LAWFARE
HARD NATIONAL SECURITY CHOICES

SECURITY BY DESIGN

Standards for Software Liability: Focus on the Product for Liability, Focus on the Process for Safe Harbor

Jim Dempsey*

January 2024

A workable standard for liability would include a rules-based floor and a process-based safe harbor; none of the existing frameworks for secure software development is sufficiently definitive, but the elements of floor and ceiling are readily at hand.

INTRODUCTION

In calling for legal reform that imposes liability on the developers of insecure software,[1] the Biden administration has taken on a very hard issue—what Paul Rosenzweig dubbed "the third rail of cybersecurity policy."[2] Because there is general agreement that the manufacturers of software should not be made insurers of their products but rather should be liable only when a product is unreasonably insecure, getting software liability right turns a lot on defining a standard of care. To borrow a point made decades ago by tort law giant Aaron Twerski about products liability in general, if the intent is not to impose liability for all flaws but only for some especially consequential ones, "it becomes clear that some external standard must be used."[3]

I use the word standard in this paper in two senses, referring both to the legal concept of a standard of care and to technical standards (broadly defined to include frameworks and guidelines), such as those adopted by the National Institute of Standards and Technology (NIST)

***Jim Dempsey** is a lecturer at the UC Berkeley Law School and a senior policy advisor at the Stanford Program on Geopolitics, Technology and Governance. From 2012-2017, he served as a member of the Privacy and Civil Liberties Oversight Board. He is the author of Cybersecurity Law Fundamentals (IAPP, 2021).

[1] The White House, *National Cybersecurity Strategy* (March 2023), https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf. Strategic objective 3.3 states, "The Administration will work with Congress and the private sector to develop legislation establishing liability for software products and services."

[2] Paul Rosenzweig, "The Cyber Liability Fight Begins," *Lawfare*, Jan. 6, 2023, https://www.lawfaremedia.org/article/cyber-liability-fight-begins.

[3] Aaron D. Twerski, "From Defect to Cause to Comparative Fault: Rethinking Some Product Liability Concepts," *Marquette Law Review* 60 (1977): 297, https://scholarship.law.marquette.edu/cgi/viewcontent.cgi?article=2164&context=mulr.

or private-sector entities. I use the phrase "standard of care" to refer to defined rules of software design noncompliance with which, when it causes legally actionable harm, will generate liability on the software developer for that harm.[4] One goal of this piece is to invoke the experience in many other fields of basing legal standards of liability on technical standards, such as building codes.

If developers of software are to be held responsible for the harm caused by defects in their products, we cannot risk the impact on innovation that would result from a lack of clarity as to the standard of care. Nor, given the urgency of the cybersecurity threat, can we afford to proceed at the pace of common law, with its incremental and often inconsistent articulation by judges across many cases over many years. To ensure timely progress, to reduce the costs of litigation, and to promote resource allocation to engineers rather than lawyers, we need a standard of care that is objectively measurable. To get there, I propose federal legislation that would be implemented by regulatory action drawing upon real-world observations of common and routinely exploited software flaws plus technical standards for secure software development.

My proposal is for a three-part definition of liability:

1. A rules-based approach would define a floor—the minimum legal standard of care for software—focused on specific product features or behaviors to be included or avoided.
2. However, a list of known coding weaknesses cannot suffice alone. Software is so complex and dynamic that a liability regime also needs to cover design flaws that are not so easily boiled down. For these, I propose a standard based on the defects analysis common to products liability law.
3. But this liability should not be unlimited or unpredictable. As the Biden administration's National Cybersecurity Strategy recognizes, developers deserve a safe harbor that shields them from liability for hard-to-detect flaws above the floor. For that, I would turn to a set of robust coding practices.

I thus bridge the distinction in traditional legal discourse between open-ended "standards" and bright-line "rules,"[5] arguing for a standard of care that combines bright-line rules and open-ended processes, the first based on real-world evidence, the latter based on technical standards.

Section 1 of this paper sets the stage by briefly describing the problem to be solved. Section 2 canvasses the different fields of law (warranty, negligence, products liability, and certification) that could provide a starting point for what would have to be legislative action establishing a system of software liability. The conclusion is that all of these fields would face the same

[4] I adopt and adapt here a phrasing provided to me by Fordham law professor and torts expert Ben Zipursky, when he was responding in an email to a draft of this paper.

[5] Pierre J. Schlag, "Rules and Standards," *UCLA Law Review* 33 (1985): 379, https://lawweb.colorado.edu/profiles/pubpdfs/schlag/schlagUCLALR.pdf.

question: How buggy is too buggy? Section 3 explains why existing software development frameworks do not provide a sufficiently definitive basis for legal liability. They focus on process, while a liability regime should begin with a focus on the product—that is, on outcomes. Expanding on the idea of building codes for building code,[6] Section 4 shows some examples of product-focused standards from other fields. Section 5 notes that already there have been definitive expressions of software defects that can be drawn together to form the minimum legal standard of security. It specifically calls out the list of common software weaknesses tracked by the MITRE Corporation under a government contract.[7] Section 6 considers how to define flaws above the minimum floor and how to limit that liability with a safe harbor.

SECTION 1: THE PROBLEM TO BE SOLVED

"MICROSOFT, AND OUR AFFILIATES, RESELLERS, DISTRIBUTORS, AND VENDORS, MAKE NO WARRANTIES, EXPRESS OR IMPLIED, GUARANTEES OR CONDITIONS WITH RESPECT TO YOUR USE OF THE SERVICES. YOU UNDERSTAND THAT USE OF THE SERVICES IS AT YOUR OWN RISK AND THAT WE PROVIDE THE SERVICES ON AN 'AS IS' BASIS 'WITH ALL FAULTS' AND 'AS AVAILABLE.' YOU BEAR THE ENTIRE RISK OF USING THE SERVICES. … WE DO NOT GUARANTEE THE SERVICES WILL BE … SECURE, OR ERROR-FREE OR THAT CONTENT LOSS WON'T OCCUR."[8]

Language like this appears in almost every contract for software today. With such disclaimers, and due to other barriers under common law doctrines of torts or contract, software developers have been insulated from responsibility for the defects in their products—defects that are one source of the abysmal state of cybersecurity today and literally billions[9] in annual losses by businesses and individuals. It is within this context that the Biden administration promised in its cybersecurity strategy to work with Congress and the private sector to develop legislation establishing liability for software products and services.

In this paper, I focus on defining a standard of care to be established pursuant to federal legislation and applied in private litigation. An alternative or complementary approach (which the strategy does not foreclose) would rely on enforcement by regulatory agencies, which are not bound by software licensing terms. That too, however, would incentivize timely progress and provide clarity only if it defined a standard of care.

---

[6] Carl Landwehr, "We Need a Building Code for Building Code," *Communications of the ACM* 58 (2015): 24. See also C.E. Landwehr, "Building Code for Building Code: Putting What We Know Works to Work," *Proceedings of the 29th Annual Computer Security Applications Conference* (2013), http://www.landwehr.org/2013-12-cl-acsac-essay-bc.pdf (calling on the technical community, working with industry, to create a building code for critical infrastructure system software that could be adopted by industry and "perhaps eventually given legal force through government adoption").

[7] MITRE Corp., "CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/.

[8] Microsoft, *Microsoft Services Agreement* (published: July 30, 2023; effective: Sept. 30, 2023), https://www.microsoft.com/en-us/servicesagreement#12_Warranties.

[9] Federal Bureau of Investigation, *Internet Crime Report 2022* (2023), https://www.ic3.gov/Media/PDF/AnnualReport/2022_IC3Report.pdf.

SECTION 2: WARRANTY VS. NEGLIGENCE VS. PRODUCTS LIABILITY VS. CERTIFICATION: THE QUESTION IS THE SAME

To achieve the Biden administration's goal through private litigation, there are many plausible theories of liability—traditional legal frameworks that could serve as the starting point for crafting a software standard of care. At the core of all of them is an important question: How do we distinguish software that is too insecure from software that is secure enough? After all, perfectly secure software is not the goal; it is widely accepted that perfectly secure software would be unusable, or too expensive, or both. Nor is the goal to hold developers liable for every flaw. The goal, instead, is to incentivize the development of more secure software than we're getting today, across both less sophisticated and more sophisticated developers.

One legal framework that software liability might fit into is warranty. Under the common law, every sale of a product carries an implied warranty of merchantability—a merchant's basic promise that the goods sold will do what they are supposed to do and that there is nothing significantly wrong with them.[10] And if the seller states to its customers that a product can be used for some specific purpose, there arises an implied warranty of fitness for that purpose. Both types of warranty implied by common law—merchantability and fitness for particular purpose—were codified in Article 2 of the Uniform Commercial Code (UCC),[11] which has been adopted in every state but Louisiana. The current lack of legal responsibility for dangerously faulty software is due in part to the fact that software developers routinely disclaim these warranties, as they are permitted to do under both the common law and the UCC.

So one approach to software liability is to make disclaimers of the implied warranties ineffective. There is precedent for this. Massachusetts, for example, has by statute made unenforceable any attempt by a seller or manufacturer of consumer goods and services to exclude or modify any implied warranties of merchantability and fitness for a particular purpose or to exclude or modify the consumer's remedies for breach of those warranties.[12] And there is also precedent for federalizing aspects of warranty law. In 1975, Congress adopted the Magnuson-Moss Warranty Act. At its core is some language that might serve as the basis for a software liability regime: "No supplier may disclaim or modify … any implied warranty … if … such supplier makes any written warranty. … A disclaimer, modification, or limitation made in violation of this section shall be ineffective for purposes of this chapter and State law."[13]

There are some big hurdles to this approach, starting with the fact that the implied warranties traditionally apply only to the "sale" of "goods." Today, most software is not sold, but licensed,

[10] Federal Trade Commission, *Businessperson's Guide to Federal Warranty Law,* https://www.ftc.gov/business-guidance/resources/businesspersons-guide-federal-warranty-law.

[11] U.C.C. § 2-314 (merchantability), https://www.law.cornell.edu/ucc/2/2-314, and U.C.C. § 2-315 (fitness for particular purpose), https://www.law.cornell.edu/ucc/2/2-315.

[12] Massachusetts General Laws ch. 106, § 2-316A ("Limitation on Exclusion or Modification of Warranties"), https://malegislature.gov/Laws/GeneralLaws/PartI/TitleXV/Chapter106/Article2/Section2-316A.

[13] 15 U.S.C. § 2308.

and Article 2 of the UCC does not cover licenses. As to whether software is not a good but rather a service (and thus outside the UCC), that has been the subject of much confusion in the courts,[14] a confusion surely compounded over the past decade by the rapid shift to cloud computing and software-as-a-service. So, as Michael Rustad has long argued, a warranty-based approach would first have to make it clear that there are warranties, implied or expressly mandated, in the licensing of software, and, like the law in Massachusetts, it would have to prohibit efforts to disclaim them.[15]

Another approach is tort law, where common law doctrines of negligence and products liability provide possible frameworks. Trey Herr and colleagues at the Atlantic Council argue for the former.[16] In a negligence framework, they argue, "following secure development and vulnerability-management practices would comprise a negligence standard. Such a clear definition of liability would shift vendor behavior by holding parties accountable for failing to meet baseline expectations." To get there, Congress would need to act, either by creating a private right of action or by empowering a federal regulator to bring enforcement actions.

By contrast, Chinmayi Sharma and Benjamin Zipursky make a strong case for products liability law, specifically for its "design defect" prong, which focuses on whether the product was sold "in a defective condition unreasonably dangerous to the user or consumer or to his property."[17] As Sharma and Zipursky acknowledge, this approach faces hurdles too: Legislation would have to stipulate that software is to be treated as a product and that liability waivers are unenforceable. Also, the economic loss rule of many states, which prohibits recovery in tort for damage to one's business and other nonphysical harms, would have to be abrogated.

Yet another approach is certification or licensing. Europe is proceeding along these lines, with a voluntary certification process[18] that, in its complexity, is far beyond anything the U.S. has the stomach for. On a more limited scale, under President Biden's Executive Order 14028,[19] the

---

[14] Robert Dube, "So Good It's a Service: The Changing Legal Perspective on Computer Software," Eckland & Blando, Nov. 17, 2021, https://www.ecklandblando.com/blog/2021/11/so-good-its-a-service-the-changing-legal-perspective-on-computer-software/.

[15] Michael L. Rustad and Elif Kavusturan, "A Commercial Law for Software Contracting," *Washington and Lee Law Review* 76 (2019): 775.

[16] Trey Herr, Robert Morgus, Stewart Scott, and Tianjiu Zuo, "Buying Down Risk: Cyber Liability," Atlantic Council, May 3, 2022, https://www.atlanticcouncil.org/content-series/buying-down-risk/cyber-liability/.

[17] Chinmayi Sharma and Benjamin C. Zipursky, "Who's Afraid of Products Liability? Cybersecurity and the Defect Model," *Lawfare*, Oct. 19, 2023, https://www.lawfaremedia.org/article/who-s-afraid-of-products-liability-cybersecurity-and-the-defect-model.

[18] Regulation (EU) 2019/881 of the European Parliament and of the Council of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) No 526/2013 (Cybersecurity Act), https://eur-lex.europa.eu/eli/reg/2019/881/oj. The certification program is described in Title III, Articles 46-65. For a brief overview illustrating the system's complexity, see ENISA, *EU Cybersecurity Certification*, https://certification.enisa.europa.eu/.

[19] Executive Order 14028, "Improving the Nation's Cybersecurity," Federal Register 86 (May 12, 2021): 26633.

federal government is pursuing a self-attestation process for the software it purchases.[20] This may work for the federal government, since such attestations will be automatically backed up by the civil and criminal penalties of the False Claims Act,[21] which would apply notwithstanding any disclaimers.

But whether policymakers choose the frame of warranty or negligence or products liability or something else, such as certification or attestation, the question remains: What is the standard of care? Moreover, can it be developed quickly enough and with sufficient clarity to be compatible with the rapidly evolving world of software, where the problem is urgent and uncertainty will stifle innovation?

If a liability regime were situated within the context of warranty, what is it that is warranted? At common law, the implied warranty of merchantability was notoriously ill defined, leading twentieth-century legal titan Karl Llewellyn to write that "there is no more puzzling question than what this word [merchantable] means."[22] Language in the Uniform Commercial Code is no help: The implied warranty of merchantability defined there says that goods "must be at least such as … pass without objection in the trade" and they must be "fit for the ordinary purposes for which such goods are used."[23]

To get clarity, one could turn to legislators. But a statutorily imposed warranty that software products are completely free of defects would be too strict, since it is widely assumed that the development of perfectly secure software would be too expensive, if not impossible. A warranty that a product is free of known defects, however, would be too narrow, rewarding those who did not look for yet unrecognized vulnerabilities.[24] Something more fine grained is needed.

Likewise, under either a negligence or a products liability framework, reform would have to define a standard. A key consideration in establishing a negligence standard for software, according to Herr and colleagues,[25] is defining a duty of care. And as legal scholar David Owen wrote about products liability law in general, danger inheres in every product, so the central question of "How much safety is enough?" arises in every design defect case.[26] As Sharma and

---

[20] Shalanda D. Young, "M-22-18, Memorandum for the Heads of Executive Departments and Agencies, Enhancing the Security of the Software Supply Chain Through Secure Software Development Practices," Office of Management and Budget, Sept. 14, 2022, https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf.

[21] Cornell Law School, "False Claims Act," https://www.law.cornell.edu/wex/false_claims_act.

[22] Karl N. Llewellyn, *Cases and Materials on the Law of Sales* (1930): 324. See Robert W. Gomulkiewicz, "The Implied Warranty of Merchantability in Software Contracts: A Warranty No One Dares to Give and How to Change That," *UIC John Marshall Journal of Information Technology & Privacy Law* 16 (1998): 393, https://repository.law.uic.edu/cgi/viewcontent.cgi?article=1238&context=jitpl.

[23] U.C.C. § 2-314, *supra* note 11.

[24] Chinmayi Sharma and John Speed Meyers, "Bugs in the Software Liability Debate," *Just Security,* July 18, 2023, https://www.justsecurity.org/87294/bugs-in-the-software-liability-debate/.

[25] Herr, Morgus, Scott, and Zuo, *supra* note 16.

[26] David G. Owen, "Defectiveness Restated: Exploding the 'Strict' Products Liability Myth," *University of Illinois Law Review* (1996): 743, 777.

Zipursky explain in their argument for products liability, the design defect question in ordinary products liability cases turns on a reasonableness inquiry. In fact, it turns on two reasonableness inquiries: Was there a "reasonable" alternative design that the manufacturer could have used to avoid the vulnerability, and, without that alternative, was the actual design used "unreasonably dangerous"?

Years ago, Twerski complained that courts were too often assessing whether a product suffered from a design defect the way that Justice Potter Stewart infamously defined obscenity. "Unfortunately," Twerski wrote, "'I know it when I see it' will not suffice as a judicial standard for products liability."[27] Yet the nonbinding 1998 Restatement (Third) of Torts: Products Liability, which sought to summarize and guide U.S. law and for which Twerski was co-reporter, doesn't answer the question. It says that a product is defective in design "when the foreseeable risks of harm posed by the product could have been reduced or avoided by the adoption of a reasonable alternative design … and the omission of the alternative design renders the product not reasonably safe" (emphasis added).[28]

So the issue would go to judges and juries on a case-by-case basis. Bryan Choi argues that courts are institutionally quite capable—maybe more capable than regulatory agencies—of making the kinds of decisions necessary to assess cybersecurity flaws.[29] And Sharma and Zipursky argue that products liability law over the past few decades has spawned a sophisticated set of rules and principles for dealing with expert testimony that will be useful if not indispensable in evaluating evidence for and against a claim of defective design.[30]

But I don't think we have time for decades of case-by-case development of a software standard of care. As in the data breach arena, where I know of only one case ever that has reached a judgment on the merits, most software liability cases would settle with no admission of wrongdoing by the defendant and thus no identification of what the design defect, if any, was. Even if cases were to go to trial, jury verdicts offer a poor way to develop a standard of care. A typical jury verdict form in a products liability case asks, "Did the defendant supply a product that was not reasonably safe as designed," with the jury required only to check one of two boxes, "yes" or "no."[31] And juries are not questioned about the basis for their verdicts.

Moreover, as applied to software, the traditional test of design defect offers no safe harbor. It is simultaneously too strict and not definitive enough. For any given flaw, there is almost always an alternative design. And, as Gary Marchant and Rachel Lindor argue, the software

---

[27] Twerski, *supra* note 3, at 304–05. Justice Stewart wrote "I know it when I see it" in his concurring opinion in *Jacobellis v. Ohio*, 378 U.S. 184, 197 (1964) (reversing the conviction of a manager of a motion picture theater, who was convicted under a state obscenity law of possessing and exhibiting an allegedly obscene film).

[28] Restatement (Third) of Torts: Products Liability § 2 (1998).

[29] Bryan H. Choi, "Institutional Choice for Software Safety Standards," *Hastings Law Journal* 73 (2022): 1461, https://www.hastingslawjournal.org/wp-content/uploads/8.-Choi_Final.pdf.

[30] Sharma and Zipursky, *supra* note 17.

[31] *Pattern Jury Instructions-Civil*, 7th ed. (Vols. 6 and 6A, *Washington Practice* Series) WPI 110.30.01.

manufacturer will almost always lose the cost-benefit analysis required by the "reasonable" alternative test, given the small marginal cost of an alternative for any particular flaw.[32] However, for the other half of the test—is the product "unreasonably dangerous"?—there seems to be no touchstone: All software is potentially vulnerable to attack, but is any software unreasonably dangerous, given the prevalence of flaws and the iterative lifecycle model dominant in the industry, which accepts flaws (and patches) as the price of development? Leaving the question of what a design flaw is entirely to judges and juries, despite their history of coping with other sophisticated technologies, risks decisions that send inconsistent and confusing signals to industry.

So, it is hard to see certainty—a key criterion for a liability scheme—arising from a reasonableness standard applied case by case.

Instead, as Derek Bambauer argued in 2021, cybersecurity liability should be based on some clear-cut dos and don'ts to which liability should attach.[33] Bambauer was talking mainly about the practices of system operators, but the same should be true of software, as Bambauer and Melanie Teplinsky have explored recently.[34]

SECTION 3: EXISTING SOFTWARE STANDARDS FOCUS ON THE PROCESS, NOT THE PRODUCT

In what is perhaps the most important contribution of their article, Sharma and Zipursky point out that products liability focuses on the product, not on the processes of the manufacturer. This, they argue, makes the products liability framework especially desirable as the model for software.

In contrast, however, existing secure software development standards (by which I also mean frameworks and guidelines) focus largely on process. The secure software development framework promulgated in February 2022 by NIST[35] consists mainly of recommended processes: "Track and maintain the software's security requirements, risks, and design decisions." "Follow all secure coding practices that are appropriate to the development languages and environment to meet the organization's requirements." "Determine which compiler, interpreter, and build tool features should be used and how each should be configured, then implement and use the approved configurations." None of these actually describes a secure product; none identifies a weakness that should be avoided. As Sharma and Zipursky note, a developer could easily comply with all of these processes and still make bad choices and produce insecure software.

---

[32] Gary E. Marchant and Rachel A. Lindor, "The Coming Collision Between Autonomous Vehicles and the Liability System," *Santa Clara Law Review* 52 (2012): 1321, 1334.

[33] Derek E. Bambauer, "Cybersecurity for Idiots," *Minnesota Law Review Headnotes* 106 (2021): 172, https://minnesotalawreview.org/wp-content/uploads/2021/11/Bambauer_Final.pdf.

[34] Derek E. Bambauer and Melanie J. Teplinsky, "Shields Up For Software," *Lawfare,* Dec. 19, 2023, https://www.lawfaremedia.org/article/shields-up-for-software.

[35] Murugiah Souppaya, Karen Scarfone, and Donna Dodson, *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities* (Feb. 2022), NIST Special Publication 800-218, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf.

The international framework for certifying avionics software[36] is no more definitive: "System requirements allocated to software, including safety-related requirements, are developed and refined into software requirements that are verified by the software verification process activities." "Methods and tools should be chosen that aid error prevention and provide defect detection." Other statements in the framework are so definitive as to be meaningless: A recommendation to "choose … verification methods that ensure that errors introduced are detected" is simply unrealistic, since there is no verification method that will detect all errors.[37]

The principles and approaches for secure-by-design software issued in revised version in October 2023 by the Cybersecurity and Infrastructure Security Agency (CISA) and its partners in a dozen other countries[38] includes some specific features or controls discussed below. But the CISA principles are largely illustrative, and they too rely heavily on process: "Convene routine meetings with company executive leadership to drive the importance of secure by design and secure by default within the organization." "Use a tailored threat model during resource allocation and development to prioritize the most critical and high-impact features." "[C]onduct security-focused user research to understand where the security user experience (UX) falls short." "Document conformance to a secure SDLC [software development lifecycle] framework." Other recommendations in the guide, while outcome focused, are indeterminate: "Reduce hardening guide size." (This is a response to the fact that some manufacturers, rather than setting safer defaults, opt to produce a hardening guide that customers must implement at their own expense, and these guides may be too voluminous and too complex to implement.)

SECTION 4: THE FLOOR: LESSONS FROM OTHER FIELDS

It doesn't have to be this way. Other fields that place liability on manufacturers and developers have evolved very definitive standards that focus on features, not processes. Take building codes, mostly promulgated by trade associations but widely enacted into law at the state or local level. Typical of their specificity is the National Electric Code, managed by the National Fire Protection Association and adopted in one of its versions by 42 states.[39] In 918 pages, the National Electric Code very precisely defines a standard of care, with requirements like this: The minimum size of conductors for voltage ratings up to and including 2000 volts shall be 14 AWG copper or 12 AWG aluminum or copper-clad aluminum. In the motor vehicle field (where

[36] Radio Technical Commission for Aeronautics, *DO-178C - Software Considerations in Airborne Systems and Equipment Certification* (2011), https://my.rtca.org/productdetails?id=a1B36000001IcmqEAC.

[37] For more, see Bryan Choi's brief analysis of how the avionics standard was softened over time. Choi, *supra* note 29, at 1471–72.

[38] Cybersecurity and Infrastructure Security Agency (CISA) et al., *Secure by Design - Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software* (2023), https://www.cisa.gov/sites/default/files/2023-10/SecureByDesign_508c.pdf.

[39] National Fire Protection Association, *NFPA 70: National Electric Code* (2023), https://www.nfpa.org/codes-and-standards/7/0/70 (requires purchase).

modern tort law arguably got its start[40]), the common law tort system has evolved a set of safety expectations, but they have been backed up over the past five and a half decades by highly definitive federal motor vehicle safety standards.[41] To pick just one example at random: Under federal rules, the ERPB (equilibrium reflux boiling point) of DOT Grade 4 brake fluid, used in most cars produced after 2006, shall not be less than 446°F.[42] (The boiling point of brake fluid is very important, since when the fluid boils, it turns to vapor, which is highly compressible, such that slamming the brake pedal will have little effect.)

The Consumer Product Safety Commission manages what is in essence a federal products liability regime. But although the express statutory mandate of the commission is "to protect the public against unreasonable risks of injury associated with consumer products,"[43] it does not rely on the reasonableness test for identifying design defects. Instead, it publishes precise rules, like the one for walk-behind rotary power lawn mowers that requires a blade control system that will cause the blade motion to come to a complete stop within three seconds after release of the control.[44]

The pattern is repeated in field after field. Across the U.S. Code of Federal Regulations, there are more than 27,000 incorporations by reference of private technical standards, giving them the force of law.[45]

The objection will be raised that software is far more complex than a lawn mower or an automobile and even far more complex than the system of systems that is a large office tower subject to multiple building codes. But the goal for this part of the standard of care—setting a definitive floor on software security—would not be to address all possible flaws of code. It is to identify specific features that are no-nos and specific features that are must-haves. And the list does not have to be static; indeed, a software liability standard of care can be versioned just as rapidly as software itself is.

Another objection is that building codes and motor vehicle safety standards are based on the laws of physics, which are static and not consciously malicious, while computer code is a human construct facing persistent and adaptive adversaries with bad intent. But even in computer code, we know some features that consistently make products more secure and some features that consistently create vulnerabilities.

---

[40] See Jim Dempsey, "Cybersecurity's Third Rail: Software Liability," *Lawfare*, March 2, 2023, https://www.lawfaremedia.org/article/cybersecuritys-third-rail-software-liability (describing the groundbreaking 1916 New York high court decision in *MacPherson v. Buick Motor Company*, where then-state judge Benjamin Cardozo held that the automobile maker was responsible for defects in the product it assembled).

[41] See NHTSA, *A Drive Through Time*, https://one.nhtsa.gov/nhtsa/timeline/index.html (noting that the first federal safety standards for cars became effective Jan. 1, 1968).

[42] 49 C.F.R. § 571.116, *Standard No. 116; Motor vehicle brake fluids,* https://www.ecfr.gov/current/title-49/section-571.116.

[43] 15 U.S.C. § 2051, https://www.law.cornell.edu/uscode/text/15/2051.

[44] 16 C.F.R. § 1205.5, https://www.ecfr.gov/current/title-16/section-1205.5.

[45] NIST, *Standards Incorporated by Reference (SIBR) Database*, https://sibr.nist.gov/.

Choi is correct that the complexity of software has so far rendered both process standards and performance standards inadequate, if not outright failures.[46] But there is a third approach that has not yet been adequately explored: a liability standard that, as a floor, focuses not on development processes and not on unmeasurable performance goals but rather on the products themselves. An effective standard would probably express these specific requirements as good features that a product must have and vulnerable features that a product must not have.

Focusing on the product instead of the process has the advantage of encouraging outcomes-based or performance-based standards, which the Biden administration endorsed in its cybersecurity strategy. Much of what passes under the label of "performance based" isn't actually performance based, because there is no standard that can be measured.[47] What I am urging here is a focus on granular and objectively measurable aspects of product performance—specific behaviors that must be either in or out.

Focusing on product features also addresses the concerns that a liability standard would be too rigid and would be immediately outdated by innovation. Narrowly crafted behavior-based rules actually could avoid this problem. The behavior in log4j that allowed attackers to insert malicious code into a logged message and then allowed that code to access a malicious server[48] is a design no-no that will never be outdated. Likewise, neutralizing special elements within a pathname that can cause the pathname to resolve to a location that is outside of a restricted directory (the weakness known as "path traversal") would seem to be a durably required feature as long as software depends on pathnames and directories.[49]

SECTION 5: THE SOFTWARE LIABILITY FLOOR: COMPILING DEFINITIVE SOFTWARE DOS AND DON'TS

Already, there is a welter of definitive software features scattered across a number of "standards." (Here I use the word broadly to include technical guidance and frameworks.)

As hinted at above, the CISA secure-by-design guidance for software[50] includes some definitive items that focus on the products themselves and offer a glimpse of how a secure software legal

[46] See Bryan H. Choi, "Software as a Profession," *Harvard Journal of Law & Technology* 33 (2020): 557, https://jolt.law.harvard.edu/assets/articlePDFs/v33/33HarvJLTech557.pdf.

[47] See Jim Dempsey, "Cybersecurity Regulation: It's Not 'Performance-Based' If Outcomes Can't Be Measured," *Lawfare*, Oct. 6, 2022, https://www.lawfaremedia.org/article/cybersecurity-regulation-its-not-performance-based-if-outcomes-cant-be-measured.

[48] NIST, *National Vulnerability Database, CVE-2021-44228 Detail,* https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

[49] See MITRE Corp., "Common Weakness Enumeration, CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')," https://cwe.mitre.org/data/definitions/22.html.

[50] CISA, *supra* note 38.

standard of care could be constructed: "Eliminate default passwords." "Create secure configuration templates." "Provide logging at no additional charge."

Another source of product features is the guides developed by computer scientist Carl Landwehr, building on his idea of building codes for building code. Landwehr published one for power system supply software as well as one for medical devices software.[51] Landwehr's guides include many process-oriented elements, but they also define specific features that should be expected of all software. His approach of developing different standards for different contexts is probably right, although an initial generalized standard could be divided over time into specific standards for specific sectors or specific types of software.

Other definitive features can be found in the Microsoft Security Development Lifecycle (SDL) Process Guidance.[52] One of the goals of reforming software liability is to reduce the significance of Patch Tuesday, Microsoft's monthly release of dozens, sometimes scores of patches, with almost every month's release fixing some vulnerabilities that are critical and some that are being actively exploited. So something at Microsoft needs to change. Nevertheless, its internal guidance undeniably identifies many features of secure software that could be included in a baseline standard of care (as well as many practices that could be included in a process-focused safe harbor, discussed below).

The most recent version of the Microsoft development guidance that is available online is from 2012. It is safe to assume that since then many new items have been added. Making the current guidance public could provide a rich source of secure software features. Microsoft may be reluctant to release the current version for fear of giving tips to the bad guys, but the time for security by obscurity seems long past. Alternatively, CISA should get a copy confidentially, extract from it definitive features, and combine them with items from other sources, to obscure the Microsoft provenance.

But perhaps the most definitive source of dangerous features are CISA's annual publication of the most frequently exploited vulnerabilities[53] and the MITRE Corporation's annual list[54] of the twenty-five most dangerous software weaknesses (referred to as CWEs using a labeling and descriptive system called Common Weakness Enumeration[55]). Crucially, the latest version of the CISA list links vulnerabilities (specific to products and easily eliminated now that they have been

---

[51] Carl E. Landwehr and Alfonso Valdes, "Building Code for Power System Software Security," IEEECS, https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/BCPSSS.pdf; Tom Haigh and Carl Landwehr, "Building Code for Medical Device Software Security," IEEECS, https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/BCMDSS.pdf.

[52] Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2, May 23, 2012, https://www.microsoft.com/en-us/download/details.aspx?id=29884.

[53] CISA et al., "2022 Top Routinely Exploited Vulnerabilities," Aug. 3, 2023, https://www.cisa.gov/sites/default/files/2023-08/aa23-215a_joint_csa_2022_top_routinely_exploited_vulnerabilities.pdf.

[54] MITRE Corp., *supra* note 7.

[55] See MITRE Corp., "Common Weakness Enumeration," https://cwe.mitre.org/index.html.

identified) to weaknesses, which are generic expressions of software flaws. At least some of these weaknesses can be converted into concrete features to be avoided or features to be added. For example, the path traversal flaw, CWE-22, and how to avoid it have been known for decades. Allowing it into software should be grounds for liability. Yet it still emerged recently in a Microsoft product, where it was the cause of one of the most frequently exploited vulnerabilities of 2022. Indeed, a pretty compelling response to those who argue that the cybersecurity landscape is changing too rapidly to be subject to liability or any other form of regulation can be found in the fact that there are fifteen weaknesses that have been present in every list of the top twenty-five most dangerous software weaknesses from 2019 through 2023.[56]

As of January 3, 2024, MITRE's CWE list, version 4.13, contained a total of 934 weaknesses.[57] It will take someone with more computer coding knowledge than me to say whether all or only some subset of these weaknesses can be converted into specific no-nos. But as a lawyer and policy geek, I can easily pick out some weaknesses on the list whose presence in any product should be legally treated as a design defect—for example, CWE-798: Use of Hard-coded Credentials, such as passwords, which has been on every list of the most dangerous software weaknesses since 2019. By contrast, I would think that the buffer overflow described in CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ("Injection"), responsible for two of the top twelve most routinely exploited vulnerabilities in 2022, would be a definitive no-no, but I've heard the argument that current coding practices are unable to prevent it. (That's quite an indictment of software development.)

And yet another source of no-nos is the OWASP Top Ten, issued annually by the nonprofit, community-led Open Worldwide Application Security Project.[58] The Top Ten also uses the CWE taxonomy. Also noteworthy is the fact that commercial services are available to scan software for some subset of weaknesses on the CWE list, suggesting that the avoidance of these weaknesses is not too high a bar.[59]

My proposal here parallels that of Bambauer and Teplinsky, who would base a liability "inverse safe harbor"—a zone of automatic liability—on a set of "practices for which there is widespread consensus that they are dangerous."[60] Unlike Bambauer and Teplinsky, however, I do not find useful elements for software design in the regulations, guidelines, and Federal Trade Commission settlements applicable to data custodians or other system operators.

---

[56] MITRE Corp., "Stubborn Weaknesses in the CWE Top 25," https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html.

[57] MITRE Corp., "CWE List Version 4.13," https://cwe.mitre.org/data/index.html.

[58] OWASP, "OWASP Top Ten," https://owasp.org/www-project-top-ten/. From the Top Ten page: "The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. … Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code."

[59] See, for example, Veracode, "Veracode and the CWE," https://docs.veracode.com/r/c_review_cwe.

[60] Bambauer and Teplinsky, *supra* note 34.

We need both liability for software developers and liability for software users, with a fact-finding process that allocates responsibility between them for the mistakes associated with any particular cybersecurity incident. In terms of software users (that is, system operators), there is already a developed and quite rapidly developing system of liability. This system is a patchwork stitched together from common law negligence (where, for example, courts find a duty to protect personal information) and sector-specific rules such as the security rule for health care providers,[61] the binding standards for bulk electric supply promulgated under the Energy Policy Act,[62] and the regulations established for pipelines after the Colonial Pipeline incident.[63] Much more needs to be done to build out the regulatory framework for the cybersecurity of critical infrastructure,[64] but that is a separate issue, and any reform regarding software liability should not lessen the legal responsibility of system operators.

The rules-based standards for the developers of software will be different from the standard of care applicable to the users of software, whether the latter are custodians of personal data or operators of critical infrastructure. For example, choosing a secure software algorithm as the default is the obligation of the software developer, but encrypting data is the obligation of the data custodian. Not hard-coding passwords is the obligation of the developer, but ensuring employees do not set 1234 as their password is the obligation of the user of the software (the system operator).

How to move forward in developing a baseline list of features and controls for software developers? One approach is to legislatively task CISA with developing the list, updated regularly, based on objective measurements such as the annual list of most exploited vulnerabilities and the weaknesses behind them, the MITRE list of most dangerous weaknesses, and the OWASP Top Ten, as well as data gleaned from reporting under the still-to-be-implemented Cyber Incident Reporting for Critical Infrastructure Act. CISA is not a regulator and doesn't want to be one. If a tort law framework were adopted by legislation, and if the private litigation system were relied on for enforcement, CISA would not be the enforcer of the list. Perhaps CISA could issue a request for information asking industry and others to contribute to the list.

---

[61] The security rule promulgated under the Health Insurance Portability and Accountability Act is at 45 C.F.R. §§ 164.302–164.318

[62] See Rahul Awati and Ben Cole, "North American Electric Reliability Corporation Critical Infrastructure Protection (NERC CIP)," TechTarget, March 2022, https://www.techtarget.com/searchsecurity/definition/North-American-Electric-Reliability-Corporation-Critical-Infrastructure-Protection-NERC-CIP#:~:text=The%20North%20American%20Electric%20Reliability,(BES)%20in%20North%20America. The NERC CIP standards are at https://www.nerc.com/pa/Stand/Pages/ReliabilityStandards.aspx.

[63] Transportation Security Administration, *Security Directive Pipeline-2021-02D, Pipeline Cybersecurity Mitigation actions, Contingency Planning, and Testing* (effective July 27, 2023), https://www.tsa.gov/sites/default/files/tsa-sd-pipeline-2021-02d-w-memo_07_27_2023.pdf.

[64] See, for example, Jacob Horne and Jim Dempsey, "A Cyber Threat to U.S. Drinking Water," *Lawfare*, Dec. 21, 2023, https://www.lawfaremedia.org/article/a-cyber-threat-to-u.s.-drinking-water.

Whatever the ultimate liability regime, I would propose, drawing on tort law principles, that liability attach only when a design flaw is actually exploited and causes actionable damage to any particular person, questions that could only be decided case by case. This would avoid a trolling-type situation where plaintiffs scoured software for flaws that had never caused a breach or other failure. Bug bounties are intended to address yet-unexploited flaws, and I would expect a vulnerability management program to be part of the secure software development process that would provide the safe harbor discussed below.

SECTION 6: FOR ALL OTHER FLAWS: A PROCESS-BASED SAFE HARBOR

It would be nice to stop there, with a list of features that served as the definitive expression of a legal standard of care. However, the list by itself would be inadequate, even dangerous, for it could quickly become a ceiling on security as well as a floor. At the very least, relying solely on a definitive list would incentivize developers to do the bare minimum and would discourage security innovation.

Here, it is appropriate to draw from traditional negligence law and its products liability offshoot. Under both traditional tort law and under products liability law, failure to comply with a government standard is proof of a design defect, but compliance with federal or state requirements for the manufacture of products does not immunize the manufacturer from liability.[65] This is based on the recognition, directly pertinent to software, that the world of risks, on the ground, is too varied. No government-endorsed list could ever capture all risks, especially in a field as innovative and dynamic as software. Thus, under current law applicable to other domains, a plaintiff can always seek to prove that a product was defective even though it complied with a government or industry standard. That should be true of software. So a broader liability standard would be needed, probably best based on the products liability concept of design defect. For how to actually express that standard, I would defer to torts law experts, since there may be nuances depending on whether the test is phrased in terms of reasonableness or in terms of risk-utility analysis. Since legislation would be needed to create the liability scheme, legislation could spell out specific factors for deciding what is or isn't a defect.

However, as explained above, a design defect standard drawn from products liability would be too open ended, threatening software developers with unpredictable and unlimited liability. Thus, a safe harbor is needed, and this is where secure software development processes have a lot to offer. As the work of Danny Weitzner and others shows, process does make a difference; management-based practices are crucial to cybersecurity, even if their contribution is not quantifiable. Thus, adherence to certain processes should be encouraged, through the safe

[65] *Burch v. Amsterdam Corp.,* 366 A.2d 1079 (D.C.Ct. App. 1976) ("the overwhelming majority of courts presented with similar arguments in product liability cases have held that compliance with federal and state requirements for the manufacture and sale of products does not immunize a manufacturer or seller from liability"), https://law.justia.com/cases/district-of-columbia/court-of-appeals/1976/9774-3.html.

harbor. For security flaws not on the no-no list (that is, for security flaws above the floor), an entity would escape liability if it could show that it adhered to the safe harbor processes.

The NIST Secure Software Development Framework (SSDF) most emphatically is not the standard I have in mind. It was never intended to be used in determining liability. But the SSDF and other existing software development guides do contain the elements that could be composed into a safe harbor: Obtain provenance information (a software bill of materials, or SBOM, source composition analysis, and binary software composition analysis) for each software component, and analyze that information to better assess the risk that the component may introduce. (From the NIST SSDF.) Conduct fuzz-testing at both the component and system levels. (From the Landwehr and Valdes guide on power system software, and the Microsoft SDL guidance.) Conduct static analysis of source code and repair reported bugs in specified error categories. (Also from the Microsoft SDL guidance.) These are merely illustrative of the point that enforceable specificity is possible even within the relatively flexible and open-ended context of software development.

As with the question of who would define the floor, there is the critical question of who would define the safe harbor. What governmental entity or standards body would have both the expertise and the institutional fortitude to withstand the inevitable industry pressure for a weak, broadly worded standard? A logical locus of authority would be CISA, which has demonstrated with its cross-sector cybersecurity performance goals for critical infrastructure[66] that it can develop concrete standards that go beyond ratifying the status quo. Is CISA equipped to undertake the task of defining an enforceable standard of care for software? Given the make-or-break importance of institutional design, these questions are beyond the scope of this paper.

CONCLUSION: IF NOT THIS, THEN WHAT?

There are plenty of questions that need to be addressed: How should actionable damages be defined? Should software users who misconfigure their systems or disable security features or fail to install patches be precluded from holding software developers liable? Most importantly, perhaps, would be questions of timing. For how long should legacy systems be immunized under a grandfather clause? Accepting the concern that current practices cannot capture even frequently recurring flaws such as buffer overflows, how much lead time would be appropriate before liability began to attach?

The goal here is not to create perfectly secure software; it is instead to compensate users of software for losses caused by unreasonably dangerous defects in software.

Among other issues, the system proposed here is intended to respond to the criticism that software security is context dependent, so that for some products it is appropriate to ignore certain weaknesses, since they are unlikely to be exploited. By including harm as an element of

[66] CISA, "Cross-Sector Cybersecurity Performance Goals," https://www.cisa.gov/cross-sector-cybersecurity-performance-goals.

a cause of action, the approach respects context and trade-offs: Applying a cost-benefits calculation, certain flaws in a certain product might be tolerated. If the trade-off was calculated correctly, harms will be nonexistent or minimal and thus inexpensively compensated. But developers should not be able to keep the benefits of the trade-off while passing the costs to others.

The system proposed here is also designed to minimize the cost of litigation while incentivizing improvements in software security. By defining a floor based on common weaknesses, it would create a clear-cut test that would involve no inquiry into who said what or how the flaw appeared. The presence of a well-known flaw—the outcome—would be evidence of liability per se, and the process by which the developer got there would be irrelevant. For these flaws, discovery—a disruptive and costly feature of much modern litigation—would be minimal. For other flaws, the defendant would have the ability to avoid liability if it could show that it adhered to secure software development practices. This would provide the impetus for companies to adopt and follow a secure software development process, which could be tailored to their own flavor of software.

If not this, then what?

It is not inconceivable that one day judges would decline to recognize the contractual waivers that currently allow software developers to avoid responsibility for their mistakes. After all, courts have long since denied automobile makers the ability to disavow liability for defects in their products.[67] Perhaps for software it will occur in a consumer case or a case of a small business, where the power disparities between software developer and customer are particularly acute. A decision that invalidates waivers for software, even by judges in just one state, would open the floodgates for software litigation with no standard other than reasonableness. The ensuing confusion would surely hurt the software sector. Likewise, under the Federal Trade Commission Act or comparable state laws that prohibit unfair or deceptive practices, aggressive regulatory enforcement action on software security under the prevailing reasonableness standard would be chaotic and unpredictable.

Better to start now, using the real-world data at hand in terms of commonly exploited weaknesses to define certain features that constitute a floor, with a process-based safe harbor that would cabin liability for other harm-causing defects.

[67] *Henningsen v. Bloomfield Motors, Inc.,* 32 N.J. 358, 161 A.2d 69 (1960), https://casetext.com/case/henningsen-v-bloomfield-motors-inc.