# Unsafe at Any Clock Speed: the Insecurity of Computer System Design, Implementation, and Operation

Sean Peisert

November 10, 2021

> "It appears that there are enormous differences of opinion as to the probability of a [system failure]. The higher figures come from the working engineers, and the very low figures from management. When playing Russian roulette the fact that the first shot got off safely is little comfort for the next. [T]here have been recent suggestions by management to curtail elaborate and expensive tests as being unnecessary. This must be resisted. The proper way to save money is to curtail the number of requested changes, not the quality of testing for each.
>
> "Let us make recommendations to ensure that [management deals] in a world of reality in understanding technological weaknesses and imperfections well enough to be actively trying to eliminate them. They must live in reality in comparing the costs and utility. Only realistic schedules should be proposed, schedules that have a reasonable chance of being met. If in this way support [would not exist], then so be it. For a successful technology, reality must take precedence over public relations, for nature cannot be fooled." *(Author's Note: ellipses omitted for readability.)*

One could be forgiven for thinking that this text came from a critique of SolarWinds Orion, Adobe Flash, or Microsoft Office or Internet Explorer, or from a recent report that led to a set of strong recommendations contained in a recent White House Executive Order on Cybersecurity. As most readers of this magazine likely recognize, that would be wrong, as this is of course excerpted and edited text written by Richard Feynman, in his appendix to the 1986 Rogers Commission report studying the Challenger disaster, "Personal Observations on Reliability of Shuttle." [1]

I quoted portions of Feynman's report here because I believe that we have a similar problem in computer software for similar reasons: companies developing software prioritize maximum shareholder profit and productivity over software safety, robustness, and security. It is not unreasonable or unexpected that companies prioritize profit. At the same time, many companies have embraced "corporate social responsibility," having recognized that supporting employees, customers, and the broader world can positively impact both reputation and profit. As just one example, we see healthcare organizations balance profit with patient safety, because not doing so would lead to public outrage, which in turn would impact profits. However, with only rare exceptions do we see a similar effort to balance shareholder primacy with software security. The consequences of this lack of balance range from events like the major breaches, ransomware, and attacks against critical systems like hospitals and utilities — the NotPetya attacks affecting Maersk's shipping and port operations worldwide, the WannaCry ransomware attacks against U.K. National Health Service hospitals, and the Colonial Pipeline attack in the U.S.

So where *is* the public outrage? And how did we get to this state, and why it is acceptable to so many organizations to live with this level of vulnerability and compromise? These incidents are not mere annoyances. Real people are affected in real ways. Given this, how is it possible that this is not a virtually identical moment to automobile safety before Ralph Nader's *Unsafe at Any Speed* [2] demonstrated the need for and barriers to mandating safety improvements in cars, and led directly to seat belts and other safety advances? Or public and agricultural safety before Rachel Carson's *Silent Spring* [3] exposed the toxicity of the chemical DDT and led directly to its ban? Or medical safety before John Snow's *On the Mode of Communication of Cholera* [4] exposing that germs, not "miasma," cause disease, which led directly to water safety and sewage improvements in London and beyond? Or the Flexner Report's [5] impact on bringing mainstream scientific protocols to medical education? Or the Institute of Medicine's *To Err is Human* [6] exposing that the same number of daily deaths from medical errors in the United States is equivalent to the number of deaths from a jumbo jet crashing each day, leading directly to a fundamental change in the approach to quality of care, and the renaming of an agency to the "U.S. Agency for Healthcare Research and Quality"? It is an inconvenient truth that software and hardware engineers make mistakes, those mistakes can become "bugs," some of those bugs represent vulnerabilities that can be attacked, and that, at times that are unpredictable, some of those vulnerabilities are attacked. So where is the equivalent response for software quality?

In fact, the reason for this situation is essentially identical as what Feynman indicated more than three decades ago: profit, expediency, and succumbing to the requests for "changes" (ususally "features"). The answer as to why there isn't public outrage surely cannot be because we accept that shareholder profit should be prioritized over software quality. In fact I would argue that it even does a disservice in the long term to shareholder value to prioritize short-term profit over software quality. At some point, companies that allow enough vulnerability will see the impact in their profits. At the same time, it isn't like we haven't advocated substantially more secure systems, and even "clean slate" solutions before — certainly, with Multics [7] and the aspirations for Orange Book A1-certified computer systems [8], there were goals to meet provably secure operational requirements. Indeed, 46 years ago, in 1974, Karger and Schell pointed out "Multics is not Now Secure" but went on to suggest essentially that it *could* be made secure if we worked just a little bit harder [9]. However, writing in 2002 on their observations in the 28 years since the original paper, they note [10]:

> "In the nearly thirty years since the report, it has been demonstrated that the technology direction that was speculative at the time can actually be implemented and provides an effective solution to the problem of malicious software employed by well-motivated professionals. Unfortunately, the mainstream products of major vendors largely ignore these demonstrated technologies."

A decade later, beginning in 2012, two DARPA programs run by Howie Shrobe, "Clean-slate design of Resilient, Adaptive, Secure Hosts" (CRASH) and "Mission-Oriented Resilient Clouds" (MRC) [11] sought to draw inspiration from "visionary ideas of the past" to develop and demonstrate secure and resilient systems. The "Turtles All the Way Down" piece that my colleagues Matt Bishop, Ed Talbot and I wrote in 2012, advocated building and rebuilding systems with pervasive use of formal methods, diversity, and Byzantine fault tolerance [12] "from atoms to eyeballs" in a 13-level stack.

Fast forward to this past year when Paul van Oorschot noted in this magazine that the C language lacks type and memory safety, "...having learned our lesson from 45 years of use, surely we do not still use C in new projects and in building brand new systems, do we? As it turns out, the evidence suggests we do." [13] Van Oorschot continued, noting that in the past, even though type-safe languages are available for use, such as Java, Go, and Apple's Swift, the fact that those languages have not been appropriate for systems development may have prolonged the use of C and C++. As van Oorschot writes, performance languages appropriate for systems work now exist, but perhaps it will take something like requirements for government procurement to see languages like Rust adopted at scale. (As a side note, it is insufficient to leverage type safe languages if the runtimes for those languages are also written in C/C++, as the runtimes for Java and Ruby are, for example.) The wonderful "Cyber Moonshot" piece in the very next issue of *S&P* by Hamed Okhravi, also advocates the use of semantically-rich processors, type and memory-safe systems languages, and fine-grained operating system compartmentalization [14].

It is probably unreasonable to expect that these examples that I have given of attempts to radically improve computer security would have the effects of the clarion calls in *Silent Spring* or *Unsafe at Any Speed* — both books specifically aimed at the general public. However, scholarly writings in the medical domain, including *On the Mode of Communication*, *To Err is Human*, and the Flexner Report, *have* been transformative, whereas despite 46 years of efforts, from Karger and Schell to the present day, I don't believe that we've seen similar effects in transforing computer security.

What I believe has changed since Karger and Schell, and perhaps even since the DARPA CRASH and MRC programs is that technology and techniques have improved to the point that *we are now finally at a place where we can actually, practically do something about this situation.* In fact, in the same Challenger report, Feynman again even gave us a portion of the solutions — bottom-up engineering:

> "The software is checked very carefully in a bottom-up fashion. First, each new line of code is checked, then sections of code or modules with special functions are verified. The scope is increased step by step until the new changes are incorporated into a complete system and checked. But completely independently there is an independent verification group, that takes an adversary attitude to the software development group, and tests and verifies the software as if it were a customer of the delivered product. A discovery of an error during verification testing is considered very serious, and its origin studied very carefully to avoid such mistakes in the future. The principle that is followed is that all the verification is a test of that safety, in a non-catastrophic verification. A failure here generates considerable concern." *(Author's Note: ellipses omitted for readability.)*

Yet, regardless of the actual approach — top-down, bottom-up, or some combination of the two — in the past, we have found Feynman's prescription regarding the degree of assurance required utterly untenable for all but the most critical systems. Times have changed in at least two ways: one is that we have gone from a world in which computer-controlled systems were mostly only running commercial and military aircraft and NASA's Space Shuttles. to a world in which dozens or hundreds of processors exist in the modern automobile, building "control systems," and numerous other domains in life in which humans are dependent. A second and vital change is that technology useful for safety and security has advanced profoundly in the past 25 years since the Rogers Commission report was released. Let's take a look at some of those advances:

Consider type-safe languages: Buffer overruns have been the "most dangerous" sortware weakness for years. Why should the public put up with something that is exposed as public enemy #1 year after year with little progress? In contrast, Rust has emerged as a type and memory-safe language suitable for systems programming. Mozilla's Servo browser engine is being written in the Rust, and numerous Linux libraries and utilities are being rewritten in Rust. Rewriting old code in Rust can be a tough sell although Google's recent effort to implement site isolation in Chrome, and

Mozilla's development and application of RLBox to Firefox — both significant manual efforts — show progress can be made when the needed resources are devoted. This will also become easier as more third party libraries are developed for Rust and more new developers learn Rust in computer science courses.

Consider formal methods today: There exist many software elements that underlie the modern Internet and its usage that have been revealed as substantially lacking in security rigor for years, such as the vulnerabilities that plagued OpenSSL until organizations like Google, Microsoft, and OpenBSD stepped in. Why is it that the public is so forgiving of the reliance on such blatantly problematic software by major companies? And, many other examples of such software certainly still remain. In contrast, seL4 is a formally-verified microkernel, CertiKOS is a formally verified kernel, the Linux KVM hypervisor has been formally verified, and DARPA's "Little Bird" is a formally verified autonomous helicopter, having survived hacking contests as part of the DARPA HACMS program [15], run by Kathleen Fisher, John Launchbury, and Raymond Richards, in 2017, and again at DEFCON this past year. In addition, numerous key elements of Amazon Web Services have been formally verified, Facebook leverages the Infer system to continuously verify code, and Microsoft's Project Everest is developing a formally verified stack to improve secure web communications. Not every formal verification is as useful as another and it may never be tenable to formally verify all code, but the DARPA exercises alone seem to have demonstrated considerable value. At the very least, there is strong evidence that building systems on top of formally-verified elements that are now available and usable could substantially ameliorate a large swath of security problems. Having even more verified systems that provide support for additional functionality would help encourage broader adoption of assured systems.

Consider security-enhanced hardware today: as discussed earlier, security weaknesses often result from the use of "unsafe" languages and shared infrastructure. In contrast, University of Cambridge and SRI's CHERI [16] provides a capability-based system that provides fine-grained memory protection and software compartmentalization, thereby protecting against a host of weaknesses exposed by the use of unsafe languages, code injection attacks, and more. This is particularly valuable protection when existing software cannot easily be rewritten in type and memory safe languages, for example, due to the vast amount of existing libraries written in C/C++. CHERI also now has numerous formally-verified elements. In addition, Arm's forthcoming CHERI-extended Morello prototype CPU, system-on-a-chip, and board will ship early next year, and will consist of a full industrial quality and high-performance adaptation of Arm's Neoverse N1 CPU design. This prototype is in fact the culmination of a kind of "moonshot" that has been developed over 10 years and with $250 million of DARPA, United Kingdom government, and in-kind industry funding, and seems like it could serve as a model for advancing other security techniques and technologies.

In addition to capability-enhanced hardware, consider hardware trusted execution environments (TEEs). Running on traditional servers, including those in the cloud, requires complete trust of the system administrator as well as the numerous levels of the stack that seek to mitigate attempts by one user to attack another. Who is really happy about putting complete and unquestioning trust regarding data and computation in giant corporations? In contrast, TEEs provide strong isolation properties, sometimes even from system administrators and physical attacks. they are available or announced from every major CPU platform, including AMD's SEV; ARM's v9's Confidential Compute Architecture, and Intel's SGX, alongside open source TEEs, such as the RISC-V-based Keystone. Further, some form of TEE-like "confidential computing" service is available from the three major commercial cloud providers: AWS Nitro Enclaves, GCP Confidential Computing, and Azure Confidential Computing. The Linux Foundation also hosts the Confidential Computing Consortium (CCC). The cloud and community efforts provide the software model and cryptographic infrastructure to make the use of confidential computing more straighforward. For usability and performance reasons, not all of these architectures are useful for general-purpose computing. However, for certain workloads running on single nodes, SEV can carry little overhead beyond that of virtualization itself, and is readily available in cloud environments.

The reluctance of organizations to adopt some of these techniques and technologies has echoes of the White Queen informing Alice about the (lack of) availability of jam today [17]. However, despite past failures to make significant progress toward securing systems via Multics and the Orange Book, all of this recent progress has shown what *is* possible with the tools that we have *today.* Organizations can use type and memory safety (Rust), formally verified components (seL4, CertiKOS, Linux KVM), and obtain strong hardware isolation (AMD's SEV) *today.* At least in the case of the Rust language as well as cloud environments that have broad frameworks supporting confidential computing on top of AMD's SEV and related technologies, this can entail little extra effort. Organizations can use Facebook's Infer automated reasoning system for static analysis *today.* Prototype hardware supporting CHERI will be available roughly at the time this piece goes to press, and may well be in broader production in not too many more years.

I've enumerated a non-exhaustive list of numerous techniques and technologies here that could all represent elements of this improvement I speak of. Not all will be part of the final solutions, and undoubtedly there are others that I haven't covered, such as the automated verification tools available that showed such success during DARPA's Cyber Grand Challenge [18] and advances in the "grand challenges" of user-centered security that make it harder for users to make decisions in a way that will lead to security failures [19]. Further, the solutions that I have discussed will also not solve all problems, and will not be adopted everywhere — for example, consider all the software written by "citizen developers" or are outsourced to the lowest bidder. But if enough of the important software leverages these solutions, it would seem that doing so could solve a substantial number of problems, thereby enabling security researchers and engineers to focus on the problems for which we do not yet have solutions.

Portions of this cybersecurity vision likely are a "moonshot." But I think it would be a misrepresentation to characterize the entire endeavor as such. So, what's in the way? We've already pointed to cost, so how do we lower that cost or overcome that barrier? Existing public sentiment about every new security breach that takes place clearly hasn't been enough. Perhaps the public has just been convinced it has no choice but to accept the status quo. In contrast, I think the public has a right to be outraged about computer security. Further, even though the government itself suffers computer security failures large and small on an ongoing basis, the appetite for significant regulation (e.g., liability for insecure software, substantially increasing requirements for software and hardware security in government procurement) in this space seems not to exist. Thus, in the face of evidence that there are in many cases relatively low bars to much safer systems, the reason for the continued prevalence of low adoption of the components that would could systems much safer remains something of a mystery, given that numerous key components are here now, and the rest may well not be that far in the future from being deployed.

*The barriers to large-scale adoption of emerging security techniques and technologies urgently need to be investigated.* This investigation should include a focus on technical issues, but should also include experts who can illuminate usability, education, economic, policy, and social issues, and other systematic barriers to technology transition for innovation. At least on a technical level, there are few excuses not to be embracing many of the approaches that I've illustrated here. There are few excuses for not writing most or all new systems code in Rust; for systems, where appropriate, to be built using verified components and/or on top of security-enhanced hardware, and for applications to be run on those systems wherever possible; and for the most important source code to leverage modern, automated program verification tools and possibly formal methods.

In another passage from their 2002 piece, Karger and Schell [10] write:

> "In our opinion this is an unstable state of affairs. It is unthinkable that another thirty years will go by without one of two occurrences: either there will be horrific cyber disasters that will deprive society of much of the value computers can provide, or the available technology will be delivered, and hopefully enhanced, in products that provide effective security. We hope it will be the latter."

Computer systems and networks have become "unsafe at any speed." The time to change that is now. The future is here. There is no further room for excuse, ignorance of reality, or fooling of nature.

## References

[1] Richard P. Feynman. The Presidential Commission on the Space Shuttle Challenger Accident Report, Volume 2, Appendix F: "Personal Observations on the Reliability of the Shuttle", June 6, 1986.

[2] Ralph Nader. *Unsafe at Any Speed: The Designed-In Dangers of the American Automobile.* Grossman Publishers, 1965.

[3] Rachel Carson. *Silent Spring.* Houghton Mifflin, 1962.

[4] John Snow. *On the Mode of Communication of Cholera.* John Churchill, London, 1855.

[5] Abraham Flexner. Medical Education in the United States and Canada: A Report to the Carnegie Foundation for the Advancement of Teaching. *Bulletin of the Carnegie Foundation for the Advancement of Teaching*, Bulletin Number 4, 1910.

[6] Institute of Medicine. *To Err Is Human: Building a Safer Health System.* National Academies Press, Washington, D.C., 2000.

[7] Elliot Organick. *The Multics System: An Examination of Its Structure.* MIT Press, Boston, MA, USA, 1972.

[8] Department of Defense. Trusted Computer System Evaluation Criteria ("Orange Book"). Technical Report DoD 5200.28-STD, December 26, 1985.

[9] Paul A. Karger and Roger R. Schell. Multics Security Evaluation: Vulnerability Analysis. HQ Electronic Systems Division: Hanscom AFB, MA. URL: `http://csrc.nist.gov/publications/history/karg74.pdf`, 1974.

[10] Paul A. Karger and Roger R. Schell. Thirty Years Later: Lessons from the Multics Security Evaluation. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, pages 119–126, 2002.

[11] Howard Shrobe and Daniel Adams. Suppose We Got a Do-Over: A Revolution for Secure Computing. *IEEE Security & Privacy*, 10(6):36–39, Nov/Dec 2012.

[12] Sean Peisert, Ed Talbot, and Matt Bishop. Turtles All The Way Down: A Clean-Slate, Ground-Up, First-Principles Approach to Secure Systems. In *Proceedings of the 2012 New Security Paradigms Workshop (NSPW)*, pages 15–26, Bertinoro, Italy, September 19–21, 2012.

[13] Paul C. van Oorschot. Toward Unseating the Unsafe C Programming Language. *IEEE Security & Privacy*, 19(2):4–6, Mar/Apr 2021.

[14] Hamed Okhravi. A Cybersecurity Moonshot. *IEEE Security & Privacy*, 19(3):8–16, May/June 2021.

[15] 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.

[16] Robert N. M. Watson, Jonathan Woodruff, Peter G Neumann, Simon W Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, pages 20–37, 2015.

[17] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. “The rule is, jam to-morrow and jam yesterday — but never jam to-day ... It’s jam every other day: to-day isn’t any *other* day, you know.”. Macmillan, London, 1872.

[18] Mike Walker. Machine vs. Machine: Lessons from the First Year of Cyber Grand Challenge. *24th USENIX Security Symposium*, August 12, 2015.

[19] Mary Ellen Zurko. User-Centered Security: Stepping up to the Grand Challenge. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, 2005.