SOFTWARE TORTS AND SOFTWARE CONTRACTS: REFRAMING THE DEVELOPER'S DUTY

MICAH R. MUSSER*

Flawed software costs businesses and consumers millions of dollars every year, but existing tort law does not generally require developers to compensate others for economic injuries caused by bad code. Discontented scholars and policy analysts have produced an array of proposals that would force developers to pay for harms flowing from vulnerabilities that hackers exploit to injure software users. This basic model—which would impose a duty on developers to eliminate security-related vulnerabilities but not other types of software flaws—dominates legislative and academic debates about reform. This Note argues that this focus is misconceived. It is technically ambiguous, doctrinally anomalous, and would throw national security and consumer welfare goals into conflict. Liability proponents have focused on it because they recognize that imposing new duties on software developers must realistically be limited in some way. Although the vulnerability-based limitation is ultimately misguided, this Note proposes that a party-based limitation restricting recovery to parties in near-privity is more defensible. Focusing on party-based limitations on duty instead of a vulnerability-based limitation would require thinking of software development not as a product, but rather as a professional practice subject to malpractice-like standards. This reframing, I argue, better aligns proposals for expanding software developers' duties with existing tort doctrine while focusing a liability evaluation on the most important aspects of the software development process.

 J.D. Candidate, 2026, New York University School of Law; B.A., 2019, Georgetown University. I am grateful to Mark Geistfeld, Maggie Gardner, Barry Friedman, Sam Issacharoff, Daniel Francis, and Randal Milch for reading and commenting on various drafts of this piece. I am also indebted to the valuable feedback of the Furman Academic Scholars and the Center for Cybersecurity's Cyber Scholars who read this piece, and especially Keton Kakkar, who offered comments on at least three distinct drafts. Jack Samuel and Dakota Cary deserve special mention for their excellent comments and encouragement on this project. My thanks also to the various members of the *New York University Law Review* who engaged thoughtfully with my arguments, and in particular to Jeremy Venook, whose attentive editing has made the piece immeasurably better. Finally, I am grateful to my former colleagues at the Center for Security and Emerging Technology for allowing me to test their intuitions regarding the appropriate scope of developer liability.

Introduction

One afternoon, a tech company pushes what is meant to be a routine software update to its users. Unbeknownst to the company, the code in that update contains a flaw. As the update reaches consumers throughout the country, IT devices suddenly shut down, displaying "blue screens of death" that leave computers permanently disabled. Businesses are crippled, including critical services in the life sciences and transportation industries. The disruptions sprawl far beyond the actual users of the tech company's software, causing approximately $10 billion in losses. Once the dust settles, congressional committees convene hearings on the subject, taking the opportunity to opine at length about America's vulnerability to attacks in cyberspace.

This fact pattern applies equally well to two of the largest cyber incidents in history: the Russian-sponsored NotPetya attack in 2017, which inserted a damaging exploit into custom accounting software developed by the Ukrainian company Linkos Group,[1] and the aftermath of CrowdStrike's buggy update to its Falcon Sensor product in 2024.[2]

---

[1] On the origin and impacts of NotPetya, see generally Andy Greenberg, *The Untold Story of NotPetya, the Most Devastating Cyberattack in History*, Wired (Aug. 22, 2018), https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world [https://perma.cc/8SPV-DYBQ]. For a post-NotPetya congressional hearing, see *Cyber Threats to Our Nation's Critical Infrastructure: Hearing Before the Subcomm. on Crime & Terrorism of the S. Comm. on the Judiciary*, 115th Cong. (2018) (statement of Michael Moss, Deputy Dir., Off. of the Dir. of Nat'l Intel.).

[2] For coverage of the CrowdStrike incident, see Dan Milmo, Julia Kollewe, Ben Quinn, Josh Taylor & Mimi Ibrahim, *Slow Recovery from IT Outage Begins as Experts Warn of Future Risks*, The Guardian (July 19, 2024), https://www.theguardian.com/australia-news/article/2024/jul/19/microsoft-windows-pcs-outage-blue-screen-of-death [https://perma.cc/K4WN-PYRW]; Lian Kit Wee, *Here Comes the Wave of Insurance Claims for the CrowdStrike Outage*, Bus. Insider (July 22, 2024), https://www.businessinsider.com/businesses-claiming-losses-crowdstrike-outage-insurance-billions-losses-cyber-policies-2024-7 [https://perma.cc/NQF3-YHNG].

The more recent CrowdStrike incident occurred at a time of renewed interest in the possibility of federal legislation to impose tort liability on software developers for harms caused by flawed code.[3] It also attracted far more criticism: While the disruption caused by CrowdStrike appears to have been the result of a rushed and untested update,[4] NotPetya was a piece of malware carefully slipped into an unsuspecting Ukrainian company's production process by a sophisticated and state-backed hacking group.[5] Commentators largely agree that CrowdStrike made risky decisions to cut corners, while Linkos Group almost certainly could not have done anything to prevent being compromised by such a sophisticated and well-resourced opponent.[6]

And yet, puzzlingly, virtually all current proposals for software tort liability would subject a company like Linkos Group to potential liability, while leaving CrowdStrike off the hook. The reason for this is straightforward: Existing liability proposals link liability to the exploitation of a *vulnerability* in software.[7] Vulnerabilities are flaws in code which can be exploited by malicious actors to degrade the confidentiality, integrity, or availability of data, resulting in the exposure of sensitive information, data erasure, computer shutdowns, or many other types of harms.[8] But vulnerabilities are a subset of software flaws (call the general category "bugs"), and many bugs can cause exactly the same types of harms *without* the presence of a malicious third-party hacker. Yet the dominant paradigm among proponents of software tort reform does not acknowledge the existence of bugs and links liability only to the existence of a vulnerability.[9]

Advocates for existing proposals have generally emphasized two grounds for structuring liability in this way. First, because almost all

---

[3] *See infra* Section I.C.

[4] *See* Zeba Siddiqui, *CrowdStrike Update that Caused Global Outage Likely Skipped Checks, Experts Say*, Reuters (July 22, 2024), https://www.reuters.com/technology/cybersecurity/crowdstrike-update-that-caused-global-outage-likely-skipped-checks-experts-say-2024-07-20 [https://perma.cc/SJ5C-DKT2].

[5] *See* Greenberg, *supra* note 1 (observing that NotPetya was the culmination of an operation that required "months of unfettered access to victims' networks").

[6] *Compare* Barath Raghavan & Bruce Schneier, *The CrowdStrike Outage and Market-Driven Brittleness*, Lawfare (July 25, 2024), https://www.lawfaremedia.org/article/the-crowdstrike-outage-and-market-driven-brittleness [https://perma.cc/KBL3-ZXL9] ("Last week's update wouldn't have been a major failure if CrowdStrike had rolled out this change incrementally: first 1 percent of their users, then 10 percent, then everyone."), *with* Greenberg, *supra* note 1 (describing NotPetya's use of leaked NSA-developed exploits which made use of subtle zero-day vulnerabilities to wreak havoc).

[7] *See, e.g.*, Cyberspace Solarium Comm'n, Legislative Proposals 117 (2020) [hereinafter CSC Proposal] (providing text of a proposed statute which would attach liability only to harms due to "a cyber incident caused or enabled by a covered security vulnerability").

[8] *See infra* note 38 and accompanying text.

[9] *See infra* Section I.C.

commentators agree that some rate of error is inevitable in software development, they suggest that imposing liability for vulnerabilities but not other types of bugs will help to guard against the risk of excessive liability.[10] Second, they see software liability tort reform as an opportunity to promote the national security interest in defending against intrusions from hostile governments and nonstate actors.[11] But the decisions shaping this focus have also been made in passing, with little commentary or justification; proponents of software tort liability have spent far more time discussing how to define a developer's standard of care than how to scope their duty.[12] This Note argues that such a focus is mistaken. In order to produce a coherent and workable federal tort regime for harm-causing code, proponents of software liability need to spend more time articulating the scope of a developer's duty—the element of a tort claim which, unlike the question of whether a defendant's conduct fell short of reasonable care, is ordinarily decided as a matter of law.[13] If the developer's duty is articulated in a principled and well-limited fashion, it would reduce the pressure to define precisely what constitutes "reasonable" software development practices, an issue which has consumed the existing debate.[14]

At a high level of generality, we might categorize no-duty rules in two ways. One set of duty-based limitations holds that actors are not responsible for certain types of *harms* to others. Thus, although everyone has a general duty to exercise reasonable care if their conduct creates a risk of physical harm to others, they do not have a general duty to avoid unintentionally causing economic loss to others.[15] A different set of duty-based limitations holds that actors only owe certain duties to particular *persons*. This is especially common in the case of affirmative duties; landlords, for instance, have special duties to protect tenants from foreseeable criminal conduct, as do other actors with a "special relationship" to a third party.[16] Although manufacturers of products or

---

[10] *See, e.g.*, *infra* notes 140, 182 and accompanying text; *see also infra* Section I.C.

[11] *See infra* notes 64–65 and accompanying text; *see also infra* Section II.D.

[12] *See infra* Section I.C.

[13] *Compare* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 7 cmt. i (AM. L. INST. 2010) (discussing the role of the court in identifying categorical no-duty rules), *with id.* § 8(b) (identifying the role of the jury in determining whether a defendant's conduct lacked reasonable care).

[14] *See infra* Section I.C.

[15] *Compare* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 7(a) (AM. L. INST. 2010) (providing the default rule of duty in physical-harm cases), *with* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR ECON. HARM § 1(1) (AM. L. INST. 2020) (providing the general no-duty rule in economic-harm cases).

[16] *See* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 40 (AM. L. INST. 2010).

product components were once liable only to persons who purchased directly from them (the privity requirement), this is no longer true, and products liability now recognizes a duty to all subsequent purchasers.[17]

The vulnerability-based focus of current software liability proposals is a restriction on duty of the former kind. These proposals would require developers to guard against introducing vulnerabilities into their software, but not other types of potentially harm-causing flaws.[18] This is, moreover, the *only* meaningful limitation on duty that proponents of software liability would apparently retain.[19] Unfortunately, it is also an arbitrary and impractical limitation. It is not always possible to determine abstractly whether a particular bug is a vulnerability or not, which means that the proposal of linking liability to the existence of a vulnerability creates technical ambiguities that have not been explicitly analyzed by defenders of the vulnerability paradigm.[20] As a doctrinal matter, tort law is suspicious of duties to prevent third parties from causing certain types of harms, especially where the defendant has no duty to avoid negligently causing the same type of harm themselves.[21] And although proponents of the vulnerability paradigm often argue that software tort reform would be good both for national security and for consumers, restricting liability to cover only vulnerabilities would incentivize developers to rush buggy patches that directly harm consumers in order to eliminate low-risk vulnerabilities.[22]

This Note argues that proponents of software liability should instead focus on identifying appropriate *party-based* limitations on developers' duties. In particular, software developers should only have duties to those who stand in a contractual relationship of near-privity. Privity-based limitations have been attacked—especially in the products liability context—as "pernicious,"[23] "unnecessary,"[24] "anachronistic,"[25] and "unintelligible."[26] Contemporary proponents of software liability, who are often eager to treat software as a product, seem similarly averse

---

17 *See* William L. Prosser, *The Assault upon the Citadel (Strict Liability to the Consumer)*, 69 YALE L.J. 1099, 1099–102 (1960) (discussing the fall of privity restrictions for products liability suits).

18 *See infra* Section I.A for a discussion of the meaning of vulnerabilities as contrasted with bugs.

19 *See infra* Section I.C.

20 *See infra* Section II.A.

21 *See infra* Sections II.B–C.

22 *See infra* Section II.D.

23 Prosser, *supra* note 17, at 1134.

24 *Id.*

25 *See* Richard C. Angino, Comment, *Limitation of Liability in Admiralty: An Anachronism from the Days of Privity*, 10 VILL. L. REV. 721, 724 (1965).

26 *See* Geoffrey C. Hazard, Jr., *The Privity Requirement Reconsidered*, 37 S. TEX. L. REV. 967, 967 (1996).

to contract-based limitations on liability.[27] But privity-based limitations on liability have not totally been eliminated from the domain of tort, especially in cases of economic (not physical) harm—and, in particular, in contexts of professional malpractice that cause economic loss.[28] Many of the harms of flawed code which software liability is meant to address involve economic harms, where the risk of cascading impacts and correlated risk make the specter of "crushing liability" extremely acute.[29] In such a context, liability should not operate as an "on-off" switch which would immunize companies from liability for all non-security bugs, yet subject them to potentially ruinous judgments upon a single finding of liability for a single vulnerability. The way to address the concern of crushing liability is by limiting *who* can recover damages, not by trying to carefully delimit *what* types of errors trigger liability as to every software user in the world. And this privity-based limitation on liability leads naturally to the thought that software liability might be best thought of as a form of professional malpractice liability, since malpractice liability is one of the few contexts in which tort law acts to supplement contractual relationships.

This Note is structured as follows. Part I offers background in three stages. First, I introduce some key concepts regarding software. Second, I discuss the existing barriers that have prevented developers from being liable in tort for damages caused by insecure or faulty software. And third, I canvass recent proposals for reform, highlighting both their focus on the question of "reasonable" cybersecurity and their emphasis on vulnerabilities. Part II critiques this harm-based limitation on liability. I argue that limiting liability to harms caused by vulnerabilities is technically ambiguous (Section II.A), doctrinally misshapen (Sections II.B–C), and objectionable on policy grounds (Section II.D). Finally, in Part III, I argue that party-based limitations are a better way to avoid the specter of crushing liability than harm-based limitations on developers' duties, and, in particular, that a limitation on recovery to parties in near-privity is the best way to implement a software liability regime. I offer this argument by using the work of other scholars who have argued that software development should be treated like a profession subject to malpractice standards. A malpractice frame better justifies a liability regime while also offering reasons to limit the scope of that duty, reasons which have been underappreciated by proponents of software liability.

---

[27] *See infra* Section I.C.

[28] *See* Catherine M. Sharkey, *The Remains of the Citadel (Economic Loss Rule in Products Cases)*, 100 MINN. L. REV. 1845, 1862–65 (2016) (identifying privity-based limitations in cases of economic harms caused by defective products); Hazard, *supra* note 26, at 967 (acknowledging the continued importance of privity in cases of attorney malpractice).

[29] *See infra* Section III.A.

I
Contemporary Proposals for Software Tort Liability

This Part situates the reader. Before jumping directly to the main substance, Section I.A introduces some key concepts and terminology regarding software development. Section I.B discusses the status quo regarding software tort liability. It discusses where liability does and does not exist, and why software developers face relatively little legal responsibility for their code as compared to most other industries. Section I.C ends by overviewing the current state of play regarding proposals for reform. I review existing proposals and discuss their focus on both vulnerabilities (as opposed to bugs more generally) and on specifying the meaning of "reasonable care" in the cybersecurity context (as opposed to explicitly discussing limitations on the developer's duty).

A. Definitions: Software, Bugs, and Vulnerabilities

It is worth beginning by offering up a definition of "software." Software can be thought of as all of the code which runs on a computer, in contrast to its physical components.[30] A key characteristic of modern software is that it can often be updated remotely after being sold, whereas physical components generally cannot. Some software comes embedded in physical devices (like the operating system used by your laptop); some is packaged into applications that are sold to end users (think Microsoft Word); some is written by hobbyists and posted online for anyone to use as they see fit. When software is simply made available to anyone to use for free under a license that permits broad reuse without bilateral agreements—that is, in the language of contracts, when the user and the developer do not stand either directly or indirectly in privity with one another—it is referred to as "open source software."[31] Note that there is nothing which requires software to be deterministic; any software which makes use of statistical AI models is fully covered by this definition. A software tort regime would therefore apply to AI developers.[32] Although applying a software tort regime to AI may raise

[30] *See generally Software*, Brittanica (July 23, 2025), https://www.britannica.com/technology/software [https://perma.cc/CQ9G-E7DV].

[31] For an overview of the GitHub ecosystem of open source software, see Christian Schoeberl, *GitHub Data: Capturing Open Source Software and Implementation*, Ctr. for Sec. & Emerging Tech. (Sept. 28, 2022), https://cset.georgetown.edu/publication/github-data-capturing-open-source-software-and-implementation [https://perma.cc/YNS9-WT8J]. Whether developers of open source software should be subject to tort liability is an important point of dispute in current debates. *See infra* Section III.A.

[32] There is significant interest in tort liability for AI developers. *See, e.g.*, Press Release, Am. L. Inst., ALI Launches Principles of the Law, Civil Liability for Artificial Intelligence (Oct. 22, 2024), https://www.ali.org/news/articles/ali-launches-principles-law-civil-liability-

particular problems of implementation,[33] the purpose of this Note is to seek greater clarity on the core tort framework which should apply to software generally. I therefore do not extensively discuss concerns specific to AI systems.

Three software-related concepts are important to define here: bugs, vulnerabilities, and exploits. *Bugs* refer to any error in software introduced by a developer's failure to consider some edge case which causes a system to behave in an unanticipated or undesirable way.[34] For instance, developers may program computers to store the current year in memory as a two-digit integer without considering what happens when the year changes from 1999 to 2000; when that happens, some computers may malfunction, corrupting data or shutting down completely.[35] The relevant edge case which may trigger the bug could occur during "normal" system operation, as in the Y2K example, or it could be triggered by unexpected user input or an unanticipated system configuration.[36] Because it is impossible for developers to fully test for bugs that may emerge in any arbitrary system configuration, it is typical to test in a limited set of environments and advise users about proper system configuration. (Think of downloading a new application

artificial-intelligence [https://perma.cc/6XRS-WPN3]. How the tort regimes proposed by various authors would interact with the parallel discussion on software liability that this Note engages in is beyond the scope of my argument.

[33] See the discussion of adversarial attacks on AI systems in *infra* Section II.A.

[34] On terminology, see Irena Bojanova & Carlos Eduardo C. Galhardo, *Bug, Fault, Error, or Weakness: Demystifying Software Security Vulnerabilities*, 25 IT Pro. 7, 7–8 (2023). My use of the term "bug" includes both what Bojanova & Galhardo refer to as "bugs" as well as what they refer to as "faults." Note that bugs might also refer to more literal errors, like a typo in a codebase. The "failure to consider an edge case" frame is nonetheless a more useful one for thinking about bugs, because it cuts to issues of *design*, rather than implementation errors that might be more analogous to product malfunctions (and, as such, for which a case for liability seems even easier to establish). The definition I use here, which focuses on unanticipated or unexpected behavior, does not attempt to clarify whether the behavior must be unanticipated by the *user* or the *developer*. For simplicity, I focus only on software flaws that unambiguously are viewed as undesirable by both parties. Intuitively, if a system behaves in a way which the user views as undesirable but the developer views as routine and expected, the proper way to proceed would be to analyze which party's expectations are more reasonable. Where there is a contract between the developer and the user, this appears to become straightforwardly a contract interpretation question which focuses on the parties' reasonable expectations. Already in the definitional stage, then, we see that there are ambiguities which are best resolved by applying contract principles to software exchanges—a feature which bolsters my claim that software tort liability should seek to complement and not supplant contractual relationships. *See infra* Part III.

[35] *See* Comm. on Gov't Reform & Oversight, The Year 2000 Problem, H.R. Rep. No. 105-827, at 2 (1998).

[36] For a discussion of misconfiguration errors, see generally Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou & Shankar Pasupathy, *Do Not Blame Users for Misconfigurations*, Proc. Twenty-Fourth ACM Symp. on Operating Sys. Principles 244, 244 (2013).

that indicates suitability for Windows 10 or 11, but expects you not to use the software on a thirty-year-old computer running Windows 95.) Still, unexpected configurations happen, and when they do, bugs can be disastrous.[37]

Vulnerabilities are a subset of bugs which a malicious actor could exploit to degrade the confidentiality, integrity, or availability of data stored on a computer system.[38] For instance, if an unexpected user input to a database simply displays an error on the inputting user's computer, it is a bug but not a vulnerability; on the other hand, if the input can be manipulated to cause the database to reveal or erase stored information, it is a vulnerability (and a bug). For the purposes of this Note, it is sufficient to say that all vulnerabilities are bugs, but not all bugs are vulnerabilities.[39] And yet, *there is no abstract or inherent technical difference* between bugs and vulnerabilities: Both can be introduced through similar means, and there are often protracted debates about whether a particular bug can or cannot be exploited to circumvent a security constraint.[40] Further, many bugs are vulnerabilities only in

[37] Talk of "unanticipated edge cases" or "unexpected configurations" immediately suggests questions about foreseeability and proximate causation. This Note brackets those questions. It is enough to observe that if a developer's duty requires them to guard against all unintentionally inflicted economic injuries, then the developer's duty would include an obligation to guard against all bugs. Questions about the standard of care might involve disputes over whether a reasonable testing process would have uncovered a particular bug or whether a particular user configuration was foreseeable to the developer. These are difficult questions, but my focus in this Note is on asking whether there are any flaws or harms which a developer ought to *categorically* have no duty to prevent.

[38] The origin of the "CIA Triad" (confidentiality, integrity, and availability) is sometimes attributed to Jerome H. Saltzer & Michael D. Schroeder, *The Protection of Information in Computer Systems*, 63 Proc. IEEE 1278, 1280 (1975). Saltzer and Schroeder refer to these risks as "unauthorized information release," "unauthorized information modification," and "unauthorized denial of use." *Id.* Note that as used by cybersecurity practitioners, "integrity" means that all data on a system should be free from unauthorized manipulation, and that the system should operate as anticipated when deployed, which is made clear by Saltzer and Schroeder's use of the more descriptive term "unauthorized information modification" to describe threats to information integrity.

[39] One can imagine vulnerabilities that stretch the concept of a bug. For instance, if a vendor distributes software packages that use "admin" as a hardcoded default password, it seems odd to consider this a bug. Bugs imply some form of unexpected system behavior, but if a malicious hacker enters "admin" as a password and is granted access to the protected data, nothing has happened that would be unexpected to the developer. And yet this is clearly a vulnerability. These edge cases, however, need not detain us. Most such "vulnerabilities" are trivial to identify and are not the focus of common vulnerability-tracking databases, and in any event, it is standard to treat vulnerabilities as a subclass of bugs. *See* Bojanova & Galhardo, *supra* note 34, at 7.

[40] It is often not clear based only on the behavior of a bug whether it might pose any security concerns, and there is extensive academic research that attempts to help developers identify security-relevant bugs. *See, e.g.*, Katerina Goseva-Popstojanova & Jacob Tyo, *Identification of Security Related Bug Reports via Text Mining Using Supervised and*

certain contexts or configurations, or may become vulnerabilities only after someone discovers a new method of exploiting them.

Finally, an *exploit* is code or a process which makes use of one or more vulnerabilities in order to violate a security constraint embedded in the software.[41] The only important thing to emphasize regarding exploits is that (non-security) bugs can cause system errors spontaneously—or at least without anyone's deliberately intending to produce the erroneous behavior—whereas vulnerabilities in the absence of anyone to exploit them are entirely benign.

B. The Status Quo: (Near-)Immunity in Tort

Software developers do sometimes face liability in tort for flaws in their code. Courts have allowed cases to proceed where a software flaw caused a heart monitor to malfunction and burn a patient's heart;[42] where a computer chip in a truck's fuel delivery system caused the truck to stall in an open intersection;[43] or where a baby monitor's software caused its alarm system to fail, leading to a child's premature death.[44] In all of these cases, however, software errors led to physical injuries. When it comes to economic injuries caused by software bugs—such as business interruptions, data breach remediation costs, ransomware attacks, and so on—software developers face essentially no liability for harms they cause or enable.[45]

The primary reason for this lack of liability is the economic loss rule, which ordinarily prohibits injured parties from suing in tort to recover financial losses caused by a defendant's negligence, at least where there is no related physical injury or property damage.[46] There are two prominent justifications for this rule that apply in different contexts.

Unsupervised Classification, 2018 IEEE Int'l Conf. on Software Quality, Reliability & Sec. 344, 344.

41 *See id.*; Saltzer & Schroeder, *supra* note 38.

42 Singh v. Edwards Lifesciences Corp., 210 P.3d 337, 339 (Wash. App. Div. 1 2009).

43 Gen. Motors Corp. v. Johnston, 592 So. 2d 1054, 1056 (Ala. 1992).

44 Graves v. CAS Med. Sys. Inc., 735 S.E.2d 650, 659 (S.C. 2012) (affirming summary judgment against plaintiffs on the basis of inadequate expert testimony regarding "complex issues of computer science," but acknowledging that a design defect claim with stronger evidentiary support would be appropriate in such a case).

45 *See* Bryan H. Choi, *Crashworthy Code*, 94 Wash. L. Rev. 39, 42 nn.8–10 (2019) (collating cases in which courts dismissed claims arising out of economic harms); *see also* Chinmayi Sharma & 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 [https://perma.cc/PKK5-NS8S] ("Software manufacturers most often face little to no accountability under the law.").

46 *See* Restatement (Third) of Torts: Liab. for Econ. Harm § 1(1) (Am. L. Inst. 2020) ("An actor has no general duty to avoid the unintentional infliction of economic loss on

In the case where a plaintiff and defendant are complete strangers, the rule is often justified on the grounds that holding actors liable for inadvertently caused economic losses would generate expansive or "unlimited" liability which could bankrupt even well-resourced actors as a result of momentary oversights.[47] As between two parties to a contract, however, the justification is different. In that context, courts argue that the economic loss rule is necessary to preserve the distinction between tort and contract by insisting that parties negotiate over remedies for ineffective goods or performances, instead of relying on the tort system for after-the-fact recovery.[48] Notably, however, the economic loss rule is curtailed between contracting parties in the case of professional malpractice—an important observation to which I return in Part III.[49]

In the context of software, however, a rule prohibiting recovery of economic losses comes close to simply immunizing developers from *all* tort liability. Where courts have permitted lawsuits over flaws in code, it is usually only because the code is tightly bound up in the physical control of a tangible product clearly governed by products liability.[50] Most freestanding software, by contrast, cannot control physical systems but can err in various ways that cause economic losses, such as by triggering computer shutdowns, exposing sensitive data, or inadvertently deleting information—all of which cause purely economic injury. Since much software can *only* cause economic injury, the economic loss rule therefore effectively immunizes its developers

---

another."). Note that the economic loss rule does not prohibit suits over economic injuries caused by *intentional* conduct such as fraud or defamation.

47 *See* Catherine M. Sharkey, *In Search of the Cheapest Cost Avoider: Another View of the Economic Loss Rule*, 85 U. CIN. L. REV. 1017, 1027–29 (2018) (discussing the "pragmatic" justification for the economic loss rule as between strangers).

48 For a canonical statement of this purpose of the economic loss rule, see East River S.S. Corp. v. Transamerica Delaval, Inc., 476 U.S. 858, 872–73 (1986) ("Contract law, and the law of warranty in particular, is well suited to commercial controversies of the sort involved in this case [featuring economic losses due to a product malfunction] because the parties may set the terms of their own agreements."); *see also* Jay M. Feinman, *The Economic Loss Rule and Private Ordering*, 48 ARIZ. L. REV. 813, 817 (2006) ("[T]he principal explanation for the rise of the economic loss rule and the related attempt to limit liability to third parties is a preference for private ordering over public regulation."); Ward Farnsworth, *The Economic Loss Rule*, 50 VAL. U. L. REV. 545, 545–56 (2016) (describing the best justification of the economic loss rule as one which prohibits recovery of losses caused by the negligent performance of a contract).

49 *See* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR ECON. HARM § 4 cmt. a (AM. L. INST. 2020) (discussing malpractice liability as a "prominent exception" to the economic loss rule).

50 *See supra* notes 42–44 and accompanying text; *see also* Asaf Lubin, *On Software Bugs and Legal Bugs: Product Liability in the Age of Code*, 100 IND. L.J. 1891, 1905–10 (2025) (comparing cases in which courts have held that software is not a product with those in which courts have reached the opposite conclusion).

from tort suits, whether brought by strangers or contracting parties.[51] No wonder, then, that proponents of software liability typically call for the economic loss rule to be "curtailed" or "abrogated" in the context of software.[52]

The economic loss rule, however, varies considerably between states.[53] Not every state's version of the rule prohibits lawsuits against developers for economic injuries.[54] To further protect themselves, software vendors routinely use contracts or licenses which include disclaimers of implied warranties, disclaimers of tort liability, or express limitations on liability.[55] And, rather than carefully scrutinizing whether these disclaimers are unconscionable, courts routinely find them enforceable and prevent plaintiffs from recovering, even where the disclaimers arise in contracts of adhesion.[56]

It is important to emphasize that both of these barriers to tort liability—the economic loss rule and enforcement of contractual limitations on liability—emerge out of legal doctrines regarding tort *duties*. The economic loss rule is described by the Restatement (Third) of Torts as a categorical limitation on an actor's duty.[57] And where contractual waivers of liability are held to be enforceable, these waivers negate whatever other tort duties a software vendor may have (such as a duty to avoid causing physical injury).

---

[51] *See* Catherine M. Sharkey, *Can Data Breach Claims Survive the Economic Loss Rule?*, 66 DePaul L. Rev. 339, 350–53 (2017) (listing cases applying the economic loss rule in the "stranger paradigm" to prevent recovery in data breach suits).

[52] *See* Jim Dempsey, Standards for Software Liability: Focus on the Product for Liability, Focus on the Process for Safe Harbor 1, 5 (Lawfare, Security by Design Paper Ser., 2024), https://s3.documentcloud.org/documents/24371794/krp-editsdempsey_sbd-paper_final_jan23.pdf [https://perma.cc/R6SR-WZYH] (noting in passing the need to abrogate the economic loss rule as part of a software liability regime); Sharma & Zipursky, *supra* note 45.

[53] *Compare* People Express Airlines v. Consol. Rail Corp., 495 A.2d 107, 109 (N.J. 1985) (describing the economic loss rule as a *per se* rule barring recovery), *with* Tiara Condo. Ass'n, Inc., v. Marsh & McLennan Cos., 110 So. 3d 399, 407 (Fla. 2013) (limiting the economic loss rule only to the context of products liability).

[54] *See, e.g.*, *In re* Target Corp. Data Sec. Breach Litig., 66 F. Supp. 3d 1154, 1170–76 (D. Minn. 2014) (reviewing, at the motion to dismiss stage for a nationwide data breach class action, the economic loss rule in eleven states and concluding that the economic loss rule in six of those states did not facially bar plaintiffs' claims).

[55] *See* Florencia Marotta-Wurgler, *What's in a Standard Form Contract? An Empirical Analysis of Software License Agreements*, 4 J. Empirical Legal Stud. 677, 704–05, 707 (2007) (reporting frequent use of disclaimers or warranties or limitations in liability in standard form software licenses).

[56] *See* Michael D. Scott, *Tort Liability for Vendors of Insecure Software: Has the Time Finally Come?*, 67 Md. L. Rev. 425, 434–41 (2008) (collating cases in which courts determined that software is governed by the Uniform Commercial Code and enforced limitations of liability); *see also* Dempsey, *supra* note 52, at 4–5 (discussing the barriers posed by waivers of liability in form contracts); Sharma & Zipursky, *supra* note 45 (same).

[57] *See supra* note 46.

None of this is to say that software vendors have no legal obligations when it comes to the quality of their code, even as regards economic losses. Both the Federal Trade Commission (FTC) and the Securities and Exchange Commission (SEC) have exercised their regulatory powers following high-profile cybersecurity incidents or data breaches to penalize companies who did not reasonably secure their IT systems—though these are forms of public enforcement that serve no compensatory purpose for injured parties.[58] Although these actions are sporadic, a review of FTC settlements suggests that the agency has insisted on a consistent but adaptable set of "minimum" cybersecurity practices, even though its authority to regulate cybersecurity practices remains in doubt.[59] Via executive order, the Biden administration imposed security attestation obligations on federal contractors, violation of which may raise liability under the False Claims Act.[60] And of course, if software vendors do explicitly warrant the security or usability of their goods, disappointed consumers may always bring a claim for breach of contract.[61] But since very few software vendors are so foolish, the vast majority of economic injuries caused by defective software currently generate no liability for the software vendor.

C. Proposals for Reform

This status quo is not exactly beloved, except perhaps by the software industry. Legal commentators have been calling for reform for decades.[62] Leaders in the field of cybersecurity, frustrated by the lack of resources that vendors allocate to security, have also advocated

---

[58] *E.g.*, FTC v. Wyndham Worldwide Corp., 799 F.3d 236, 243–49 (3d Cir. 2015) (accepting, on appeal from a motion to dismiss, that at least some forms of inadequate cybersecurity may constitute "unfair business practices"); SEC v. SolarWinds Corp., 741 F. Supp. 3d 37, 48–52 (S.D.N.Y. 2024) (finding that misleading statements of cybersecurity practices are actionable under the Securities Exchange Act, but rejecting the SEC's claim that existing statutes impose substantive cybersecurity requirements).

[59] *See* Isabella Wright & Maia Hamin, Atl. Council, "Reasonable" Cybersecurity in Forty-Seven Cases: The Federal Trade Commission's Enforcement Actions Against Unfair and Deceptive Cyber Practices 15–16, 27–29 (Samia Yakub ed., 2024).

[60] Exec. Order No. 14,028, 86 Fed. Reg. 26633, 26637–41 (May 17, 2021); *see also* Steven B. Lipner, *Incentives for Improving Software Security: Product Liability and Alternatives*, Lawfare (May 14, 2024), https://www.lawfaremedia.org/article/incentives-for-improving-software-security-product-liability-and-alternatives [https://perma.cc/LL7Z-W83P].

[61] *But see* Scott, *supra* note 56, at 437 ("No reported decision has unequivocally held that a software vendor has breached an express warranty.").

[62] *See, e.g.*, Susan Nycum, *Liability for Malfunction of a Computer Program*, 7 Rutgers Comput. & Tech. L.J. 1 (1979); Stephen E. Henderson & Matthew E. Yarbrough, *Suing the Insecure? A Duty of Care in Cyberspace*, 32 N.M. L. Rev. 11 (2002); Emily Kuwahara, Note, *Torts v. Contracts: Can Microsoft Be Held Liable to Home Consumers for Its Security Flaws?*, 80 S. Calif. L. Rev. 997 (2007); Scott, *supra* note 56; Choi, *supra* note 45.

for greater liability exposure.[63] So far, however, courts and legislators have paid relatively little attention to this issue, and the status quo has continued largely unperturbed.

But bipartisan dissatisfaction is growing. In 2019, Congress created the Cyberspace Solarium Commission with a mandate to draw up a comprehensive strategy to guide the U.S. government's response to growing cyber threats. The Commission's final report, released in March 2020, proposed that "Congress should pass a law establishing that final goods assemblers of software, hardware, and firmware are liable for damages from incidents that exploit known and unpatched vulnerabilities."[64] And in March 2023, the Biden administration released a National Cybersecurity Strategy that called for "legislation establishing liability for software products and services."[65] These moves sparked a new round of commentary. Unlike previous discussion, this debate has attracted heavy interest not only from academics writing in law reviews,[66] but also from think tanks[67] and public-facing venues like *Lawfare*.[68] Although the Trump administration has yet to indicate any particular interest in the topic, commentary has not fully disappeared.[69] Ultimately, it will likely take another high-profile cyberattack affecting the U.S. government to determine whether the Trump administration will be similarly receptive to calls for software liability.[70] To an overwhelming

---

[63] *E.g.*, Bruce Schneier, *Computer Security and Liability*, Schneier on Sec. (Nov. 3, 2004), https://www.schneier.com/blog/archives/2004/11/computer_securi.html [https://perma.cc/3LCM-SZU8]; Sara Peters, *Dan Geer Touts Liability Policies for Software Vulnerabilities*, DarkReading (Aug. 6, 2014), https://www.darkreading.com/vulnerabilities-threats/dan-geer-touts-liability-policies-for-software-vulnerabilities [https://perma.cc/AN9A-WNJN].

[64] Cyberspace Solarium Comm'n, Final Report 76 (2020).

[65] White House, National Cybersecurity Strategy 20–21 (2023).

[66] *E.g.*, Charlotte A. Tschider, *Locking Down "Reasonable" Cybersecurity Duty*, 41 Yale L. & Pol'y Rev. 75, 75 (2023).

[67] *E.g.*, Trey Herr, Robert Morgus, Stewart Scott & Tianjiu Zuo, *Buying Down Risk: Cyber Liability*, Atl. Council (May 3, 2022), https://www.atlanticcouncil.org/content-series/buying-down-risk/cyber-liability [https://perma.cc/2A67-C5F9]; Maia Hamin, Sara Ann Brackett & Trey Herr, Atl. Council, Design Questions in the Software Liability Debate (2024), https://www.atlanticcouncil.org/in-depth-research-reports/report/design-questions-in-the-software-liability-debate [https://perma.cc/T6B5-42J8].

[68] The focal point for much of the debate has been a series of essays and blog posts published by *Lawfare*. *See Security by Design*, Lawfare (July 24, 2024), https://www.lawfaremedia.org/projects-series/lawfare-research-intiative/security-by-design [https://perma.cc/3LMA-2DKB].

[69] *See, e.g.*, Jim Dempsey, *The MAGA Case for Software Liability*, Lawfare (Feb. 19, 2025), https://www.lawfaremedia.org/article/the-maga-case-for-software-liability [https://perma.cc/AV3H-6AJ9].

[70] The Trump administration significantly revoked much of an executive order issued at the very end of the Biden administration which imposed more extensive cybersecurity requirements on federal contractors. *See Fact Sheet: President Donald J. Trump Reprioritizes Cybersecurity Efforts to Protect America*, White House (June 6, 2025), https://www.

extent, the commentary of the past half-decade has focused on two key issues. First, should software liability take the form of products liability or a negligence standard? Abstracting considerably, the products liability approach would focus attention on the software code itself, analyzing whether a harm could have been eliminated by a "reasonable alternative design" that did not include a particular flaw.[71] A negligence approach would instead focus on the processes used by the developer, asking whether the developer expended reasonable effort to catch foreseeable flaws, where "reasonability" would require evaluation of factors including professional custom and the risk associated with a particular piece of software.[72]

Proponents of the products liability frame suggest that the notion of "design defect" is a natural fit for thinking about vulnerabilities in code, and that products liability comes with frameworks to structure an inquiry into a developer's responsibility for complex design decisions.[73] Some proponents also draw on products liability's doctrine of strict liability for product malfunctions to argue that developers should be subject to strict liability for violating certain worst practices.[74] Advocates of a negligence frame, by contrast, argue that code is an enormously complex and highly differentiated process that requires a more freestanding reasonableness inquiry into a developer's conduct.[75] Some further emphasize that cybersecurity may also be likened to a profession, and suggest that software liability should be modeled on (negligence-based) professional malpractice standards.[76]

The second issue concerns whether liability, within either frame, should be limited by safe harbors. Many commentators, concerned that liability may be too unpredictable without some sort of safe harbor to protect developers who follow best practices and are nonetheless

---

whitehouse.gov/fact-sheets/2025/06/fact-sheet-president-donald-j-trump-reprioritizes-cybersecurity-efforts-to-protect-america [https://perma.cc/NLH8-SBSQ] (criticizing the Biden executive order for "[i]mposing unproven and burdensome software accounting processes that prioritized compliance checklists over genuine security investments"). While this move indicates less interest in using the purchasing power of the federal government to force *particular* security practices among contractors, and while it is likely that the Trump administration has little appetite for new business liability regimes, as of this writing there have been no high-profile cyberattacks during the second Trump administration which might seriously test this pro-business outlook.

71 *See* Sharma & Zipursky, *supra* note 45. For a critique that this approach would necessarily collapse into a process-oriented evaluation that would resemble a negligence inquiry, see *infra* Section III.C.

72 *See, e.g.*, Bryan H. Choi, *Software as a Profession*, 33 HARV. J.L. & TECH. 557, 564 (2020).

73 *See* Sharma & Zipursky, *supra* note 45; *see also* Scott, *supra* note 56, at 457–71.

74 *E.g.*, DEMPSEY, *supra* note 52, at 11–15.

75 *See, e.g.*, Choi, *supra* note 72, at 567–86; *see also* Scott, *supra* note 56, at 441–57.

76 *See* Choi, *supra* note 72, at 622–33; *see also infra* Part III.

attacked by sophisticated hackers, believe it should.[77] Proponents of software liability have expended enormous energy trying to more precisely define what might constitute "reasonable" software development,[78] or how a safe harbor might be defined.[79] For example, a liability regime might immunize developers who follow secure coding practices, or who verify before release that their code does not contain any known vulnerabilities as tracked by the National Vulnerability Database.[80] Key issues here involve debates over how detailed safe harbors or development guidance can feasibly be,[81] whether compliance with a safe harbor would be treated as a factual dispute during litigation or could be certified in advance,[82] and how standards would be adapted over time to remain up to date with changes in the threat landscape.[83]

---

[77] *See* White House, *supra* note 65, at 21 (describing Biden administration's plan to support development of a safe harbor framework); Tschider, *supra* note 66, at 80–85 (proposing a dynamic and highly contextual approach to liability).

[78] *E.g.*, Tschider, *supra* note 66, at 87–88 (arguing for the importance of formally describing "reasonable" cybersecurity practices).

[79] *See* Dempsey, *supra* note 52, at 15–16 (drawing elements from National Institute of Standards and Technology (NIST) and Microsoft guidance documents as potential origins for a safe harbor).

[80] *See* Derek E. Bambauer & Melanie J. Teplinsky, *Standards of Care and Safe Harbors in Software Liability: A Primer*, Lawfare (May 31, 2024), https://www.lawfaremedia.org/article/standards-of-care-and-safe-harbors-in-software-liability-a-primer [https://perma.cc/T8YJ-XRWM] (discussing desiderata of a safe harbor for developers); Jane Chong, *The Challenge of Software Liability*, Lawfare (Apr. 6, 2020), https://www.lawfaremedia.org/article/challenge-software-liability [https://perma.cc/DH79-58D2] (distinguishing liability for "unknown and unforeseeable vulnerabilities" from those based on known vulnerabilities like those in the National Vulnerability Database).

[81] *Compare* Dempsey, *supra* note 52, at 9–11 (advocating for an approach modeled off the granularity of building codes to set a floor for software quality), *and* Derek E. Bambauer & Melanie J. Teplinsky, *Shields Up for Software*, Lawfare (Dec. 19, 2023), https://www.lawfaremedia.org/article/shields-up-for-software [https://perma.cc/X8N6-SWTL] (proposing an "inverse safe harbor" based on the Common Weaknesses Enumeration resource), *with* Choi, *supra* note 72, at 567–86 (suggesting that software is too complex to benefit from formal evaluation metrics), *and* Lipner, *supra* note 60 (discussing repeated historical failures to develop systematic security evaluation schemes).

[82] For an argument in favor of a safe harbor certification process, see Charlotte A. Tschider, Will a Cybersecurity Safe Harbor Raise All Boats? 13–20 (2024), https://s3.documentcloud.org/documents/24489356/sbd-tschider_final_3-15-1.pdf [https://perma.cc/Y5TX-2HA3].

[83] Possible solutions involving giving the authority to maintain and update a safe harbor to an agency, including NIST, the Cybersecurity and Infrastructure Security Agency (CISA), or even the FTC. *See* Dempsey, *supra* note 52, at 16 (discussing NIST and CISA standards agencies in relation to the creation of a safe harbor); Wright & Hamin, *supra* note 59, at 28–29 (suggesting that the FTC is "designed and primed" to handle the questions arising from cybersecurity harm).

Scholars have come up with many creative ways of addressing both issues.[84] But even though the two major barriers to tort liability concern limitations on developers' *duty* to consumers, these debates focus on the *standard of care* by which developers' conduct should be judged. These are meaningfully different issues. Observe that the economic loss rule (at least regarding harms caused to strangers) is concerned about avoiding *unlimited* liability, whereas safe harbors seek to prevent *unpredictable* liability. The latter is a concern about the clarity regarding the type of conduct which will trigger the possibility of liability to anyone, or in other words, a concern regarding the standard of care to which developers should be held. But the former is a concern about whether a developer who has concededly and negligently failed to remediate a flaw should nonetheless be obligated to pay for every resulting injury, no matter the type of harm that results or the remoteness of the injured party. Judicial concerns about overly expansive duties take the form of no-duty rules, which either prohibit recovery on the basis that actors do not have a duty to avoid causing certain types of harms, or that they do not owe duties to certain types of people.[85] When it comes to these types of limitations on tort liability, proponents of reform have little to say: Leading advocates simply assert that Congress should "abrogate" the economic loss rule or make waivers of liability in software contracts uniformly unenforceable.[86]

---

84 Jim Dempsey, for instance, has proposed a three-part inquiry which blends strict liability (automatic liability for software products violating pre-specified worst practices), a process-based safe harbor for developers using secure coding practices, and a generalized products liability framework for residual cases. *See* DEMPSEY, *supra* note 52, at 2.

85 The Restatement (Third) of Torts states that actors ordinarily have a duty to exercise "reasonable care" if their conduct creates a risk of harm, but that "an articulated countervailing principle or policy" may lead courts to deny that a defendant had a duty across a class of cases. *See* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 7 (AM. L. INST. 2010). This account gives duty a relatively narrow role to play in the analysis of tort, with many limitations on liability being reclassified as failures of proximate causation (which the Restatement relabels as considerations regarding a defendant's "scope of liability"). *Compare id.* cmt. a, *with* John C. P. Goldberg & Benjamin C. Zipursky, *The* Restatement (Third) *and the Place of Duty in Negligence Law*, 54 VAND. L. REV. 657, 669 (2001) (critiquing a draft of the Restatement (Third) for treating no-duty rules as exceptional and not as a required element of every tort claim). But even on this narrow account of "duty," the major types of limitations on liability addressed throughout this Note—limiting software liability to harms caused by vulnerabilities, excluding economic losses from recovery, or limiting liability to parties in near-privity—are all categorical limitations on liability which the Restatement (Third) would classify as no-duty rules, not as issues involving proximate causation or scope of liability. This fact underscores the important role which the concept of duty can play in designing a new liability regime, and the dangers of treating "duty" as a peripheral matter meriting less attention than defining a standard of care.

86 For examples of proponents indicating a desire to "abrogate[]" or "relax" the economic loss rule without any further details, see DEMPSEY, *supra* note 52, at 4–5, and Sharma & Zipursky, *supra* note 45. For similarly blunt statements regarding waivers, see Sharma &

And yet, many proponents of software tort reform embed one major no-duty rule in their proposals: They either explicitly argue that liability should only be tied to vulnerabilities (not other types of bugs),[87] or they offer proposals that make no mention of non-security bugs.[88] The Cyberspace Solarium Commission's legislative proposal, for instance, would have facially excluded liability for harms caused by non-security bugs.[89] And although the National Cybersecurity Strategy made a passing reference to the need to "prevent bad outcomes," it discussed liability only in terms of harms arising from "insecure software" or "vulnerable products."[90] Commentators in public-facing venues have uniformly discussed liability in terms of "cyberattacks," "improved cybersecurity," or "cybersecurity threat."[91] And academics advocating for various types of software liability have similarly focused their attention on security, as opposed to other types of potential software flaws.[92]

While at least some proposals would facially exclude harms from non-security bugs from tort recovery, it is not clear how many proponents of software liability would formally endorse such a limitation. Some proponents simply appear to speak in terms of "security" because they view this focus as a way of implicitly highlighting the most important types of software harms.[93] Alternatively, some analysts may feel that they are responding to a particular set of national security needs, and that liability for bugs would either be secondary or irrelevant to those goals.[94] Other proponents discuss liability in ways that emphasize "security" while drawing from examples concerning non-security bugs,

---

Zipursky, *supra* note 45 ("[S]uccessful legislation will stipulate that software is to be treated as a product and liability waivers are unenforceable.").

87 *E.g.*, Dempsey, *supra* note 52, at 15 ("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 . . . ."); *see also supra* note 64 and accompanying text.

88 *See, e.g.*, Sharma & Zipursky, *supra* note 45 (ignoring scenarios arising from the "introduction of a bug" within the proposed liability framework). One exception is Bambauer & Teplinsky, *supra* note 80, which opens by discussing the difficulty of creating "bug-free software." But rather than discuss liability that might arise from software bugs generally, Bambauer and Teplinsky proceed to illustrate their discussion only with examples of harms that arise when "software is hacked."

89 *See supra* note 64 and accompanying text.

90 White House, *supra* note 65, at 20–21.

91 *E.g.*, Sharma & Zipursky, *supra* note 45; Dempsey, *supra* note 52, at 1–2.

92 *See, e.g.*, Derek E. Bambauer, *Cybersecurity for Idiots*, 106 Minn. L. Rev. 172, 179–80 (2021) (advocating for a regulatory regime of strict liability for cybersecurity "worst practices" which could generate liability even in the absence of any harm); Tschider, *supra* note 66, at 83, 97 n.81, 109 nn.131, 132–33 & 142 (linking "harm," throughout, to injuries caused by data breaches or cyberattacks).

93 *See, e.g.*, Dempsey, *supra* note 52, at 1 (stating that "the intent is not to impose liability for all flaws but only for some especially consequential ones").

94 This possibility and its implications are discussed in *infra* Section II.D.

making it difficult to tell whether they intend their proposals to cover liability for *all* bugs or only vulnerabilities.[95] The point is that recent proponents of software liability simply have not given much thought to whether liability is meant to cover only vulnerabilities or harms flowing from every type of software bug. To the extent that at least some view vulnerabilities as the focus of their proposals, they in effect propose the replacement of one type of no-duty rule—the economic loss rule—with another one that excludes liability for non-security bugs. But is such a no-duty rule a coherent way to limit developers' liability, and does it make sense to treat vulnerabilities as different or more central to any proposed liability regime than other types of bugs?

II
Vulnerabilities as a Limitation on Liability

This Part argues that restricting developers' liability to harms caused by vulnerabilities—and not other types of software bugs—is a poor way to limit liability. I make this argument in three ways. First, in Section II.A, I discuss whether it is technically possible to neatly separate liability for bugs from liability for vulnerabilities. Second, in Sections II.B and II.C, I point out that liability that only attaches to exploited vulnerabilities is doctrinally strange. On the one hand, it might be thought of as a form of liability that requires developers to guard against foreseeable criminal conduct, which would make it analogous to affirmative duties owed by landlords or storeowners to their tenants and patrons. But several features of existing software liability proposals make them difficult to justify as an affirmative duty along these lines. On the other hand, if we were to treat software as a product subject to products liability standards, then it is doctrinally odd to exclude liability for harms caused by non-security bugs. Finally, in Section II.D, I examine the incentives facing software developers and suggest that liability that attaches only to vulnerabilities would distort their incentives in ways that would likely undermine consumer welfare. This Part thus offers a technical critique, a doctrinal critique, and a policy critique of any attempt to limit developers' potential liability by focusing only on vulnerabilities, as well as a warning that proposals which rhetorically focus on "security" may nonetheless carry much more expansive liability for non-security bugs.

---

[95] *See, e.g.*, Scott Shackelford, Janine Hiller, Christos Makridis, Iain Nash, Kathryn Kisska-Schulze & Hannibal Travis, *Moving Slow and Fixing Things*, 100 Ind. L.J. 1611, 1615–16 (2025) (using the CrowdStrike bug as an example of the need for proactive cybersecurity and liability that encourages more secure software development, while acknowledging that it was "not a breach in itself").

A. The Inherent Ambiguity of a "Vulnerability"

What does it mean to say that developers ought to have a duty to guard against "insecure" software? This Section discusses three ambiguities arising from the technical features of software that make it difficult to tell just how far-reaching existing liability proposals are meant to be.

First, consider what it means to link liability to harms "caused or enabled by a . . . vulnerability."[96] Does this mean harms caused by an exploit*ed* vulnerability or an exploit*able* vulnerability? Where they discuss this issue, most proponents require actual exploitation on the basis that only exploitation of a vulnerability can create a predicate harm cognizable by the tort system.[97] But this is not so, at least in the absence of the economic loss rule. Simply discovering a vulnerability triggers an often costly process of patching and mitigation, whether or not a malicious hacker is actually present. For instance, in 2021, a vulnerability was discovered in Log4j, an open source library used to perform system logging that was used at the time by as many as ninety-three percent of all cloud servers.[98] The vulnerability was so severe and so widespread that, according to the Cyber Safety Review Board, a single federal agency spent 33,000 hours mitigating it.[99] These costs were certainly *caused* by the developer's potentially negligent failure to eliminate a vulnerability from their code. But they look a good deal like the cost of repairing products, and absent contractual warranties to the contrary, consumers are generally expected to bear repair costs.[100]

Second, it is not always clear what counts as a vulnerability. This is most often due to the fact that, as discussed in Section I.A, some bugs only become vulnerabilities in certain system configurations, or upon the discovery of a valid exploit.[101] Sometimes, however, the conceptual

---

[96] CSC Proposal, *supra* note 7, at 117.

[97] *E.g.*, Dempsey, *supra* note 52, at 15 ("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 . . . ."). *But see* Bambauer, *supra* note 92, at 176 (advocating a regulatory approach that "would impose liability when entities deviate below regulatory minima even in the absence of harm").

[98] *See* Ami Luttwak & Alon Schindel, *Log4Shell 10 Days Later: Enterprises Halfway Through Patching*, Wiz (Dec. 20, 2021), https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell [https://perma.cc/V42U-2DRZ].

[99] Cyber Safety Rev. Bd., Review of the December 2021 Log4j Event 17 (2022).

[100] *E.g.*, E. River S.S. Corp. v. Transamerica Delaval, Inc., 476 U.S. 858, 870 (1986) ("Even where the harm to the product itself occurs through an abrupt, accident-like event, the resulting loss due to repair costs, decreased value, and lost profits is essentially the failure of the purchaser to receive the benefit of its bargain—traditionally the core concern of contract law.").

[101] After an exploit is discovered, the flaw—which may or may not have previously been noticed and dismissed as a non-security bug—becomes retroactively referred to as a

distinction between bugs and vulnerabilities breaks down in more profound ways. Consider the generation of "adversarial examples," which seeks to systematically generate new inputs to AI systems that will cause unexpected and erroneous behavior.[102] Is this a form of hacking? Legal scholars view this as a difficult question and suggest that it is unclear whether the Computer Fraud and Abuse Act (the United States' primary federal anti-hacking statute) might criminalize adversarial attacks.[103] But typically exploiting a vulnerability is understood to imply the circumvention of some sort of formal security constraint, an assumption which is not true in the context of adversarial attacks on AI systems.[104] Imagine that an autonomous vehicle crashes because a few pieces of tape tricked it into perceiving a stop sign as a "Speed Limit 45" sign.[105] It is easy to think of this as a failure at an edge case unanticipated by a developer and thus as a bug, but it remains unclear whether it should also be considered a vulnerability and the tape an exploit.[106]

"zero-day vulnerability." The name refers to the fact that when an exploit is first discovered, it can be deployed with "zero days" available to prepare defenses. *See* Kim Zetter, *Hacker Lexicon: What Is a Zero Day?*, Wired (Nov. 11, 2014), https://www.wired.com/2014/11/what-is-a-zero-day [https://perma.cc/7GEE-3FCE].

102 For a technical description of adversarial examples, see, for example, Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, Explaining and Harnessing Adversarial Examples (Mar. 20, 2015) (arXiv manuscript), https://arxiv.org/pdf/1412.6572 [https://perma.cc/A3M6-TEMA].

103 *See, e.g.*, Ram Shankar Siva Kumar, Jonathon Penney, Bruce Schneier & Kendra Albert, Legal Risks of Adversarial Machine Learning Research (June 29, 2020) (arXiv manuscript), https://arxiv.org/pdf/2006.16179 [https://perma.cc/4P49-PQUQ]; Kendra Albert, Jonathon Penney & Ram Shankar Siva Kumar, *Ignore Safety Directions. Violate the CFAA?*, Black Hat (July 27, 2024), https://i.blackhat.com/BH-US-24/Presentations/US-24-Albert-IgnoreYourGenerativeAISafetyInstructions-Thursday.pdf [https://perma.cc/AU34-KXZP].

104 *See* Kumar et al., *supra* note 103, at 7. For the purposes of interpreting the phrase "unauthorized access" in the CFAA, the Supreme Court has directed courts to conduct a "gates-up-or-down inquiry" to evaluate whether a hacker violated some formal security constraint. Van Buren v. United States, 141 S. Ct. 1648, 1658 (2021). It seems to follow that placing tape on a stop sign could not constitute "unauthorized access" to a vision model in an autonomous vehicle, because the action does not bypass any "gate" on the model. For a piece nonetheless analyzing attacks on AI systems through the lens of the traditional CIA triad for cybersecurity vulnerabilities, see Andrew J. Lohn, Ctr. for Sec. & Emerging Tech., Hacking AI (2020).

105 *See* Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno & Dawn Song, *Robust Physical-World Attacks on Deep Learning Visual Classification*, 2018 IEEE Conf. on Comput. Vision & Pattern Recognition 1625, 1626 (2018) (sharing results from research project enacting this exact road sign scenario).

106 *But cf.* Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran & Aleksander Madry, *Adversarial Examples Are Not Bugs, They Are Features*, 33 Conf. on Neural Info. Processing Sys. (2019) (suggesting that adversarial attacks are not bugs either, though without defining a "bug" for the purposes of the claim).

Finally, the prospect of patches introduces additional puzzles. A vendor who notices a vulnerability might release a patch to correct it, but the code in that patch could introduce new bugs (including vulnerabilities), or it might simply alter the performance of the underlying software program. In one example, Intel identified a vulnerability in some of its CPUs which could only be fixed by adopting a patch that slowed the devices' performance by up to fifty percent.[107] A number of consumers viewed this cure as worse than the disease, leading them to file a class action lawsuit alleging negligence, which was eventually dismissed on the basis of the economic loss rule.[108] If the consumers accepted the patch—which was only pushed by the vendor to address its prior failure to spot a vulnerability—but were saddled with a computing device less capable than the one they bargained for, were they harmed in a way that the tort system should recognize? What if they refused the patch, only to subsequently be hacked? Should they fully be denied recovery on the basis of assumption of risk or comparative fault?

FIGURE 1: SOURCES OF SOFTWARE HARMS

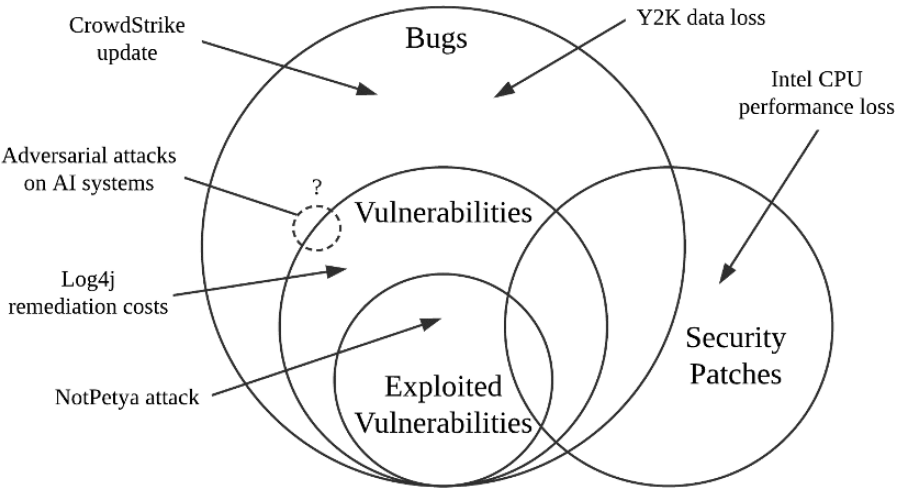

Figure 1 summarizes this discussion. It highlights that harms—which in each of these case studies take the form of economic loss—can

107 *See* Amended Class Action Complaint & Demand for Jury Trial at 40–42, Smith v. Intel Corp., 745 F. Supp. 3d 853 (N.D. Cal. 2024) (No. 23-cv-057610). Note that this outcome was deemed a necessary tradeoff by the developer, so the reduced performance after the patch was not unexpected and therefore was not a bug.

108 *See id.* at 81–82 (alleging that Intel's defective design of its original software and its patch caused diminished battery life and decreased life expectancy of Intel products); *Smith*, 745 F. Supp. 3d at 865 (dismissing negligence claims because physical damage to the computer is not "analytically separate" from damage to the CPUs themselves).

be caused by bugs, vulnerabilities (even without a hacker), exploited vulnerabilities, or by a security patch. Anything within *either* the circle marked "vulnerabilities," *or* the circle marked "security patches" is clearly caused by a vulnerability.[109] Existing software liability proposals do not say anything about the scope that a developer's duty should take once the economic loss rule is abrogated and waivers of liability are made unenforceable. As a result, it is difficult to tell which injuries marked in the figure are intended to fall within a developer's duty to guard against, and which are not.

All of these issues press on ambiguities about what it means to say that vendors should be liable for vulnerabilities. But it is critical to keep in mind that ordinary bugs can cause many of the same harms as vulnerabilities.[110] For example, as discussed in the Introduction, the global effects of both CrowdStrike's buggy update and the NotPetya attack were remarkably similar.[111] This is a general point. A disabled hospital IT system could be due to ransomware or a system misconfiguration; an autonomous vehicle crash could be due to an adversarial manipulation or a spontaneous error; a degraded battery could be due to cryptojacking malware or an overloaded CPU. To simply assert that liability should only attach to vulnerabilities is to draw an arbitrary distinction between harms that are compensable and those that are not. As the next Sections argue, this lack of attention to the issue of bugs is more than simply a trivial oversight. Rather, it cuts to the core of debates over the purposes and formulation of any software liability regime.

---

109 This is true in the sense of but-for causation. Whether a vulnerability is a proximate cause of performance issues introduced by a patch is a distinct legal question, but one which to date has received no attention from proponents of software liability.

110 There are some categories of harms which are admittedly much more likely to be caused by vulnerabilities than other types of bugs. In particular, where the harm is caused by *exposure* of data—such as the harms of identity theft or espionage—the harm itself seems to require exposure *to* someone, who will most often be a malicious actor seeking to exploit a vulnerability. As a rough heuristic, it is probably the case that harms entailing a loss of confidentiality will be associated primarily with vulnerabilities, whereas harms entailing a loss of data integrity or availability could be spontaneously caused by a bug just as plausibly as they might be deliberately caused by a hacker. *See* Saltzer & Schroeder, *supra* note 38, at 1280 (describing the CIA triad). In theory, this might offer a rationale for restricting liability to vulnerabilities only, but I am aware of no proponent of software tort liability who has suggested that confidentiality-based harms are somehow different in kind or more important than integrity- or availability-based harms. *See supra* Section I.C (discussing the existing field's focus on the standard of care to be used in evaluating software to the exclusion of issues concerning duty or proximate causation). In fact, many proponents base their calls for liability on the prospect of damages to the *integrity* of data, especially in computer systems controlling critical infrastructure. *See, e.g.*, Dempsey, *supra* note 69.

111 *See supra* notes 1–6 and accompanying text.

B. Foreseeable Criminal Conduct

Proponents of software liability have said a great deal about the need to minimize vulnerabilities in software, but remarkably little about the hackers who exploit them.[112] Insofar as the focus of software liability remains on (exploited) vulnerabilities, this is somewhat odd, as every cognizable injury would be the result of at least two actors' actions: the software developer who introduced the vulnerability and the hacker who exploited it.

Reframed this way, it becomes surprising that little has been said about the relationship between software liability and other actors who have affirmative duties to prevent foreseeable criminal conduct. Could it be useful to analogize the developer of a software component to a landlord or storeowner who has a duty to protect tenants or patrons? These are useful points of analogy because it is rare within tort law for an actor to have an obligation to anticipate and guard against the malicious conduct of a third party, and ordinarily such duties are classified as "affirmative" duties that are considered departures from ordinary tort assumptions.[113]

The genesis of a landlord's duty to take reasonable precautions to protect tenants from third-party criminal conduct is often attributed to *Kline v. 1500 Massachusetts Avenue*.[114] In that case, a defendant landlord was found liable for a tenant's assault that occurred after the landlord, knowing about a rash of assaults in common areas, stopped employing a doorman.[115] Many of the justifications given for imposing a duty in that case mirror reasons to impose liability on software developers for third-party hacks: The developer often has the "exclusive power" to take preventative measures and the customer has in some way *submitted* to the control of the developer by entrusting valuable assets (such as personal data) to their care.[116] But the existence of a duty found by

---

112 Some authors do acknowledge that a subset of hacks are caused by nation-state actors or advanced persistent threats (APTs), and there is typically agreement that attacks by such sophisticated actors are exceptionally hard to prevent. *See, e.g.*, Tschider, *supra* note 66, at 77–78 (acknowledging that organizations hacked by APTs—and perhaps *only* such organizations—are not "bad actors").

113 *See* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 40 cmts. j, m (AM. L. INST. 2010) (discussing duties of landlords and storeowners to guard against third-party criminal conduct as a type of affirmative duty).

114 439 F.2d 477, 483–85 (D.C. Cir. 1970); *see also* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 40 cmt. m (AM. L. INST. 2010) (discussing the status of *Kline* as the origin for landlord duties).

115 439 F.2d at 481.

116 *Id.* at 481, 483. Other justifications, however, such as an implied contract between the landlord and tenant, do not generalize to software due to the ways that courts have interpreted software licenses. *See supra* Section I.B.

the court in *Kline* was not a generalized duty to mitigate all *possible* crimes; the court emphasized that the extent of the landlord's duty was linked to its specific knowledge of already occurring criminal conduct.[117] Courts imposing a duty on businesses to protect their patrons similarly require a detailed showing of the foreseeability of particular types of criminal conduct when evaluating the scope of a business's duty to guard against such conduct.[118]

If we take these canonical affirmative duties to prevent foreseeable criminal conduct as a starting point, it becomes clear that aspects of current software liability proposals are difficult to accommodate within existing tort theory. Courts following *Kline* closely tie the scope of the landlord's duty to the foreseeability of specific criminal conduct.[119] And in order to perform that highly context-specific inquiry, courts rarely depart from ordinary negligence evaluations.[120] In many ways, this parallels a core objection that many cybersecurity practitioners have raised when confronted by the prospect of cyber liability: Even seemingly clear-cut worst practices may be entirely reasonable in certain situations where the risk of exploitation is sufficiently low.[121] Indeed, virtually all commonly used guidance documents regarding cybersecurity practices emphasize that a risk management approach is necessary for any organization to determine which guidance to follow.[122] If a liability regime does not allow for inquiries into the actual degree of risk associated with a particular software component, and instead relies on pre-described formulations of how developers should act, it will impose costly obligations that in many situations have little to no security benefit.

---

117 *See* 439 F.2d at 483.

118 *See, e.g.*, Posecai v. Wal-Mart Stores, Inc., 99-1222, p. 9 (La. 11/30/99), 752 So. 2d 762, 768 ("A very high degree of foreseeability is required to give rise to a duty to post security guards, but a lower degree of foreseeability may support a duty to implement lesser security measures . . . .").

119 *See supra* notes 117–18 and accompanying text.

120 *See, e.g.*, *Posecai*, 99-1222, pp. 8–9, 752 So. 2d at 768 (balancing the foreseeability and gravity of the harm based on the facts and circumstances of the case).

121 *See* Lipner, *supra* note 60 (describing the limited usefulness of lists of "must-fix" security flaws).

122 *E.g.*, The Software All., The BSA Framework for Secure Software 8 (2020), https://www.bsa.org/files/reports/bsa_framework_secure_software_update_2020.pdf [https://perma.cc/5EST-J42Y] ("[O]rganizations should build software development processes around careful analysis of the risks associated with their products . . . ."); Murugiah Souppaya, Karen Scarfone & Donna Dodson, Nat'l Inst. of Standards & Tech., Secure Software Development Framework Version 1.1, at vi (2022) ("Not all practices are applicable to all use cases; organizations should adopt a risk-based approach to determine what practices are relevant, appropriate, and effective to mitigate the threats to their software development practices.").

More generally, cases like *Kline* illustrate that proponents of software liability appear to have started by taking on a doctrinally hard problem (vulnerabilities) while ignoring its easier counterpart (bugs). If a software liability regime only covers harms caused by hackers exploiting vulnerabilities, it would essentially create an "enabling tort" which punishes developers for enabling *others* to cause harms. Enabling torts, however, are deeply controversial—including among some authors who would attach software liability to the existence of a vulnerability.[123] Additionally, in cases like *Kline*, courts confronted liability for landlords against a backdrop that could safely assume a landlord would be liable for injuries to tenants if they negligently caused the injuries themselves, instead of merely enabling others to do so. But as discussed in the Introduction and Section II.B, this is precisely the type of assumption that does *not* hold for software liability—at least as long as developers are not liable for bugs that cause the same types of injuries that hackers can.

These considerations point to a key conclusion: When liability is attached to harms caused by exploited vulnerabilities and focus is shifted from the fact that software is insecure to the existence of a malicious hacker, it becomes clear that products liability and detailed ex ante safe harbors are poor fits for the duty.[124] If liability is tied to actual exploitation, then an appropriate liability regime should focus on the reasonableness of a developer's actions in light of the foreseeability of a particular type of attack.[125] Without room for such an inquiry, any attempt to define minimum adequate cybersecurity practices will necessarily be both over- and under-inclusive: over-inclusive, because in many situations it would require costly measures to secure software

---

123 *Compare, e.g.*, John C.P. Goldberg & Benjamin C. Zipursky, *Intervening Wrongdoing in Tort: The* Restatement (Third)*'s Unfortunate Embrace of Negligent Enabling*, 44 WAKE FOREST L. REV. 1211, 1218–31 (2009) (marshalling expansive evidence to show courts' continued resistance to enabling torts), *with* Sharma & Zipursky, *supra* note 45 (endorsing products liability for software while referring only to risks of "cyberattacks").

124 A focus on the mere fact of insecure software can be misleading because all software is riddled with an unknown number of undiscovered vulnerabilities. *See generally* O.H. Alhazmi, Y.K. Malaiya & I. Ray, *Measuring, Analyzing, and Predicting Security Vulnerabilities in Software Systems*, 26 COMPUTS. & SEC. 219 (2007) (developing a model to predict the frequency of vulnerabilities per thousand lines of code); Andy Ozment & Stuart E. Schechter, *Milk or Wine: Does Software Security Improve with Age?*, SEC. '06: 15TH USENIX SEC. SYMP. 93 (2006) (finding that the median vulnerability persists for multiple years before discovery). This means that the brute fact that a vulnerability existed in a software product is rarely, on its own, a clear sign regarding the developer's fault.

125 *See* Tschider, *supra* note 66, at 142–43 (suggesting that reasonable cybersecurity should be based, at least in part, on "dynamic" duties to respond appropriately to threats, and that common law—and not statute—is the appropriate way to articulate such inherently contextual duties).

that carries no meaningful risk of exploitation;[126] and under-inclusive, because regulators and the public should expect that companies protecting highly sensitive data from advanced persistent threats do far more than simply avoid worst practices in securing their software.[127] Instead, an appropriate inquiry would focus centrally on the threat landscape facing an organization and their response to it—the type of inquiry which is inconsistent with an ex ante safe harbor certification scheme.

C. Products Liability

If the purpose of a software liability regime is to prevent hacks, then the appropriate form of liability should involve a flexible negligence evaluation of developers' conduct in the face of potential attackers. This is, in essence, an argument that choices about the scope of a developer's duty should affect the way in which other elements of a tort claim are conceptualized—including the basic choice between negligence and products liability. The same logic can be run in reverse. Adopting products liability as a cause of action ought to influence the scope of a developer's duty.

Product manufacturers generally have a duty to adopt designs that minimize harms even where a product is foreseeably misused.[128] But where products are not merely *mis*used but *maliciously* used, courts hesitate to treat a failure to design against such malicious uses as a defect in the product itself. Courts have rejected attempts to apply products liability to: hollow-point bullets used by a mass shooter to increase the lethality of his shooting spree;[129] fertilizer used by terrorists to manufacture a bomb;[130] Tylenol capsules that were tampered with to poison consumers;[131] or phones that were not designed with controls

---

126 *See* Lipner, *supra* note 60 (critiquing the usefulness of detailed guides like Microsoft's Security Development Lifecycle).

127 *See id.* (pointing out that a security "floor" divorced from risk management frameworks is likely to provide little practical security benefit).

128 *See* RESTATEMENT (THIRD) OF TORTS: PRODS. LIAB. § 2 cmt. p (AM. L. INST. 1998). However, the Restatement notes that product misuse may affect the causation analysis or may lead to a reduction in a plaintiff's damages in accordance with local law, an issue on which it does not attempt to provide guidance. *Id.*

129 McCarthy v. Olin Corp., 119 F.3d 148, 154–55 (2d Cir. 1997) (dismissing design defect claims against ammunition manufacturer because the bullets performed their intended function of causing injury).

130 Port Auth. v. Arcadian Corp., 189 F.3d 305, 313 (3d Cir. 1999) (holding that manufacturers have no duty to design products so as to minimize injuries where third parties have substantially altered the product).

131 Elsroth v. Johnson & Johnson, 700 F. Supp. 151, 162–64 (S.D.N.Y. 1988) ("The notion that manufacturers should nonetheless be forced to write-off the consequences of

to prevent use of certain functions while driving.[132] There are exceptions to the rule that manufacturers are not liable for the tortious conduct of third parties—for instance, when a product design affirmatively encourages such conduct[133]—but these exceptions are typically ad hoc departures from the ordinary assumptions of products liability doctrine.[134] Courts are especially uncomfortable with imposing a duty on manufacturers to guard against third-party malicious use where the conduct is not merely tortious but criminal in nature (as hacking is).[135]

Were products liability to apply to software developers on the terms which it applies to product manufacturers generally, then, it is not entirely clear whether designing against the risk of exploitation by third parties would fall within a typical developer's duty to reasonably design their products. What would much more straightforwardly fall within their duty—at least in the absence of any limitations on recovery of economic loss—is a failure to reasonably design products so as to not *spontaneously* cause harm.[136] So, if a company like CrowdStrike pushes an update to its product that causes IT systems across the world to freeze, resulting in billions of dollars of business interruptions, that seems like a clear violation of its duty to design software that will not cause harm—assuming, at least, that a reasonable development process would have uncovered the defect before the patch was released.[137]

But in the absence of a duty-based limitation on economic loss, this application of products liability to software would make vendors potentially liable for *any* harm caused by a coding error.[138] This, in turn,

---

determined, criminal tampering by third parties as a cost of doing business would be an unprecedented extension of the common law.").

132 Modisette v. Apple Inc., 241 Cal. Rptr. 3d 209, 221–23 (Cal. Ct. App. 2018) (holding, on public policy grounds, that phone manufacturers have no duty to install application "lockout" technology on their products to prevent misuse while driving).

133 *E.g.*, Maynard v. Snapchat, Inc., 870 S.E.2d 739, 743 (Ga. 2022) (allowing a claim that Snapchat's "Speed Filter" feature was defectively designed insofar as it encouraged unsafe driving to proceed).

134 *See, e.g.*, *id.* at 758 (Bethel, J., dissenting) ("[T]his would be the first occasion where Georgia law was understood to impose a duty on manufacturers to account for the criminal conduct of others in the design of their product.").

135 See, for example, the cases in *supra* notes 129–131, each of which involved egregiously criminal third-party conduct. For the primary statute criminalizing hacking, see 18 U.S.C. § 1030.

136 *But cf.* Restatement (Third) of Torts: Prods. Liab. § 21 (Am. L. Inst. 1998) (providing limitations on liability for economic loss from defective products as formalized in the Restatement).

137 *See supra* notes 1–6 and accompanying text. On the application of design defect standards to software, see Scott, *supra* note 56, at 467–68.

138 Due to concerns about foreseeability or feasible alternatives, not every injury would lead to recovery. But no software injury could be *categorically* excluded from the scope of a developer's duty to design against defects. *Cf.* Restatement (Third) of Torts: Prods.

would raise one of the most dreaded specters of tort law: the risk of converting sellers into insurers, without any room left for contract law to cover allocations of risk between contracting parties.[139] And yet only one contemporary proponent of software liability—Jim Dempsey—appears to have even registered this as a potential concern: "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."[140]

Although a welcome acknowledgement of the problem, simply determining a correct standard of care is insufficient to address the concern that manufacturers of software might be subject to excessive liability exposure, for two reasons. First, courts raise the concern about turning duty holders into insurers whenever they feel a need to cabin an expansion of *duty*, not when they feel a need to avoid imposing too high of a standard of care.[141] Second, Dempsey's discussion about liability for "all flaws" occurs in a context that suggests he is anxious to avoid imposing liability for hackers whose exploits were unforeseeable, but that he has not even contemplated the possibility of software liability extending to truly cover all flaws, including security-unrelated bugs.[142] As this Section has argued, however, that is precisely where any intelligible application of products liability to software would begin, treating the issue of liability for exploited vulnerabilities as a residual problem raising difficult doctrinal questions about superseding causes.[143]

LIAB. § 21 cmt. a (AM. L. INST. 1998) (describing the existence of "two [and only two] major constraints" on the kinds of harm which fall within the domain of products liability, both of which involve prohibitions on recovery of economic loss). The point of no-duty rules is, after all, to categorically exclude certain classes of harms from recovery. RESTATEMENT (THIRD) OF TORTS: LIAB. FOR PHYSICAL & EMOTIONAL HARM § 7 cmt. i (AM. L. INST. 2010) (discussing the "categorical" protection of no-duty rules, which are often justified by general considerations such as "the overall social impact of imposing a significant precautionary obligation on a class of actors").

139 Virtually every canonical case cited in this Note alludes to this concern. *See* Kline v. 1500 Mass. Ave. Apartment Corp., 439 F.2d 477, 481 (D.C. Cir. 1970) ("The landlord is no insurer of his tenants' safety, but he certainly is no bystander."); East River S.S. Corp. v. Transamerica Delaval, Inc., 476 U.S. 858, 871–72 (1986) (refusing to extend products liability to cover harms to a product itself because such losses can be insured against by the consumer); Posecai v. Wal-Mart Stores, Inc., 99-1222, p. 5 (La. 11/30/99), 752 So. 2d 762, 766 ("We now join other states in adopting the rule that although business owners are not the insurers of their patrons' safety, they do have a duty to implement reasonable measures to protect their patrons from criminal acts when those acts are foreseeable.").

140 DEMPSEY, *supra* note 52, at 1.

141 *See supra* note 139 and accompanying text.

142 *See* DEMPSEY, *supra* note 52, at 1 (describing an intent "not to impose liability for all flaws but only for some especially consequential ones").

143 For one example of a court carefully analyzing whether a hacker's conduct constitutes a superseding cause, see *In re* Cap. One Consumer Data Sec. Breach Litig., 488 F. Supp. 3d

At this point, the doctrinal dilemma can be summarized. On the one hand, proponents of software liability can acknowledge that a liability regime making vendors liable for all flaws within their software would be too expansive, and can insist on policy grounds that liability should be limited only to harms caused by vulnerabilities (or perhaps only *exploited* vulnerabilities). But any plausible justification for doing so would, presumably, explain why harms caused by *hackers* are more worrisome than other types of harms caused by software.[144] Regardless of the form that such a justification might take, when attention is refocused on the existence of a hacker, it becomes clear that the core issue to be evaluated by a liability regime is a developer's *response* to a perceived threat, which ultimately seems to reduce to a negligence standard. On the other hand, proponents of software liability could reject freestanding negligence inquiries as too plaintiff-friendly or unpredictable, and could instead insist on products liability as the appropriate cause of action for software harm. But in making this move, the question of why liability would not cover software that causes similar harms spontaneously becomes all the more pressing. Applying products liability to achieve the stated purposes of the software liability debate, without allowing recovery in the absence of a hacker, is a bit like making car manufacturers pay pedestrians struck by reckless drivers using their vehicles—but not drivers injured by spontaneous malfunctions.

D. Duty and Distortionary Incentives

I have argued that in addition to creating technical ambiguities, imposing liability for harms caused by vulnerabilities but not other types of bugs is inconsistent with the typical assumptions of tort doctrine. The most cutting objection to this line of argument is to simply deny its relevance. Perhaps common law judges must be attuned to the theoretical fit between the scope of duty and the cause of action chosen to effectuate it, but the pleas of software liability advocates are directed to legislators, and legislators need not be overly concerned about such niceties.

Consider the legislative proposal for software liability offered by the Cyberspace Solarium Commission in its final report to Congress.[145] That proposal is remarkably straightforward: Software vendors have

374, 405–06 (E.D. Va. 2020) (linking the analysis of proximate causation to a foreseeability inquiry that is common within negligence frameworks but would be hard to accommodate under a safe harbor or ex ante standard of care approach).

144 Potential arguments to this effect may involve national security considerations or a heightened concern regarding confidentiality of data. *See supra* note 110; *infra* Section II.D.

145 *See* CSC Proposal, *supra* note 7.

a duty to prevent the introduction of vulnerabilities (and not bugs generally) into their software.[146] This duty is satisfied if and only if a vendor releases a patch within ninety days of learning about a vulnerability.[147] And to prevent crushing liability, recovery is possible for harms to property or physical safety, as well as economic losses greater than $75,000—not economic losses *simpliciter*—but with recovery capped at fifteen percent of a vendor's annual revenue.[148] This type of statutory scheme is rule-like and easily administrable, and the limitations on liability it encodes seem to express a considered judgment about how to balance the deterrent aspects of tort law with the importance of preventing crippling liability.[149] Why should it matter whether these provisions form a unified theoretical whole?

One response is to query whether this proposal's limitations on liability are defensible as a policy matter. Theoretically, arbitrary-seeming limitations on liability are subject to an economic critique: If a commercial seller cannot raise prices enough to cover the costs of negative externalities caused by their goods, then it would seem that their goods cause more harm than benefit, in which case it is odd for the legal system to artificially underwrite their profitability by capping liability. Empirically, many states experimented with damages caps during the tort reform movement, and analyses of the effect of caps on medical malpractice liability have been decidedly ambivalent about whether such caps achieved their stated aims.[150] These responses,

---

146 *See id.* § 2(a) (attaching liability to harms due to "a cyber incident caused or enabled by a covered security vulnerability").

147 *See id.* § 3.

148 *See id.* § 2(c), 4(a). The proposed statutory text suggests that this cap applies to the damages recoverable in a single suit. It is not clear that this does much to protect developers. At the time of the well-known hack of SolarWinds enterprise software in 2020, the company's product was used by 425 of the Fortune 500 (among many other clients). *See* Jason Lemon, *SolarWinds Hides List of Its High-Profile Corporate Clients After Hack*, NEWSWEEK (Dec. 15, 2020), https://www.newsweek.com/solarwinds-hides-list-its-high-profile-corporate-clients-after-hack-1554943 [https://perma.cc/P3BR-7RHP]. If each client could bring a lawsuit, the cap on recoveries as applied only to Fortune 500 clients would amount to 6,375% of SolarWinds's annual revenue. And this is the liability exposure that follows from a *single* vulnerability!

149 The proposal is undercompensatory, both because of the explicit cap on liability and because parties injured within the ninety-day remediation window are barred from recovering, regardless of how negligent the vendor may have been in initially introducing the vulnerability. This suggests that the designers of this proposal sought to make use of the *deterrence* functions of tort law, which they weighed against the importance of fostering innovation, but that they were relatively unconcerned about compensating victims. Such a view is consistent with my claim in this Section that current proposals are motivated by national security interests and are not seriously concerned about consumer welfare.

150 *Compare* Patricia H. Born, J. Bradley Karl & Hugo Montesinos-Yufa, *The Effect of Damage Cap Reforms on Medical Malpractice Insurance Market Conditions During Periods of Crisis*, 86 J. RISK & INS. 1045, 1068 (2019) (finding that damages caps stabilize insurance

however, are open to counter-replies. It is likely the case, for instance, that harms from cyberattacks are both uniquely difficult to anticipate and uniquely likely to create correlated risk exposure.[151] This could justify limiting liability exposure—or even subsidizing developers of insecure software via national cyber insurance schemes and federal incident response assistance.[152]

There is, however, a much larger concern about any regime that artificially restricts liability to vulnerabilities without discussing liability for bugs: Developers may be incentivized to prefer *directly* harming consumers rather than allow third parties to harm them. This is more plausible than it seems at first glance. Recall that poorly-vetted security patches may introduce new errors, including non-security alterations that impair the performance of the system on which they are installed. At least in some cases, such as the Intel patch discussed earlier, developers seem willing—sometimes appropriately, sometimes perhaps not—to sacrifice the performance of their customers' systems in order to patch a vulnerability.[153]

Ideally, a vendor facing such a situation would be permitted to carefully weigh the risk of exploitation against the harms caused by an imperfect patch before deciding whether to rush out a patch. But if a liability regime imposed liability for vulnerabilities but not bugs, the vendor would be irrational to do so. A company approaching the ninety-day patch deadline in the Cyberspace Solarium Commission's proposal would face an enormous incentive to push out an update, even if it only partially mitigated the underlying vulnerability or introduced new bugs of its own, and even if the costs to consumers of pushing a rushed update exceeded the risk of exploitation by a hacker.

This concern suggests that liability that is limited to vulnerabilities could ultimately harm consumer welfare. To date, however, the major

---

markets), *and* David A. Matsa, *Does Malpractice Liability Keep the Doctor Away? Evidence from Tort Reform Damage Caps*, 36 J. Legal Stud. S143, S161 (2007) (finding that damages caps increase physician availability, but only in rural areas), *with* Charles Silver, David A. Hyman & Bernard Black, *Fictions and Facts: Medical Malpractice Litigation, Physician Supply, and Health Care Spending in Texas Before and After H.B.* 4, 51 Tex. Tech L. Rev. 627, 629–30, 663–64 (2019) (finding that damages caps decrease patient safety without increasing the availability of physicians or reducing medical costs), *and* Alberto Galasso & Hong Luo, *Tort Reform and Innovation*, 60 J.L. & Econ. 385, 399 (2017) (finding that the presence of damages caps reduces incentives to produce new patents for medical devices).

151 *See* Josephine Wolff, Cyberinsurance Policy: Rethinking Risk in an Age of Ransomware, Computer Fraud, Data Breaches, and Cyberattacks 4 (2022) (discussing this problem from the insurer's viewpoint).

152 For a persuasive argument along these lines, see *id.* at 188–94 (discussing calls for federal assistance to support cyber insurance through efforts including data collection, reinsurance backstops, and incident response).

153 *See supra* notes 107–08 and accompanying text.

institutional advocates of software liability have been national security entities, not consumer welfare advocates.[154] Vulnerabilities are of interest to the national security community for precisely the reason that differentiates them from other types of bugs: They can be exploited by malicious actors, including foreign rivals. While both bugs and vulnerabilities can cause similar types of harms to computer *users*—data loss, unexpected shutdowns, business interruptions, and so forth—vulnerabilities can also be exploited for espionage,[155] exfiltration of classified information,[156] or even sabotage of critical infrastructure.[157]

Proponents of software liability tend to assume that policies that will promote national security are also good for consumers. This, after all, is how the Biden administration's National Cybersecurity Strategy framed its call for a liability regime: "Markets impose inadequate costs on—and often reward—those entities that introduce vulnerable products or services into our digital ecosystem. . . . Poor software security greatly increases systemic risk across the digital ecosystem and leave [sic] American citizens bearing the ultimate cost."[158]

It is not so clear that national security interests and consumer welfare interests are aligned in the way that this quote suggests. If a major software package is discovered to have a vulnerability, national security may dictate that the developer ought to rush out a patch that cuts off a channel for foreign espionage, even if the patch leaves actual *users* worse off. Inattention to the precise scope of a developer's proposed duty has led proponents of software liability to miss these potential divergences and to advocate for a duty-based limitation that would, in effect, co-opt tort law as a tool of national security. It is easy to think that this is desirable if national security and consumer welfare are in fact in alignment. But they may often not be, especially in the context of decisions regarding patches. To limit developers' liability to harms caused only by vulnerabilities is thus not only technically ambiguous and doctrinally misshapen, it may—at least in some contexts—pose an actual threat to the consumer interests it purports to benefit.

---

154 *See* discussion of proposals from the Cyberspace Solarium Commission and the Office of the National Cyber Director *supra* notes 64–65 and accompanying text.

155 *See* Garrett M. Graff, *China's Hacking Spree Will Have a Decades-Long Fallout*, WIRED (Feb. 11, 2020), https://www.wired.com/story/china-equifax-anthem-marriott-opm-hacks-data [https://perma.cc/K8WU-3C2V].

156 For an engaging story about one of the world's first cyber campaigns of data exfiltration, see CLIFFORD STOLL, THE CUCKOO'S EGG (1989).

157 For a discussion of the ways in which the United States has sought to degrade physical infrastructure abroad to further its geopolitical aims, see KIM ZETTER, COUNTDOWN TO ZERO DAY: STUXNET AND THE LAUNCH OF THE WORLD'S FIRST DIGITAL WEAPON (2014).

158 *See* WHITE HOUSE, *supra* note 65, at 20.

III
Privity as a Limitation on Liability

The previous Part argued that attempts to limit liability facing developers based on whether or not harms are caused by a vulnerability do not withstand scrutiny. This Part offers a more affirmative account. Section III.A suggests that a better means of limiting developers' duties and avoiding the prospect of crushing liability would be to focus on the *parties* to whom the duty is owed. Specifically, I propose that developers should owe a duty to guard against software flaws only to software users to whom they stand in a relationship of near-privity. In Section III.B, I show that such a limitation already has precedent in black-letter tort doctrine in the context of professional malpractice, which partially limits the economic loss rule where certain informational asymmetries exist between professionals and their clients. Thinking of software developers not as product designers but rather as professionals, then, would justify a more defensible limitation on liability that avoids the problems of the vulnerability paradigm. This leads to the thought that software liability should itself be modeled off of malpractice standards. Section III.C concludes by discussing how software malpractice might operate functionally.

A. The Need for Party-Based Limitations on Liability

I begin by noting two important policy desiderata that have garnered broad consensus among proponents of software tort liability. First, in the interests of not unduly undermining innovation, liability should be capped, and companies should not face existential liability exposure as a result of a single oversight.[159] Second, open source developers—those developers who make their work available for free online, where it is often broadly reused by others—should not face liability for flaws in their code, at least where those flaws are the result of negligence and not malice.[160] These are rare points of near-consensus in an otherwise fractious debate.

---

[159] *See* Dempsey, *supra* note 52, at 2 ("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."); CSC Proposal, *supra* note 7, § 4(a) (capping liability at fifteen percent of a developer's annual profits); Herr et al., *supra* note 67 (same).

[160] *See* Hamin et al., *supra* note 67, at 13–15 (surveying 123 articles on software liability and finding that not a single one endorsed liability for open source developers); White House, *supra* note 65, at 21 (calling for a liability regime which would not place responsibility on open source developers). *But cf.* Bryan H. Choi, *Tainted Source Code*, 39 Harv. J.L. & Tech. (forthcoming 2025) (manuscript at 21–24), https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5169060 [https://perma.cc/99Z9-YDS4] (proposing liability for open source developers

The vulnerability paradigm, however, does nothing to advance either desideratum. Although proponents of software liability are anxious to carefully define a standard of care to govern software development,[161] oversights will be inevitable, and a single oversight that results in an exploitable vulnerability could subject a company to bankrupting liability.[162] This is an inherent problem in a context where a flaw that exists in *one* copy of software can affect millions of direct or indirect users, and where risk is extremely correlated: Treating liability as an "on-off" switch based on the nature of the flaw will necessarily mean that economic damages flowing from a legally cognizable breach of duty could well exceed any software company's total value. In addition, vulnerabilities occur in open source software just as well as in corporate software, where they can be just as serious.[163] Making the deterrence of vulnerabilities the linchpin of a software tort regime therefore fails to explain why open source developers should be free from liability.

Perhaps it is worth reframing the problem. Recall that courts are particularly hesitant to make product vendors liable for economic injuries on the theory that the risk of a product's failure which does not endanger anyone's life or property is a risk to be allocated by contract.[164] Indeed, the economic loss rule itself is often thought to apply with heightened importance in the context of products liability, and perhaps to even be derived from that body of law.[165] Perhaps the lack of liability for economic injuries caused by software is not a failure of *tort* law,[166] but rather a failure of *contract* law. As between contracting parties, the economic loss rule does not prohibit vendors from warranting against economic losses, it merely channels recovery

---

but noting that it is unconventional); John Speed Meyers & Paul Gilbert, *Questioning the Conventional Wisdom on Liability and Open Source Software*, Lawfare (Apr. 18, 2024), https://www.lawfaremedia.org/article/questioning-the-conventional-wisdom-on-liability-and-open-source-software [https://perma.cc/4EKN-GN9C] (suggesting that open source developers should bear legal liability, but only where they *intentionally* distribute malicious code, and not where they do so negligently).

161 *See supra* Section I.C.

162 Proposals that incorporate caps on damages would be unlikely to avoid this problem, at least in their current form. *See supra* note 148.

163 *See, e.g.*, discussion of Log4j *supra* note 99.

164 *See, e.g.*, E. River S.S. Corp. v. Transamerica Delaval, Inc., 476 U.S. 858, 870 (1986) (emphasizing this rationale for the economic loss rule).

165 *See, e.g.*, Restatement (Third) of Torts: Liab. for Econ. Harm § 3 cmt. a (Am. L. Inst. 2020) ("The narrower scope of the [economic loss] rule stated here also ties it more clearly to its origins in cases that involve products liability."); *see also* Tiara Condo. Ass'n, Inc., v. Marsh & McLennan Cos., 110 So. 3d 399, 407 (Fla. 2013) (holding that the economic loss rule applies *only* in products liability cases).

166 *Contra* Sharma & Zipursky, *supra* note 45 (suggesting that the lack of liability for software failures is because "the American legal system has lagged decades behind the world of cyberspace").

for economic injuries into the parties' contractual allocation of risk. In the software world, however, vendors not only refuse to warrant the security of their systems, but expressly disclaim any such liability, and courts routinely enforce such disclaimers.[167] And because software licenses are overwhelmingly contracts of adhesion, there is little hope for even well-resourced software users to meaningfully negotiate for greater protection.

Reframing the issue as a failure of contract law rather than a failure of tort law does not mean that the solution cannot take the form of tort reform. But it does suggest that legislatively-created tort duties should supplement contractual relationships. Such a privity-based limitation on tort duties would far better achieve the desiderata of limiting crushing liability and protecting open source development. Privity-based limitations on tort duties are part and parcel of tort law, most commonly in contexts where highly correlated risks make the specter of crushing liability loom large; in such contexts, courts often hold that only parties in privity may sue to recover injuries they suffer.[168] And since open source developers definitionally do not sell their software, they would be straightforwardly immunized by a privity-based limitation on liability. Indeed, although the typical policy justifications for protecting open source developers focus on the economic value of open source software[169] or its speech-like properties,[170] it is arguably more convincing to simply acknowledge that standard tort doctrines would never impose liability for negligently-caused economic losses to absolute strangers.[171] If liability for software injuries were limited to parties in privity, there would be nothing odd about prohibiting lawsuits against open source developers.

Proponents of software liability may object that a privity-based limitation on liability is too generous to developers, since the harmed victims of many software failures may not stand in privity with a

---

167 *See supra* notes 55–56 and accompanying text.

168 *See, e.g.*, Strauss v. Belle Realty Co., 482 N.E.2d 34, 36 (N.Y. 1985) ("[W]hile the absence of privity does not foreclose recognition of a duty, it is still the responsibility of courts, in fixing the orbit of duty, to 'limit the legal consequences of wrongs to a controllable degree' . . . and to protect against crushing exposure to liability . . . ." (quoting Tobin v. Grossman, 24 N.Y.2d 609, 619 (N.Y. 1969))).

169 *See* Manuel Hoffmann, Frank Nagle & Yanou Zhou, *The Value of Open Source Software* 1 (Harv. Bus. Sch., Working Paper No. 24-038) (calculating the economic value of open source software at $8.8 trillion).

170 *See* Bernstein v. United States, 176 F.3d 1132, 1145 (9th Cir. 1999) (holding that export controls limiting the publication of cryptographic algorithms entailed an unconstitutional prior restraint of speech), *withdrawn*, 192 F.3d 1308 (9th Cir. 1999).

171 *See* RESTATEMENT (THIRD) OF TORTS: LIAB. FOR ECON. HARM § 1 (AM. L. INST. 2020) ("An actor has no general duty to avoid the unintentional infliction of economic loss on another.").

developer. This would be true, for instance, when a retail business contracts for software to store its customers' data and the software is ultimately breached by a hacker.[172] This objection has some force, though less than it may seem at first. For one thing, this is a context in which—although the ultimate harm of a data breach is borne by a diffuse set of consumers—legal and compliance costs to the breached company far outweigh the costs to any individual consumer.[173] Thus, absent extensive aggregation of individual claims, permitting only the contracting business to litigate over its injuries does not significantly undermine the deterrent effects of a potential liability regime.

Much more fundamentally, software is a context in which risk factors are extremely correlated: If a bug exists, it exists in every copy of the software, and if a hacker wants to cause harm, they can simultaneously attack every consumer using a vulnerable version of the software. These features make it particularly difficult to insure against cyber-related injuries.[174] If ever there is a context to insist on limitations of duty, it is where risks are highly correlated and companies are unable to purchase insurance against existential liability exposure. But any limitation on duty which is meant to prevent crushing liability will, unfortunately, leave at least some injured parties without a remedy. This is not a flaw in a given proposal, but rather a fundamental feature of tort law. While tort law may be instrumentalized for social aims,[175] it is nonetheless only one legal part of our legal architecture. It comes with an internal logic that makes it difficult to wield as an all-purpose tool to combat all forms of harm.

A privity-based limitation on liability is precisely the type of legal feature which is well-justified as a matter of tort logic in limited contexts, but which has been completely ignored by proponents of software liability. Such a limitation would serve to reduce worries about crushing liability[176] while providing a justification to immunize open

---

172 *See* Sharkey, *supra* note 51, at 362–63 (discussing cases where data breach victims are only tenuously connected through a "web of relationships" with the hacked custodians of their data).

173 *See The Cost of Data Breaches*, Thomson Reuters (Dec. 11, 2024), https://legal.thomsonreuters.com/blog/the-cost-of-data-breaches [https://perma.cc/A3EK-EQK9] (discussing the factors that make up the average $4.88 million cost to a business of a data breach).

174 *See supra* note 152.

175 *See* John C.P. Goldberg, *Twentieth-Century Tort Theory*, 91 Geo. L.J. 513, 524 (2003) (arguing that over the course of the 20th century, tort law was transformed "to achieve collective, not corrective, justice").

176 Note that I do not claim that implementing a privity-based limitation on liability would be *sufficient* to address the concern of crushing liability. A software vendor may provide copies of its software to millions of customers simultaneously, such that following a large-scale cyberattack (or a bad code update, as in the CrowdStrike example), the potential liability exposure could still be bankrupting. Other tort doctrines, such as those

source developers from liability. It would therefore both help to achieve *and* help to justify several desiderata advanced by proponents of the software liability debate. And it would be an entirely natural component of any software liability proposal if we were to regard the current lack of liability for developers not as a failure of tort law, but rather as a failure of contract law to properly allocate the risks of software failures between vendors and consumers. To explain *why* tort liability proposals should seek to correct failures of contract law—rather than creating a freestanding duty to all the world—it is helpful to turn to the domain of malpractice liability, one of the few and notable contexts in which tort law supplements contractual relationships.

B. The Puzzle and Rationale of Malpractice Liability

As discussed above, between contracting parties, the typical justification for the economic loss rule is that it encourages private allocations of risk via contract.[177] And yet, in the case of malpractice liability—which unambiguously attaches to contracts between parties susceptible to a private allocation of risk—tort law nonetheless limits the economic loss rule and permits suits by clients who stand in privity with a professional. Since malpractice represents an alternative context in which tort law curtails the economic loss rule and imposes an independent duty on parties in privity, it is worth questioning whether its reasons for doing so apply in the context of software development.

The Restatement (Third) of Torts offers insight into why this "prominent exception" to the economic loss rule exists.[178] Its core explanation runs as follows:

> The background rule [prohibiting recovery of economic loss as between contracting parties] assumes the parties negotiated on equal footing; it serves to prevent tort obligations from interfering with the allocation of risks the parties made in their contract, and to encourage parties to specify their obligations rather than leaving them for a court to discern later. But those assumptions and policies are weak when a client hires a professional. First, the promises of professionals tend to be limited to careful efforts rather than results . . . . Second, most

relating to proximate causation and foreseeability, may be necessary to still keep liability within tolerable bounds. Legislative schemes might still experiment with statutory damages caps, though this would require greater clarity regarding how such caps would interact with aggregation devices like Rule 23 class actions than have been provided to date. But a privity-based limitation is an important step in the right direction, and it follows from the claim I advance that tort proposals should be focused on imposing tort liability only as a corrective to the failure of contract law to properly allocate risks among parties.

177 *See supra* note 100 and accompanying text.

178 Restatement (Third) of Torts: Liab. for Econ. Harm § 4 cmt. a (Am. L. Inst. 2020).

> clients do not know enough to protect themselves by inspecting the professional's work or by other independent means. Recognizing the tort claim therefore assigns the risk of the professional's negligence where it belongs: with the professional.[179]

The Restatement further observes that disclaimers of malpractice liability in professional contracts are usually suspect due to the "disparity in knowledge between the parties."[180]

The purpose of malpractice liability, at least on this prominent account, is thus to correct for an information asymmetry between providers of professional services and their clients. One party to the negotiation would like to receive an assurance of due care on the part of the professional, but the professional has no way of guaranteeing certain outcomes, and the client has no way of ensuring that the professional's conduct constitutes "due care." To correct for this imbalance in the bargaining situation, the law imposes a tort duty on the professional that is independent of the contract between the parties. This duty requires the professional "to exercise the skill and employ the knowledge normally possessed by members of the profession in similar circumstances" and subjects them to the risk of needing to prove such compliance to a jury.[181]

These rationales apply quite cleanly to the case of software development. It is impossible for developers to assure delivery of "bug-free" or "vulnerability-free" software, so users of software must simply hope that developers will exercise due care in coding and securing their software.[182] And users of software have no meaningful way of assuring the developer's due care because they lack the knowledge necessary to evaluate the robustness of a piece of software, or the judgment calls made in the process of developing it. Even if they *did* have such knowledge, the fact that software is generally delivered without access to source code means that users quite literally cannot evaluate whether the developer's services have resulted in buggy or vulnerable code.

Although some contemporary commentators have argued that software developers should be subject to malpractice-like standards,[183] they have missed two important implications of this argument. First, most scholars have missed that the analogy to malpractice is useful not

---

179 *Id.*

180 *Id.* § 4 cmt. e.

181 *Id.* § 4 cmt. c.

182 *See, e.g.*, Choi, *supra* note 72, at 570–73; Bambauer & Teplinksy, *supra* note 80 ("Simply put, software will never be perfect.").

183 The leading advocate of this approach today is Bryan Choi. *See generally* Choi, *supra* note 72.

only for the purposes of evaluating how a developer's conduct might be evaluated, but also for *justifying* the need to impose a tort duty on developers in order to correct an information asymmetry that cannot be addressed via contract.[184] Second, and as a consequence of this justificatory feature, they have treated malpractice liability as *simply* a means of evaluating a developer's conduct, and have ignored the privity-based limitations that accompany the duty of professionals to their clients.[185] But it is black-letter law that the negligent performance of a professional contract only creates a cause of action for clients and very narrow classes of third parties. The Restatement (Third) of Torts permits a third party to sue a professional for negligence only if (1) the professional had a "pecuniary interest" in the performance of a service, (2) the third party was "one of a limited group of persons for whose benefit the actor performs the service," and (3) the third party suffered economic loss through reliance on the professional's services via a transaction that the professional intended to influence.[186] One could imagine cases where a software developer accepts a contract to develop custom software for a client, fully knowing that a determinate class of third parties will rely on the software.[187] Outside of such narrow circumstances of near-privity, however, professionals owe duties only to their clients. To the extent that analogizing software flaws to professional malpractice helps to justify the need to impose a tort duty on developers, then, it should also justify a near-privity-based restriction on the developer's duty.

C. Implementing a Malpractice Tort for Software Developers

I end by discussing how a malpractice-based justification for software tort liability would affect the liability inquiry when we move

---

184 *But cf*. Mark A. Geistfeld, *Protecting Confidential Information Entrusted to Others in Business Transactions*, 66 DePaul L. Rev. 385, 394–98 (2017) (developing a similar argument in the context of data breach cases).

185 *See* Choi, *supra* note 72, at 587 n.143. Choi raises the contractual nature of software licenses not in the context of a limitation on malpractice liability, but to suggest that the justifications for the economic loss rule are weaker in the case of software. This ignores the fact that most scholars agree that the economic loss rule *does* have an important justification in the context of contracting parties, namely, to avoid displacing the possibility of contractual allocations of risk. *See supra* note 100. Choi's conclusion that "courts have stretched early economic loss cases to avoid adjudicating more difficult questions of reasonable care in software development" therefore seems premature, insofar as it fails to grapple with the substantive rationale for the economic loss rule in the context of contracting parties. Choi, *supra* note 72, at 587.

186 *See* Restatement (Third) of Torts: Liab. for Econ. Harm § 6 (Am. L. Inst. 2020).

187 *E.g.*, Murray v. ILG Techs., LLC, 798 F. App'x 486, 490–93 (11th Cir. 2020) (considering a claim brought by law school graduates against a developer who provided buggy software to a state agency for reporting the results of the Bar Exam).

from the element of duty to the evaluation of a developer's conduct. Consider first, by way of contrast, what a products liability theory of software liability requires a plaintiff to demonstrate. If software were treated like a product, a plaintiff would ordinarily be required to show that there existed a reasonable alternative design for a piece of software which would not have caused their injury, presumably because it would have eliminated a certain bug.[188] A difficulty with implementing this approach in the software context is that bugs (including vulnerabilities), once identified, are often relatively easy to patch; the problem facing a developer is not so much whether to *fix* a bug, but how much energy to invest in *finding* them.[189] For this reason, most current proposals for software liability subtly differ from a straightforward application of products liability: They might only impose liability for vulnerabilities that remain unpatched after a certain date,[190] or they might insist on safe harbors to avoid the otherwise expansive impact of products liability.[191] Even proponents who explicitly bemoan the inadequacy of process-based inquiries and call for the direct application of products liability acknowledge that this would require evaluating potential alternative designs "in light of knowable risks and feasible alternatives."[192] But in the context of software, evaluating knowable risks and feasible alternatives largely *just is* a question of asking how much a developer ought to have invested in testing their code prior to deployment, which is a quintessentially process-based question.

A malpractice theory of liability, by contrast, would explicitly focus the inquiry where it belongs: on the developer's conduct. The leading advocate of this approach today is Bryan Choi, who has argued persistently that safe harbors or regulatory schemes cannot contend with software's enormous complexity, and that only malpractice standards offer the flexibility necessary for evaluating developer

188 *See* RESTATEMENT (THIRD) OF TORTS: PRODS. LIAB. § 2(b) (AM. L. INST. 1998).

189 *See supra* note 124. Rarely does a developer need to carefully weigh whether to use a version of their product that contains a vulnerability or one that does not, because in the vast majority of contexts, eliminating a vulnerability once discovered is near-costless, unlike in cases of physical products alleged to lack expensive safeguards. There are, however, exceptions, such as the case of the Intel chips discussed *supra* notes 107–08 and accompanying text. These cases are arguably those where true products liability would most meaningfully differ from an ordinary negligence evaluation.

190 *See* CSC PROPOSAL, *supra* note 7.

191 *See* DEMPSEY, *supra* note 52, at 15 ("[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 . . . .").

192 Sharma & Zipursky, *supra* note 45 (justifying the original focus on products liability because "products liability typically forces the manufacturer to focus on whether the product's design is safe, not on what steps the manufacturer has or has not taken").

conduct.[193] Rather than asking whether a piece of software was "bug-free," plaintiffs would instead seek to show that a developer cut corners or failed to act as a reasonable developer would, a showing that—except in particularly egregious cases—would ordinarily require expert testimony about professional custom.[194] Many commentators worry that custom-focused inquiries would be overly deferential to an existing landscape of cybersecurity guidelines that have failed to result in secure software.[195] But professional custom creates only a presumption of reasonableness which can, in exceptional circumstances, be overridden by juries.[196] Indeed, proponents of products liability can't dismiss this possibility, since their affirmative case for imposing liability often involves describing conduct that they believe non-expert readers would understand as clearly unreasonable, such as the use of hard-coded default passwords.[197]

Importantly, a malpractice-based approach to software liability actually *would* justify treating vulnerabilities as more serious than other types of bugs in a way that products liability does not. There is a plethora of specialized guidelines, certifications, and textbooks that focus specifically on securing software.[198] Some empirical evidence suggests that developers act quicker and more urgently to address security-related vulnerabilities than other types of bugs.[199] A developer facing a tradeoff between patching a vulnerability and addressing some other type of bug may therefore unreasonably fall below professional

---

193 *See* Choi, *supra* note 45, at 79–86 (discussing how computational complexity makes ex ante standards impossible for software); Choi, *supra* note 72, at 570–73 (same); Bryan H. Choi, *NIST's Software Un-Standards*, 9 Geo. L. Tech. Rev. 65, 68–70 (2025) (critiquing proponents who believe that liability can be pinned to "standards" documents such as those promulgated by NIST, which offer non-binding risk-based frameworks).

194 *See* Restatement (Third) of Torts: Liab. for Econ. Harm § 4 cmt. c (Am. L. Inst. 2020) (discussing the role of expert witnesses).

195 *E.g.*, Sharma & Zipursky, *supra* note 45 ("Where negligence might forgive otherwise preventable harm because a manufacturer technically crossed the t's and dotted the i's on existing cybersecurity frameworks, products liability would demand more.").

196 *See* Choi, *supra* note 72, at 565.

197 *E.g.*, Dempsey, *supra* note 52, at 13 ("[A]s a lawyer and policy geek, I can easily pick out some weaknesses on the [MITRE Common Weakness Enumeration list] whose presence in any product should be legally treated as a design defect . . . ."); Bambauer & Teplinksy, *supra* note 81 ("[I]ncluding a hard-coded account with administrative privileges and a password that cannot be changed is clearly an unacceptable decision."); Bambauer, *supra* note 92, at 172 (opening with a discussion of a hard-coded password as a "self-evidently" insecure practice).

198 For sample frameworks and guidelines, see *supra* note 122. For example certifications, see *CompTIA Certifications*, CompTIA, https://partners.comptia.org/certifications [https://perma.cc/Z6H6-3HVQ].

199 *See* Shahed Zaman, Bram Adams & Ahmed E. Hassan, *Security Versus Performance Bugs: A Case Study on Firefox*, 8 Proc. Working Conf. on Mining Software Repositories 93, 98 (2011).

custom if they failed to focus on the vulnerability first. But this is a presumption, based on generalizations about professional custom which may not be true in some contexts, and reaching it involves a deferential inquiry into the *existing* professional custom among developers rather than a categorical limitation on liability for harms arising from non-security bugs. Thinking of the problem in terms of professional custom ironically does a much better job at reaching the result desired by most proponents of software liability than simply announcing a categorical limitation on liability.

Adapting malpractice standards for software would, however, require some departures from the common law of malpractice. One of the primary barriers to judicial recognition of malpractice claims against software developers has been suspicion of the idea that software development is a "profession," since it does not have formal licensing requirements.[200] Choi has discussed at length whether the requirement of formal licensing is appropriate, how a "profession" might be defined in more functional terms, and why software developers should qualify as one.[201] But these concerns are real ones, as they suggest that software development is too varied to meaningfully determine what professional custom requires.[202] Relatedly, malpractice liability ordinarily attaches to individuals, not corporations. Adapting such an individualistic form of liability in the context of software is admittedly problematic, since most developers work as salaried employees of large corporations—though at least some analysts think that specially-designed business entities (like law firm partnerships) or malpractice insurance could address these concerns.[203]

***

This Part has argued that a malpractice theory of liability offers a better justification for curtailing the economic loss rule in the context of software and that it can be coherently implemented, but that it also

---

200 For an early case reaching this conclusion, see *Hospital Computer Systems, Inc. v. Staten Island Hospital*, 788 F. Supp. 1351 (D.N.J. 1992).

201 *See* Choi, *supra* note 72, at 622–26 (explaining that the centrality of professional judgment to the practice of software development should qualify it as a profession); Bryan H. Choi, *Software Professionals, Malpractice Law, and Codes of Ethics*, 64 COMMC'NS OF THE ACM 22, 24 (2021) (same); *see also* Bryan H. Choi, *AI Malpractice*, 73 DEPAUL L. REV. 301, 307–10 (2024) (discussing this approach in the context of AI development).

202 *See* Scott, *supra* note 56, at 471–75.

203 *See* Choi, *supra* note 72, at 627 (arguing that individual liability for developers would not be abnormal and would not pose insurmountable coordination problems). I am deeply skeptical that this is an adequate response, given the problems of correlated risk exposure and the near-impossibility of insuring even massive corporations. *See* WOLFF, *supra* note 151, at 4. The purpose of this Note, however, is not to argue that malpractice liability maps cleanly onto software development, but more modestly only that it provides a better starting point for analyzing the problem than products liability or strict liability for "worst practices."

carries with it important duty-based limitations. I want to close by reemphasizing how this focus on duty alters the contours of the debate around software liability. By importing a limitation on duty to parties in near-privity, this approach reduces the pressure to identify arbitrary divisions between sources of software harms—like the distinction between bugs and vulnerabilities—that otherwise are deployed in haphazard ways to avoid the prospect of unlimited liability. It avoids the problem of converting vendors into insurers of their products by holding vendors liable only to their immediate purchasers, in marked contrast to products liability, which has long since abandoned a privity requirement.[204] And by limiting the scope of the developer's duty, it reduces pressure to develop a perfect ex ante standard of care, a topic that has consumed the existing debate.[205] Once the developer's duty is made more limited, it becomes possible to permit some greater degree of flexibility in the evaluation of their conduct. Negligence-like inquiries could be permitted. Loose terms like "reasonableness" could be tolerated. And perhaps repeated litigation could make use of the information-forcing functions of tort law by driving into the open information about what the standard of care among developers actually *is*.[206]

Conclusion

The current round of debate over software liability is consumed by a fear of unpredictable liability facing developers. Proponents of liability reform have sought to minimize this concern by pouring their efforts into formalizing a standard of care to which developers should be subject and by limiting liability to harms caused by vulnerabilities. This Note suggests a different approach. Tort reformers would do well to remember that *duty* is the core element of a tort claim which is determined as a matter of law. As long as the scope of developers' duties to prevent economic harm remains unclear, no tort regime can truly offer developers assurance about the extent of their liability exposure.

By contrast, addressing questions about the scope of developers' duty would help give predictability to a proposed regime. Some of these questions are rooted in technical considerations: Should developers

---

204 *See* MacPherson v. Buick Motor Co., 111 N.E. 1050, 1054 (N.Y. 1916) (curtailing the privity rule); Prosser, *supra* note 17, at 1099–1102 (discussing the fall of privity restrictions for products liability suits).

205 *See supra* Section I.C.

206 On the information-forcing functions of tort law, see Elizabeth Chamblee Burch & Alexandra D. Lahav, *Information for the Common Good in Mass Torts*, 70 DePaul L. Rev. 345 (2020).

be liable for bugs as well as vulnerabilities, and if not, how can the distinction be adequately adjudicated by courts? Some of them are policy-based: Should developers be incentivized to rush buggy patches if the result may bolster national security at the risk of losses to consumer welfare? And some of them cut to the core theoretical purposes of software liability: Is tort liability meant to supplement a regime of contract that has failed to adequately police developer conduct, or is it meant to be a freestanding duty to protect strangers?

This Note adopts a particular view of each of these problems. Any regime of software torts should be constrained by the malpractice limitations of near-privity, which can provide a defense against the specter of crushing liability. At the same time, developers should *not* be able to disclaim all harms caused by bugs, and an appropriate liability regime should be designed to avoid incentivizing actions that are harmful from a consumer welfare standpoint. But the core point is more general than this: I claim that regardless of one's particular policy views, a well-designed liability regime must be attentive to the element of duty and to the internal logic of tort law. Rather than simply seeking to wield it instrumentally, reformers who want to improve the status quo should seek to work *with* the grain of doctrine.