NOTE

INSECURE SOFTWARE IS EATING THE WORLD: PROMOTING CYBERSECURITY IN AN AGE OF UBIQUITOUS SOFTWARE-EMBEDDED SYSTEMS

John Daley

CITE AS: 19 STAN. TECH. L. REV. 533 (2016)

ABSTRACT

The proliferation of software-embedded devices through the Internet of Things (IoT) suggests that more features of daily life will be run on software and delivered as online services. Advocates argue that this accelerating shift will generate enormous benefits for consumers, businesses, and governments. But the IoT also presents an acute and growing risk of deploying insecure embedded software into critical systems. This paper argues that the economic incentives for industry are insufficient to deliver an optimal level of cybersecurity for embedded systems and that policy makers must fill the void. Through synthesis of the objectives of regulators and product counsel, a regime for promoting the development of secure embedded systems that preserves "permissionless innovation" while responsibly "protect[ing] public health, welfare and safety" is developed. Drawing on legal and economic analyses of software security and integrating lessons from the open source software movement, the proposed regime combines incentives for public source code disclosure with enhanced ex post liability for insecure software in a limited number of high-risk embedded systems. Autonomous vehicles are analyzed as a test case.

As physical objects increasingly rely on software-embedded systems, regulation must keep pace. Promoting secure software without stifling innovation is critical, particularly given the exponential rate of improvement of networked devices. This paper offers a path to promote the security of embedded systems through a regulatory regime that facilitates vigorous competition between IoT manufacturers, incentivizes secure software development at scale, reduces uncertainty surrounding liability, and fosters public confidence.

TABLE OF CONTENTS

I. INTRODUCTION

Software is eating the world. From entertainment and communications to energy, healthcare and finance, industries are increasingly "being run on software and delivered as online services."[1] The Internet of Things (IoT)—billions of software-embedded devices networked and equipped to send and receive data[2]—promises to accelerate this trend with potentially profound benefits. Proponents of the IoT argue that embedded systems will permit consumer devices to customize experiences and automate tasks and chores; smart business products will increase productivity while reducing waste; governments will deploy sensors to automatically monitor infrastructure and allocate resources more efficiently; healthcare will improve, as "comfortable wearable sensors and computers will enhance every person's awareness of his or her health condition, [and] environment."[3] The bull case for the IoT promises a software-driven utopia where "billions of digital devices . . . will communicate with each other to automate tasks and make life better."[4]

But this sixth computing revolution[5] also promises to compound the acute

1. Marc Andreessen, Editorial, *Why Software Is Eating the World*, WALL ST. J. (Aug. 20, 2011), http://www.wsj.com/articles/SB10001424053111903480904576512250915629460 [https://perma.cc/5KL8-R85H].

2. Glen Martin, *Wearable Intelligence: Establishing Protocols to Socialize Wearable Devices*, O'REILLY RADAR (Apr. 1, 2014), http://radar.oreilly.com/2014/04/wearable-intelligence.html [https://perma.cc/R82X-AKT3].

3. U.S. NAT'L SCI. FOUND., CONVERGING TECHNOLOGIES FOR IMPROVING HUMAN PERFORMANCE: NANOTECHNOLOGY, BIOTECHNOLOGY, INFORMATION TECHNOLOGY AND COGNITIVE SCIENCE 5 (Mihail C. Roco & William Sims Bainbridge eds. 2002), https://www.whitehouse.gov/sites/default/files/microsites/ostp/bioecon-%28%23%20023SUPP%29%20NSF-NBIC.pdf [https://perma.cc/28RA-YTCY].

4. Steve Lohr, *A Messenger for the Internet of Things*, N.Y. TIMES BITS (Apr. 25, 2013, 12:15 AM ET), http://bits.blogs.nytimes.com/2013/04/25/a-messenger-for-the-internet-of-things [https://perma.cc/5HGE-73PS].

5. The first five being, respectively: mainframes, minicomputers, PCs, the internet, and mobile.

crisis of cybersecurity. As software security expert Bruce Schneier argues, the pervasive vulnerability of embedded systems today is structurally similar to the security crisis of PCs in the mid-1990s—only much worse.[6] At that time software developers tended to keep vulnerabilities secret and delay security releases; consumers, on the other hand, tended to ignore patches when they were released. Today, disclosure requirements and automatic security updates have improved the situation but, unfortunately, the security of embedded systems considerably lags behind that of traditional software.[7] Three factors explain this divergence and highlight why the security implications of the IoT are so profound. First, unlike PCs in the mid-1990s, each of the billions of embedded systems in the IoT is connected to the Internet, creating many new sources of vulnerability. Second, the economic and legal structure of the software development industry leaves no single entity with strong enough incentives to secure software before it is shipped. The fragmentation of software development caused by the upsurge of companies embedding software into their products will compound this structural failure. Third, and perhaps most troubling, if software vulnerabilities in IoT devices do arise they are often impossible to patch because the full source code is not available.

In other words, the IoT will inundate our homes,[8] vehicles,[9] infrastructure,[10] and bodies[11] with insecure embedded systems that are hard to maintain and potentially unpatchable. To consider one disturbing example, *Ars Technica* recently reported on Shodan, a search engine for the IoT, that lets users easily browse vulnerable webcams. The feeds include "images of marijuana plantations, back rooms of banks, children, kitchens, living rooms, garages, front gardens . . .

6. Bruce Schneier, *The Internet of Things Is Wildly Insecure–and Often Unpatchable*, WIRED (June 1, 2014, 6:30 AM ET), http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem [https://perma.cc/75B4-VGFQ].

7. George V. Hulme, *Embedded System Security Much More Dangerous, Costly than Traditional Software Vulnerabilities*, CSO (Apr. 16, 2012, 8:00 AM PT), http://www.csoonline.com/article/2131478/critical-infrastructure/embedded-system-security-much-more-dangerous—costly-than-traditional-softwa.html [https://perma.cc/J7WD-H5ZN].

8. Dean Takahashi, *Hello, Dave. I Control Your Thermostat. Google's Nest Gets Hacked*, VENTURE BEAT (Aug. 10, 2014, 8:00 AM), http://venturebeat.com/2014/08/10/hello-dave-i-control-your-thermostat-googles-nest-gets-hacked [https://perma.cc/X85T-55UT].

9. Terrell McSweeny, *Hackers Make Cars Safer. Don't Ban Them from Tinkering*, WIRED (Oct. 21, 2015, 6:50 AM ET), http://www.wired.com/2015/10/terrell-mcsweeny-white-hat-car-hacking-makes-cars-safer [https://perma.cc/KBK9-C5HC].

10. Dan Lohrmann, *Hacking Critical Infrastructure Is Accelerating and More Destructive*, GOV'T TECH. (Apr. 11, 2015), http://www.govtech.com/blogs/lohrmann-on-cybersecurity/Hacking-Critical-Infrastructure-is-Accelerating-and-More-Destructive.html [https://perma.cc/JFR3-R43V].

11. Sophie Curtis, *Wearable Tech: How Hackers Could Turn Your Most Private Data Against You*, THE TELEGRAPH (June 25, 2014, 2:11 PM BST), http://www.telegraph.co.uk/technology/internet-security/10925223/Wearable-tech-how-hackers-could-turn-your-most-private-data-against-you.html [https://perma.cc/CE3C-PSWW].

colleges and schools, laboratories, and cash register cameras."[12] Most alarmingly, several feeds included sleeping children. But, despite these risks, neither software vendors, facing economic pressure to ship fast and fix bugs later, nor end users, notoriously unwilling to patch frequently or accept upgrades, have done much in response.

As a result, if society hopes to unlock the moonshot potential of the IoT without opening a Pandora's Box of security disasters, policy makers must fill the void. Through synthesis of the objectives of regulators and product counsel, this Note sketches out a regime for promoting the development of secure embedded systems that attempts to preserve space for "permissionless innovation"[13] while responsibly "protect[ing] public health, welfare and safety."[14] Drawing on legal and economic analyses of software security and integrating lessons from the open source software movement, the proposed regime combines incentives for public source code disclosure with enhanced ex post liability for insecure software in a limited number of high-risk embedded systems. The scope of this new liability regime is debatable but lies beyond the scope of this paper. Autonomous vehicles are analyzed as a test case but other specific examples of embedded systems that ought to be covered are left for subsequent discussions.

The argument below proceeds in four parts. Part I analyzes the economic and legal structure of software development and identifies why market forces have failed to discipline developers into enhancing security. Part II draws on parallels with software development for election machines to make the case for public source code disclosure as the best method for developing secure embedded systems. Part III integrates the arguments of parts I and II and sets out the dual-prong proposal that forms the core of this paper and deals with several key anticipated criticisms of the regime. Part IV applies that regime to the test case of autonomous vehicles.

II. LEGAL AND ECONOMIC ANALYSIS OF SOFTWARE SECURITY

Software as an information good does not fit neatly into liability regimes designed for material goods in the physical world. The incongruities between software and hardware have prompted vigorous debate regarding the degree to which software developers should be liable for security vulnerabilities in their code. Some strident critics of the industry have argued that the current paradigm is exceedingly generous to developers and has permitted the software industry to "blame cybercrime, computer intrusions, and viruses on the expertise and sophistication of third party criminals and on careless users who fail to implement

12. J.M. Porup, *"Internet of Things" Security Is Hilariously Broken and Getting Worse*, ARS TECHNICA (Jan. 23, 2016, 7:30 AM PST), http://arstechnica.com/security/2016/01/how-to-search-the-internet-of-things-for-photos-of-sleeping-babies/ [https://perma.cc/QS26-HFBD].

13. *See* ADAM THIERER, PERMISSIONLESS INNOVATION: THE CONTINUING CASE FOR COMPREHENSIVE TECHNOLOGICAL FREEDOM (2014).

14. Exec. Order No. 13,563 § 1(a), 3 C.F.R. 215-16 (2012), *reprinted in* 5 U.S.C. § 601 app. at 101-02 (2012).

adequate security, rather than acknowledging the obvious risks created by their own lack of adequate testing and flawed software design."[15] Even amongst those who advocate for software developer liability, however, no consensus exists on whether the right legal approach lies in contract, product liability, strict liability, no-fault liability, or negligence.[16] Intellectual property scholar Pamela Samuelson, for example, has argued that strict liability ought to attach to, among other things, embedded systems.[17] While strict liability has the benefit of being unambiguous, it would stifle software innovation and potentially erode incentives for other actors, like end users, to employ good security practices.

The software industry, supported by some lawyers and economists, has, not surprisingly, rejected arguments for developer liability. Legally, scholars have analogized software to books, whose value lies in the information content rather than in the physical embodiment, and suggested that a much less stringent liability rule, such as negligence, should apply. Economically-oriented critics have suggested market forces provide sufficient discipline and incentive for firms to secure their software, negating the need for regulation. Rational developers, in this analysis, are incentivized to invest in software security when the cost of that investment is less than the resulting damages, adjusted by the probability of a security event, plus the associated administrative costs.[18]

The conflicting positions have led to impasse and the multi-hundred-billion dollar software industry has matured in a "legislative void" in which software developers and vendors generally bear no liability for their products, including security vulnerabilities, and there is virtually no quality assurance.[19] Scrutiny of the economic argument advanced by those who reject software liability, however, suggests that market forces may not discipline private developers into enhancing their software security to societally optimal levels. The current regime of caveat emptor in which software consumers bear the risk of security vulnerabilities entirely is unsuited to a market where information asymmetry is high; in most cases the source code of embedded systems will be inaccessible to consumers, but even if the code is available, the vast majority of consumers lack the expertise to effectively evaluate security features. Consumers therefore lack the ability to effectively compare security across competitors, creating little incentive for

15. Michael L. Rustad & Thomas H. Koenig, *The Tort of Negligent Enablement of Cybercrime*, 20 BERKELEY TECH. L.J. 1553, 1559 (2005).

16. *See* Jane Chong, *We Need Strict Laws if We Want More Secure Software*, NEW REPUBLIC (Oct. 30, 2013), https://newrepublic.com/article/115402/sad-state-software-liability-law-bad-code-part-4 [https://perma.cc/6PLH-PERA].

17. Pamela Samuelson, *Liability for Defective Electronic Information,* COMM. OF THE ACM, Jan. 1993, at 21. *See also*, Kevin R. Pinkney, *Putting Blame Where Blame Is Due: Software Manufacturer and Customer Liability for Security-Related Software Failure*, 13 ALB. L.J. SCI. & TECH. 43, 62-82 (2002).

18. For a more in depth treatment of the arguments for and against each of these proposals, as well as numerous others, see Robert W. Hahn & Anne Layne-Farrar, *The Law and Economics of Software Security*, 30 HARV. J.L. & PUB. POL'Y 283, 337 (2005).

19. I.Lan Systems, Inc. v. Netscout Serv. Level Corp., 183 F. Supp. 2d 328, 334 (D. Mass. 2002).

developers to invest. This market failure is compounded by the fact that, because developers are insulated from liability, the costs of insecurity are externalized. Taken together, these insights suggest rational developers are likely to significantly underinvest in software security.[20]

Recognizing these market failures, some economists have analogized the software market, at least in terms of security, to the used car market and proposed "lemon laws" through which developers would be liable for vulnerable software. This analogy is inapt, however, because unlike used cars each copy of software is an exact replica of the master version. Therefore any flaws giving rise to liability would affect *every copy* of the software and generate crushing liability very quickly. A regulatory regime committed to preserving the dynamism of software innovation cannot accept such a tradeoff.

Logically, there are two approaches to improving security given these economic realities: incentivizing or mandating increased ex ante investment in security by developers, or increasing ex post punishment through a liability regime to compensate victims and provide discipline through deterrence.[21] Adopting strict liability for security vulnerabilities or crafting a new tort of negligent enablement of cybercrime are examples of possible ex post punishment reforms. Conversely, a system of government-mandated security performance standards, analogous to emission standards applied to the auto industry, could offer ex ante investment discipline. But the imperative to protect permissionless innovation within a regulatory regime suggests that government should not be in the business of dictating the pace and arc of software development.

Thus, the best approach is to structure incentives, rather than impose mandates, that make secure software development economically and legally attractive. The next section explains why incentivizing public source code disclosure is the correct tool for striking that balance.

20. I hope to develop this insight into a more formal model in a subsequent paper but we may briefly consider a stylized cost-benefit model of a firm's decision to invest in more secure software. Without the investment the firm incurs a loss, L, with some probability, p, and some cost of responding to that loss administratively, C, with the same probability, p. Expected losses are therefore $pL + pC$. If the firm can invest in some security-enhancing system or procedure, X, that reduces p to some lower probability, q, then costs are $X + qL + qC$. My basic contention in this Note is that the government can create a system in which the cost of X is low, such that $pL + pC > X + qL + qC$. A more sophisticated position would advance the argument that this differential is significantly greater if we include the external costs borne by customers, Z, on both sides, rather than viewing the firm in isolation: $pL + pC + pZ >>> X + qL + qC + qZ$. *See* Hahn & Layne-Farrar, *supra* note 18 at 300-02.

21. Note that an alternative system could include ex post punishment of consumers for improper use or failure to take appropriate precautions. Note also that the existing regime of fines and criminal sanctions for malicious hackers is not considered here.

III. WHY PUBLIC SOURCE CODE DISCLOSURE PROVIDES THE MOST RELIABLE PATH TO SECURE EMBEDDED SYSTEMS

Software development is hard. Secure software development is *very* hard. Security vulnerabilities are often subtle and difficult to detect even for experts. The open source movement has been successful in part through its ability to overcome the intellectual limits of any single developer or team of developers by leveraging contributions from hundreds or thousands of individuals. As the saying goes, "many eyes make all bugs shallow." The success of the open source model is reflected in the fact that some of the most popular development tools, such as Hadoop and Kafka, as well as exciting, sustainable businesses, such as Cloudera and MangoDB, have emerged from open source projects.[22]

David Wagner, a computer scientist at University of California, Berkeley, extracted key lessons from the open source experience when looking for ways to secure election software.[23] His answer—mandatory source code disclosure—is a significant intellectual contribution that provides a framework for constructing better incentives for the development of secure embedded systems. The societal importance of election software is obvious, but the arguments lend themselves to embedded systems that are taking on increasingly important functions.

As a preliminary matter it is important to define several terms that may be unfamiliar to non-technical readers. First, source code is the human-readable representation of the set of instructions that control the operations of a computer. Currently, in the vast majority of cases, this source code is written by commercial software vendors with some constitutive code licensed from other developers (commonly referred to as COTS: commercial off the shelf software).[24] Second, software security vulnerabilities come in two flavors: implementation (bugs) and design (flaws).[25] Both flavors create insecurity, but flaws, because they constitute structural vulnerabilities, tend to be more difficult to fix.

Currently, a variety of federal and state standards require that election software vendors disclose their source code to a testing laboratory selected by the vendor to verify compliance with promulgated standards. One could imagine a similar system applied to embedded systems. However, while analysis by third party security auditors would be an improvement over our existing system, in

22. The sustainability of these businesses has also been facilitated by a shift from the General Public License (GPL), which does not permit charging for open source software, to the Apache Software License, which lets the vendor decide whether to share code changes with the community or keep them as proprietary. Under this hybrid model, open source companies can make clients pay for product extensions (as well as services and support).

23. *Machines and Software: Hearing Before the Elections Subcomm. of the Comm. on H. Admin.*, 110th Cong. (2007) (written testimony of David Wagner, Comp. Sci. Div., U.C. Berkeley), http://www.cs.berkeley.edu/~daw/papers/testimony-house07.pdf [https://perma.cc/L3AK-BTZ6].

24. The vendor may or may not have access to the source code of COTS.

25. Paco Hope, *Bugs Versus Flaws: Know What You're up Against*, BUS. COMPUTING WORLD (May 29, 2013), http://www.businesscomputingworld.co.uk/bugs-versus-flaws-know-what-youre-up-against [https://perma.cc/K53E-4SD5].

which the vast majority of embedded system software remains proprietary without any external review, it is not sufficient. Practically, there is an inherent conflict of interest when the technical auditor is chosen and paid by the software vendor; the drastic failure of rating agencies prior to the financial crisis provides ample evidence that this is not a model to emulate. Moreover, even assuming the technical competence of the security auditors (a non-trivial assumption) any analysis is logically limited to detecting the presence of certain flaws, not their absence, so limiting review to a small number of auditors is less than ideal given the subtleties of security vulnerabilities. Finally, it is difficult to ensure that testing reflects how the software will be used in practice, limiting the strength of the conclusions auditors are able to draw.

Public source code disclosure is a superior option not because it eliminates the logical and practical difficulties faced by auditors, but because it creates better incentives for developers, entices many more reviewers to analyze the code, and lacks the financial conflict inherent to contract security auditing. At an individual level source code disclosure provides a powerful source of discipline by exposing the work of developers to scrutiny. Developers, like most professionals, tend to value the esteem of their peers. Currently, because source code remains opaque to outsiders, security vulnerabilities can persist without lowering the status of the developers responsible for source code maintenance. A public disclosure regime would create powerful social pressure to ensure that, at a minimum, code is free of well-known vulnerabilities. This by itself would be a significant improvement in security for embedded systems considering that 10% of known security vulnerabilities account for 90% of cybersecurity breaches according to Qualys, a cybersecurity firm.[26] If peer scrutiny is sufficient to ensure *only* that developers review code for known bugs (such as through a service like BlackDuck) before disclosure, the system would be worth adopting. But, as seems likely, if it also generates increased social status for secure development, it could increase accountability more broadly and lead to much more secure embedded systems.

Design flaws are more difficult to detect as they tend to be more subtle and require a holistic understanding of the software. Yet public disclosure ameliorates this risk as well by permitting technical experts, scholars, and hobbyists to probe and analyze the code. This is the precise benefit that has fueled the open source movement: distributed intellectual resources collaborating effectively. Ongoing review by thousands of interested individuals, many of whom will possess technical skills superior to employees of security auditing firms, is far more likely to detect security flaws that could compromise embedded systems.

Finally, and perhaps most importantly, public disclosure would bolster public confidence in embedded system technology. There is an intense and growing debate around the role of technology and software in our lives. Opaque, proprietary code embedded in critically important devices, such as automobiles, allows critics to conflate obscurity with incompetence, deviousness or malicious

26. Qualys, The Laws of Vulnerabilities: Six Axioms for Understanding Risk (2006), https://www.qualys.com/docs/Laws-Report.pdf [https://perma.cc/XQN6-55CF].

intent. Secrecy also robs advocates of the information they need to cultivate public confidence; regardless of intent, secrecy invites questions about whether the developer hopes to cover up problems. Transparency is needed to allow critics and advocates to be able to point to code and identify why it does or does not meet appropriate standards.

The danger that public mistrust of embedded systems poses should not be underestimated. It may be psychologically irrational, but people are more comfortable with the devil they know (observable human error) than that which they do not (obscure software vulnerabilities), even if the latter is statistically much rarer. In the case of voting, the process of manual counting observed by scrutineers is inefficient, but it is integral to public trust in the electoral system. If, like this author, you believe in the potential of technologies like the IoT to improve our lives, avoiding a public backlash such as that which has stifled deployment of nuclear energy technology for the past three decades is critical. Disclosure of critical software source code would help build public confidence by allowing oversight of its deployment and fostering a spirit of empowerment with regard to how those systems influence our daily lives.

IV. A New Liability Regime for Software Embedded Systems

A successful liability regime for embedded systems must include at least three features. First, it must combine ex ante incentives to invest in security with ex post liability that, while sufficient to discipline developers, does not stifle innovation. Second, software developers should be incentivized to publicly disclose source code because this offers the most reliable method to secure development currently available. Third, the regime must cultivate and inspire public confidence in embedded systems in order to avoid the fate of other moonshot technologies whose deployment has been deterred by negative public sentiment.

A dual disclosure-safe harbor regime coupling expanded tort liability for software developers with an opt-in public disclosure safe harbor offers a practical means to achieve these ends. Mechanically, because tort law is state law, such a regime would have to be created by a state legislature willing to draft and pass a law expanding products liability to a subset of embedded systems with critical security implications. California seems like the obvious choice because of its large market that would incentivize companies to comply in order to sell into the state and its status as the domicile of the world's largest technology companies. Courts will still face the challenge of applying tort law to software, but the costs of software insecurity will be more appropriately balanced from a policy perspective.

Simultaneously, a safe harbor would be extended to any developers who voluntarily disclosed the source code of their embedded systems. The disclosure mechanism could take on multiple forms, depending on the breadth desired. The logical extreme implies public source code disclosure but, anticipating criticisms that will be addressed more fully below, several intermediate steps of varying efficacy are also available. For example, disclosure could be limited to a smaller group, such as: self-regulating industry bodies, independent technical experts (a group of computer science scholars, for instance), or a government regulatory agency. None of these provide all of the benefits of public disclosure but each would improve on the current system. Self-regulating industry bodies, coupled with some degree of government oversight, could impose software review best practices and coordinate more closely on disseminating information about known security bugs. Disclosure to truly independent technical experts, rather than third party security auditors paid by software developers, would provide an additional source of skilled technical review. Finally, disclosure to regulatory bodies who, if needed, could commission technical experts would provide limited benefits. The disclosure-safe harbor regime could also be phased in over time with some of these intermediate proposals preceding full public disclosure.

Critics will contend that any liability borne by software vendors will extinguish the vibrant startup ecosystem but the limited scope of the regime means that the majority of software developers will not be affected. Nonetheless, there are at least three lines of criticism deserving closer consideration, including: erosion of intellectual property rights, giving aid to attackers, and transition risks. While each concern has merit, none is fatal to the proposed disclosure-safe harbor regime.

Intellectual property is central to the software industry and disclosure of source code will, admittedly, affect the intellectual property rights of embedded system developers. For one, disclosure will remove any trade secret protection. However, this firm-level loss is justified by the important public policy goals of enhancing embedded system security, particularly when we consider that copyright and patent protection, respectively, will continue to apply. In fact, copyright protection could be *strengthened* under this proposal because firms opting-in to the system would not be able to hide unauthorized use of copyright-protected code. There may be increased litigation as firms more transparently attempt to implement the "idea" behind code to avoid copyright restrictions but this would not be a structural change from the current situation. Patent protection is also still available, although the Supreme Court has limited its scope significantly in recent years.[27] Therefore, while this regime certainly entails change to the protection of embedded systems, it would not be detrimental to the intellectual property rights of developers.

A second line of attack suggests that the benefits of public source code disclosure might be offset by the risk of facilitating attacks in the short run before vulnerabilities can be recognized and fixed. This is a valid concern that computer security experts continue to struggle with. However, our view must remain on providing structural incentives for more secure software design and, in the long run, obscuring code will only slow, not stop, persistent hackers because systems that rely on secrecy for security are inherently fragile. So, to address the transition period risk, the disclosure system could be structured so that code must be made

27. Alice Corp. Pty. Ltd. v. CLS Bank Int'l, 134 S. Ct. 2347, 2360 (2014) (holding the abstract idea of an intermediated settlement patent-ineligible).

public for a certain period of time before being deployed to end-users. This would allow bugs and flaws to be identified before hackers had the chance to attack functioning systems. Risks remain, such as the possibility that hackers identify vulnerabilities in functioning systems before altruistic reviewers or that vulnerabilities identified cannot be practically fixed. Yet, in reality, hackers are already attacking these systems and enjoying considerable success. While there may be cases in which the disclosure regime facilitates malfeasance, the history and success of the open source software movement suggests that, in the vast majority of cases, security will be enhanced through the collaboration of a broad community of developers interested in promoting security.

Economic and competitive risks constitute a third set of potential issues. Firms may delay opting into the safe harbor, bearing some risk of liability, in order to gain strategic insights when their competitors disclose their code. The competitive implications of this game-theory type problem could cut in multiple directions. It is possible that firms could adopt a "free-rider" mentality by under-investing in their own development and simply using the disclosed software of competitors to build competing products. Antitrust problems may also arise if competitive dynamics emerge making it advantageous for firms to implicitly or explicitly coordinate. However, while these are difficult problems, they are difficult in a practical sense. If the normative argument for a disclosure-safe harbor regime is persuasive, then the economic and competition risks are manageable through the same legal and regulatory tools through which competition in other industries is currently managed.

One final wrinkle arises from the fact that software vendors opting into the system may not have the permission or ability to disclose the source code of third party COTS software. It is unclear how big a problem this would be if the regime was phased in over time, but it would likely create some practical difficulties during the transition period. One option would be to exempt third-party COTS from the disclosure system. This undermines many of the benefits of the regime but is perhaps an acceptable kludge, particularly if the exemption was time-limited.

Having dealt with some of the expected criticisms of a disclosure-safe harbor regime for embedded systems, it is time to test the proposal by applying it to one of the most exciting moonshot technologies currently in development: autonomous vehicles.

V. AUTONOMOUS VEHICLES AS A TEST CASE

Autonomous vehicles have the potential to change how we travel, where we live, and whether we purchase cars—all while preventing millions of traffic fatalities per year. And they are coming fast. Google's autonomous cars have already traveled several million miles on California roads. Nevada has been experimenting with autonomous freight trucks in a variety of settings. Every major car company is investing heavily in autonomous technology and transportation network companies like Uber and Lyft are speaking openly about plans to roll out fleets of autonomous on-demand vehicles for the $40B market by the mid-2020s.[28]

Autonomous vehicles also present a fascinating study in legal and regulatory responses to new technology because unlike pure software, they will constitute "products" under products liability law and therefore potentially expose developers to legal liability. Further, in the event of an accident, liability could theoretically attach to any one of multiple parties, including: the occupants, the original equipment manufacturer (OEM), the supplier or the software provider—and that is only when the vehicle is operating as intended. Hacks and faulty software could alter the calculus of where liability ought to fall.[29] In response, the RAND Corporation, a think tank, has advocated the adoption of a national no-fault insurance regime, modeled on the existing law in twelve states, that would allow crash victims to seek damages from their own auto insurers rather than from the other driver; the benefits of uniformity, they argue, would prevent insurance uncertainty from slowing the adoption of autonomous vehicle technology.[30]

Others, such as California, have suggested banning autonomous cars from being fully autonomous by requiring that an on-board human be able to take control at any time.[31] Still others have adopted the opposite approach, instructing regulators to promulgate rules promoting autonomous vehicle rollout.[32]

Yet, despite the many potential benefits, the security risks of autonomous vehicles are astounding. The prospect of hackers taking control of an IoT-connected webcam is frightening but the possibility of malicious actors commandeering a car traveling sixty miles per hour down a freeway—or worse yet a fleet of networked vehicles—is truly terrifying. Unfortunately, these risks are real. Market pressure is forcing car manufacturers to develop embedded systems for their vehicles but the intense competitive pressure creates dangerous scope for under-investment in security. The recent Volkswagen emissions scandal provides an instructive guide to how future security flaws might arise and perpetuate

28. Richard Waters, et al., *Carmakers: A New Direction of Travel*, FIN. TIMES (Jan. 8, 2016), http://www.ft.com/cms/s/0/fcb9e8cc-b5f1-11e5-8358-9a82b43f6b2f.html#axzz3wh5tMvWp [https://perma.cc/B7KB-C6MU].

29. Edward Taylor, *Autonomous Cars Shift Insurance Liability Toward Manufacturers*, REUTERS (Sep. 8, 2015, 5:53 PM EDT), http://www.reuters.com/article/us-autos-autonomous-regulations-idUSKCN0R82OQ20150908#0eFpuSoYyRAr97S4.97 [https://perma.cc/HE68-YRJF].

30. JAMES M. ANDERSON, ET AL., RAND CORP., AUTONOMOUS VEHICLE TECHNOLOGY: A GUIDE FOR POLICYMAKERS (2016), http://www.rand.org/pubs/research_reports/RR443-2.html [https://perma.cc/YQ3M-369Z].

31. Jordan Golson, *California Wants to Keep Autonomous Cars from Being Autonomous*, THE VERGE (Dec. 16, 2015, 6:30 PM EDT), http://www.theverge.com/2015/12/16/10325672/california-dmv-regulations-autonomous-car [https://perma.cc/B99C-BGUF].

32. Gabriel Weiner and Bryant Walker Smith, *Automated Driving: Legislative and Regulatory Action*, THE CENTER FOR INTERNET AND SOCIETY (Mar. 19, 2016, 11:34 PM PT), cyberlaw.stanford.edu/wiki/index.php/Automated_Driving:Legislative_and_Regulatory_Action [https://perma.cc/C2ZK-TRC6].

within autonomous vehicle manufacturers. Moreover, a high profile security incident could turn public opinion broadly against autonomous vehicles and delay widespread adoption for years or decades. This is disastrous from a technological and human standpoint as it would postpone the introduction of technology likely to save millions of lives and improve transportation efficiency.

If you believe, as I do, that autonomous vehicles are a radical safety and productivity breakthrough then the need for adopting a liability system that will accelerate, rather than impede, progress is paramount. So, let us consider how a disclosure-safe harbor regime would work in this case. If California, for instance, were to introduce the regime then any firm operating autonomous vehicles within the state would be on notice that, after the effective date, they could face liability under existing products liability law for their vehicles. However, if they chose to opt-in to the disclosure regime they would benefit from the safe harbor, insulating them from liability.

As a result, vehicle occupants, as they do currently, would be required to carry insurance for personal vehicles (collision and/or liability) and commercial carriers would still be responsible for insuring their vehicles appropriately. The critical point is simply that liability would not attach to the software operating the vehicle. If there was a security incident leading to a vehicle accident it would fall to the insurance of the vehicle owner and/or operator to provide compensation, with rates presumably reflecting the risk of those incidents.[33] On the contrary, if a firm chose *not* to opt in to the safe harbor and there was a security incident leading to damage, injury or death, the software firm could face liability. The incentives for the software developers are clear and, importantly, the difficult transition risks discussed above would not apply if the mandatory disclosure regime came into effect well before autonomous vehicles were deployed to consumers.

Furthermore, from the state's perspective, providing clarity to autonomous vehicle firms regarding liability could attract new market entrants and promote local experimentation and development of a dynamic innovation ecosystem. Framing this type of regulatory arbitrage as a local economic stimulus could entice state legislatures to support reform.

VI. CONCLUSION

As physical objects increasingly rely on software-embedded systems, regulation must keep pace. Promoting secure software without stifling innovation is critical, particularly given the exponential rate of improvement of networked devices. This Note offers a path to promote the security of embedded systems through a regulatory regime that facilitates vigorous competition between IoT manufacturers, incentivizes secure software development at scale, reduces

33. Empirical work comparing the economic gains realized from reducing cybersecurity incidents against the losses (if any) due to mandatory source code disclosure offers a fruitful avenue for extending this argument.

uncertainty surrounding liability, and fosters public confidence. The disclosure-safe harbor regime envisioned strikes a critical balance between software innovation and security that is currently lacking. In this sense, the proposal can be viewed as an extension of efforts to require disclosure of security breaches, such as California's influential law passed in 2002.[34] These laws are based at least in part on the theory that ex post disclosure of security breaches provides a source of ex ante discipline on developers and businesses sensitive to reputational harm. While perhaps not as effective as proponents had hoped, the moderate success of this initiative should ameliorate concerns that a new liability regime will stifle the robust software development ecosystem in California.

While software is a powerful accelerant of social and economic progress, "permissionless innovation" cannot be a synonym for unaccountability. Regulation, including potential liability, must be proportional to risk, and the risk of security vulnerabilities in widely deployed software-embedded systems is significant. The benefits of introducing ex post liability to a modest subset of sensitive embedded systems, when combined with a safe harbor that preserves space for permissionless innovation, significantly outweigh the costs.

34. California Security Breach Information Act, S.B. 1386, 2002 Cal. Stat. 915.