans, help the actors be simple, and keep your sense of humor. The film is directed in the making of the shot list. The work on the set is simply to record what has been chosen to be recorded. It is the plan that makes the movie." [8]

Moreover, Mamet feels this point of *a priori* planning is so critical to directing motion pictures that he too leverages analogies to the physical world to emphasize his points:

> "Some people . . . designed and built a lot of counterculture buildings. These buildings proved unlivable. . . . They all either fell down or are falling down or should be torn down.
>
> "I live in a house that is two hundred years old. It was built with an axe, by hand, and without nails. Barring some sort of man-made catastrophe, it will be standing in another two hundred years. It was built with an understanding of, and a respect for wood, weather, and human domestic requirements.
>
> "It's very difficult to shore up something that has been done badly. You'd better do your planning up front, when you have the time. It's like working with glue. When it sets, you've used up your time. . . .
>
> "If you go up into Vermont and you build a roof with a peak, the snow will fall off. You build a flat roof, the roof will fall down from the weight of the snow—which is what happened to a lot of the countercultural architecture of the 1960s." [8]

I previously expounded upon the need for formal verification [9], drawing and leveraging analogies to Richard Feynman's report on the Space Shuttle *Challenger* accident, Ralph Nader's *Unsafe* *at Any Speed*, and Rachel Carson's *Silent Spring*, among others, to convey a moral argument about the need for substantial change on software and hardware development practices. But of course much of the infrastructure that I've referenced — roads, water — is *public*, whereas much of software infrastructure is *not* developed and maintained with public funding. Of course, one possible solution for improving software infrastructure might be a dramatic increase in public investment in maintaining that software infrastructure, though that seems unlikely to happen given the nature of the mostly-capitalist global economy. Therefore the problem becomes even harder than having infrastructure maintenance funding survive the political whiplash of varying government priorities. We rely on software infrastructure developed by private industry who seek to quickly build functional systems at low cost. This leaves us widening metaphorical freeways decade after decade instead of doing it right to begin with. This amplifies the need for any solution to be a sound *business decision* — not a moral one — in order for corporations to take on what governments cannot or will not. For better or worse, it seems unlikely that we will see the equivalent of a 1780 B.C. Code of Hammurabi for the 21st Century A.D. software industry, as described by Massacci and di Tizio, that will motivate such change [10].

In that vein, we often talk about formal verification only for its security and robustness benefits and deride formal verification for its costs, or argue, as I did previously, that the cost is worth the moral imperative. And yet, notably, Amazon and Microsoft have spoken about the judicious application of formal verification as a smart *financial decision* as a result of systematically squashing bugs in code that otherwise would have required ongoing discovery and patching:

Amazon Web Services:

> "If someone comes along and says, 'for very cheaply, I can tell you this thing does not happen,' they say, 'you have me.' But, if you just say, 'we're going to find 30% more bugs,' they say, 'great, we should find more bugs. But now I've found 30% more bugs. Do I know anything more?' Not really, from a leader's perspective." [11]
>
> "...we have found that the internal use of formal reasoning tools provides good value for the investment made. Formal reasoning provides higher levels of assurance than testing for the properties established, as it provides clear information about what has and has not been secured." [12]
>
> "... proofs of compliance controls that can be shared and checked/re-checked have the possibility to reduce the cost of compliance certification, as well as reduce the time-to-market for organizations who require certification before using systems." [12]
>
> "...the business value have typically not justified the creation of scalable and easy-to-use tools for the formal verification of those models. With the cloud, much of this has changed. ... Most importantly, since those models are utilized by a large user community, it is now economically feasible to build the tools needed to verify them." [13]

Microsoft Azure:

> "...the security evaluation included a study of historic and open security bugs that our work was aimed at addressing and closed or would have closed." [14]
>
> "There are attack surfaces exposed to input validation bugs throughout the software ecosystem, and hardening them all is a long road. We believe EverParse can help." [15]
>
> "If you're writing a format description in some C-like syntax and the proofs happen automatically behind the scenes, then the maintenance story is much easier. The verificaiton tool is just a tool as part of your compiler workflow. . . . So that's a bit of an easier sell. People don't have to be F* experts to maintain and evolve this code." [16]

These insights from Amazon and Microsoft are summarized succinctly by Fisher, Launchbury, and Richards:

> "'...the right question to ask is whether the increased effort is a good investment. This question hinges not only on the development cost but also on the cost of flaws in software." [17]

Clearly, Amazon and Microsoft have found that in at least some cases, the cost of flaws in the software exceeds the cost of formal verification.

Formal verification obviously is not appropriate for all software. These business success stories stem in large part from their selective application and also the scope and scale of the use of the AWS and Azure public clouds. Even so, should we believe that there are only two companies in the world capable of deploying such approaches in their infrastructure? Other very, very large companies with large use of technology could also conceivably deploy such approaches as well. Major financial firms, electronic health record vendors, automotive companies, pharmaceutical companies, and so on. As such, what if the solution actually does significantly involve an increase in the use of formal verification to produce core components that would satisfy society's needs for secure software infrastructure?

Memory safety, for example, as Jaeger recently discussed [2], and which will also be the topic of a special issue of *S&P* in 2024, could be a particularly valuable application of formal verification. Formal verification of memory safety, or its enforcement, could provide additional carrots that could entice developers toward comprehensive memory safety. Indeed having elements formally verified as being memory safe could eventually become so ubiquitously used that their presence even becomes a stick to developers who fail to leverage such approaches.

Formal verification does have properties that are particularly useful for sustaining software over long periods of time. One might imagine software elements and services that have been made simple enough that the implementation can be formally verified against an open and understandable specification, and which would underlie significant parts of our critical software infrastructure. Initially, significant levels of effort will be required to produce the proofs. Subsequently, when the specification and proof are available, then *any implementation can be automatically checked against the specification.* Even for the initial proving, there is significant reason for optimism of automation in computer-aided verification, which is improving at a rapid pace [18]. Concrete examples of recent formal verification success stories include Google's BoringSSL, AWS's s2n-tls and LibCrypto, and Microsoft's Everparse and Evercrypt, among others. DARPA's recently-announced *Pipeline Reasoning of Verifiers Enabling Robust Systems (PROVERS)* program should continue to advance the state of the art to produce such components.

The notion of a significant increase in formal verification shows tremendous promise for hardening key elements of crumbling software infrastructure. As time is not on our side against adversaries that seek to disrupt software systems, we need to identify ways to dramatically accelerate that progress. While Amazon, Google, and Microsoft all have shown success in the application of formal verification, and their successes can have broad effect for anyone using the software resulting from that application, those successes are a mere prologue to what is necessary.

At the same time, altruism and moral arguments are clearly inadequate. Unless there is a dramatic change in the roles of the public sector in software infrastructure, we are left with for-profit companies, and for them, only money talks. Perhaps *sticks* (liability) will be one path forward as suggested both by Jaeger [2] and also, recently, in the Biden-Harris National Cybersecurity Strategy [19]. But *carrots* (profit) may be another. How to tempt the private sector with these carrots? While Amazon and Microsoft have shown business success in the appropriate application of formal verification, the financial details of their efforts are not public and we need to show evidence to more of the entities capable of taking on the task of making publicly used software infrastructure substantially more robust.

My charge to the software security community is to embark on a large and serious study on the financial arguments involved with formal verification to produce publicly available cost data around the use of formal verification at much larger scales and in a way that would help support and sustain robustness and security of at the most critical elements our society's software infrastructure. These studies could refute the business arguments that Amazon and Microsoft have made. Or, they might support what Amazon and Microsoft have found, and then the data and results of those studies might be leveraged by software development organizations throughout the world as the evidence needed to seriously consider significant expansion of the use of formal verification in the development and configuration of software underlying critical aspects of society's functioning. Let's provide the evidence to justify constructing software's Panama Canal.

References

[1] Andrew Russell and Lee Vinsel. Hail the maintainers. *Aeon*, April 7, 2016.

[2] Trent Jaeger. On Bridges and Software. *IEEE Security & Privacy*, 21(02), Mar./Apr. 2023.

[3] Andy Ozment and Stuart E. Schechter. Milk or Wine: Does Software Security Improve with Age. In *Proceedings of the 15th USENIX Security Symposium*, 2006.

[4] Fabio Massacci, Antonino Sabetta, Jelena Mirkovic, Toby Murray, Hamed Okhravi, Mohammad Mannan, Anderson Rocha, Eric Bodden, and Daniel E. Geer Jr. "Free" as in Freedom to Protest? *IEEE Security & Privacy*, 20(05):16–21, 2022.

[5] Sean Peisert, Bruce Schneier, Hamed Okhravi, Fabio Massacci, Terry Benzel, Carl Landwehr, Mohammad Mannan, Jelena Mirkovic, Atul Prakash, and James Bret Michael. Perspectives on the SolarWinds Incident. *IEEE Security & Privacy*, 19(02):7–13, 2021.

[6] Fabio Massacci and Ivan Pashchenko. Technical Leverage: Dependencies Are a Mixed Blessing. *IEEE Security & Privacy*, 19(3):58–62, 2021.

[7] Carl E. Landwehr. A Building Code for Building Code: Putting What We Know Works to Work. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC)*, pages 139–147, 2013.

[8] David Mamet. *On Directing Film.* Penguin, 1991.

[9] Sean Peisert. Unsafe at Any Clock Speed: the Insecurity of Computer System Design, Implementation, and Operation. *IEEE Security and Privacy*, 20(1):4–9, Jan./Feb. 2022.

[10] Fabio Massacci and Giorgio di Tizio. Are Software Updates Useless against Advanced Persistent Threats? *Communications of the ACM*, 66(1):31–33, 2022.

[11] Byron Cook. On the Business of Proof (Interview). *Workshop on Dependable and Secure Software Systems* — `https://www.youtube.com/watch?v=g-DH_b5bFd4`, 2021.

[12] Byron Cook. Formal Reasoning About the Security of Amazon Web Services. In *Proceedings of the International Conference on Computer Aided Verification*, pages 38–47. Springer, 2018.

[13] John Backes, Pauline Bolignano, Byron Cook, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Martin Schaef, Cole Schlesinger, Rima Tanash, Carsten Varming, and Michael Whalen. One-Click Formal Methods. *IEEE Software*, 36(6):61–65, 2019.

[14] Nikhil Swamy, Tahina Ramananandro, Aseem Rastogi, Irina Spiridonova, Haobin Ni, Dmitry Malloy, Juan Vazquez, Michael Tang, Omar Cardona, and Arti Gupta. Hardening Attack Surfaces with Formally Proven Binary Format Parsers. In *Proceedings of the 43rd ACM Conference on Programming Language Design and Implementation*, June 2022.

[15] Tahina Ramananandro, Aseem Rastogi, and Nikhil Swamy. EverParse: Hardening Critical Attack Surfaces with Formally Proven Message Parsers. `https://www.microsoft.com/en-us/research/blog/everparse-hardening-critical-attack-surfaces-with-formally-proven-message-parsers/`, May 3, 2021.

[16] Nikhil Swamy (with Joey Dodds and Shpat Morina (interviewers)). Episode #21: Fully In Bed With Dependent Types (Interview). *Building Better Systems Podcast* — `https://www.youtube.com/watch?v=ezoAtruDURY`, 2022.

[17] Kathleen Fisher, John Launchbury, and Raymond Richards. The HACMS Program: Using Formal Methods to Eliminate Exploitable Bugs. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104):20150401, 2017.

[18] William Martin, Patrick Lincoln, and William Scherlis. Formal Methods at Scale. *IEEE Security & Privacy*, 20(3):22–23, May/June 2022.

[19] The White House. National Cybersecurity Strategy. `https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf`, March 2023.