Strategically Holding Back Bugs and Patches

Marie Vasek and Ryan Castellucci

University College London, London, UK
m.vasek@ucl.ac.uk, pubs@ryanc.org

Abstract. Most academic work on patching behaviour is under the assumption that vendors must patch every bug. We outline scenarios where it is strategic to not report a bug or not patch a known bug in a system. We give examples where others have deployed these strategies and discuss the varied trade-offs in the ecosystem. Bucking the common wisdom, we offer practical considerations to otherwise taboo behaviour.

1 Introduction

Most existing research considering patching software systems thinks about when or how to patch bugs. Governments even have started to require the patching of severe bugs within a set time table [25]. But, it is not always strategic to deploy a patch to your system. The UK's Cybersecurity Center (NCSC) strongly encourages patching, though notes that sometimes it is too hard and they pragmatically lay out mitigation strategies other than patching[1] [32].

Similar to these tradeoffs, it is not always strategic for defenders to declare the bugs that they found to the appropriate vendors or write patches for your software library or codebase. Reporting bugs is hard, and often fruitless. Writing patches for bugs can be costly, particularly for bugs that are not likely to be exploited immediately or ever.

Our work considers two components. First we consider why some users do not report bugs to vendors. Next we consider why vendors do not patch bugs which are reported. While not acting in line with best practices here can open up legal liability, reporting can cause significant downsides to the reporter and patching can be more money or expose more risks than it is worth. There are trade-offs in all behaviours. We believe that bug handling needs to be treated in a proper context where not reporting or not patching is truly considered.

2 Not Reporting Bugs

Commonly, reporting security bugs ends in somebody getting paid, either an independent security researcher reporting via a bug bounty program or a paid security researcher disclosing via their employer, a professional auditing service. However, not all bugs are paid bugs. Other users report bugs to fix issues out

[1] This is mildly controversial in terms of advice that countries give about patching [27].

of concern for the ecosystem or other benevolent reasons. Existing work considers this trade-off from a nation-state perspective [1, 8, 22]. More broadly, Laube and Böhme survey the literature in cybersecurity information sharing, including a brief consideration of non-reporting of bugs [19]. We consider a variety of malicious and non-malicious reasons to not report bugs.

Bureaucracy Many systems make it hard to report bugs directly to them. Some bugs are reported by security researchers reporting it to their friend who works at the affected companies. One underappreciated asset here is the informal social network between cybersecurity professionals, many of whom have switched between bug finding and in-house defence roles. Long term relationship building skills then become invaluable for those in this field of work. For those not socially connected, open source bug trackers and corporate bug bounty programs can require a security researcher to fill out forms, sign an NDA, email back and forth with the organisation, to then not receive money in proportion with their hourly wage [6]. However, releasing a bug on social media can give a security researcher buzz about their work while allowing them to focus their efforts on security research, not bureaucracy.

Fingerprinting There's an entire class of fingerprinting bugs that are useful for anti-fraud and network defence purposes. This is sometimes done on purpose by the platform, e.g. by deliberately manipulating their software stack to detect those trying to spoof. But, other times, security researchers will find a bug on a commonly used system that will allow the researcher to uniquely identify some users. These bugs are much less useful after platforms patch them, so there's a strong incentive not to report so the identification vector stays open. Whether sometimes being fingerprintable is a bug is an entirely separate discussion. (This can be a feature.)

Red teaming. Similarly to fingerprinting, this class of bugs is found by security researchers for a purpose that stops being useful the moment it is reported. Red team people will keep bugs private for their own use, e.g., in professional hacking tournaments like DEF CON CTF. This common practice allows the security researcher limited use of the bug in order to gain professional advantage over other teams. Professional software penetration teams can keep a private set of unreported bugs to use against those that hire them; these can be reported after a commission is completed. Similarly, there are black hat private bugs where security researchers keep bugs private to infiltrate systems they are not authorised to enter. Both of these are active methods which will end up leaking the existence of the bug eventually.

Bad faith vendors. Not all vendors act in good faith after bug reports. The cryptocurrency, Iota, publicly smeared researchers for reporting a bug to

them [5]. To combat against bad faith vendors, security researchers might publicly disclose details of bugs out of spite rather than reporting. Other researchers might publicly acknowledge the bug without privately reporting or giving sufficient details to fix it. For instance, Castellucci never disclosed the details of the keyfinding attack on the BitFi cryptocurrency hardware wallet after BitFi engaged in personal attacks against them [20]. The BitFi wallet used an insecure brainwallet scheme to protect the customers' money. By publicly demonstrating their exploit, Castellucci aimed not to secure the fundamentally flawed technology, but rather to encourage users not to buy the wallet [7].

Forensic short selling. Bugs in software can also serve as a kind of proprietary research for activist short sellers and short-biased research shops. By finding bugs and then strategically releasing information on them, a security researcher can earn quite a lot more money than would be possible via a bug bounty program. For example, Muddy Waters is a short selling investment research firm that deployed this strategy against a medical device company, Abbott (formerly St. Jude) [23]. This strategy delays the reporting of bugs to the vendor to allow time for the security researchers to profit.

Commercial use. Exploit brokers serve to make money on buying exploits from security researchers and selling them to the highest bidder. These bidders use the bought exploits to gain access to unauthorised, high value systems. Some countries use them to access the devices of dissidents. Other exploit brokers, like Cellebrite, buy or develop exploits and rent their product to "gain access to systems" to third parties, like law enforcement. There is a large class of exploits that are much more commercially viable to third parties than they are to the exploited companies.

For example, Castellucci uncovered the `goto fail` bug [17] after colleagues of theirs overheard someone at a bar drunkenly bragging about how they were going to sell it to GCHQ for "half a million". Castellucci was able to develop a working exploit based on the relayed information and had the details anonymously back channelled to Apple who released a fix. While `goto fail`'s illicit usage was thwarted, there were likely others sold off at similar price points.

3 Not Patching Bugs

Most companies have an internal system to track bugs, prioritise them, and patch them as appropriate. While these systems do not necessarily lead to strict compliance with industry norms, they often lack the consideration of not patching bugs [25]. Similarly, open source software maintainers often find vulnerabilities make them look bad [2]. Incentives are not strictly to patch or even acknowledge bugs, depending on community norms, which Ayala et al. found on their (pre-interview) open source maintainer community survey. Non-reproducible bugs reported to open source repositories have been found to not be patched [10].

We note that while many academics consider corporate decisions here, the decisions here are often made by individual engineers working in a corporation. For reasons we will lay out, there are plenty of individual incentives that come into play (along with corporate decisions that developers are given). Here we lay out some cases where it is strategic to not patch bugs in software.

Compatibility. Microsoft Excel deliberately implemented a bug for compatibility with Lotus 1-2-3 which treats 1900 as a leap year [21]. This is still included in their software today, despite Lotus 1-2-3 being last updated in 2002. They have not patched it because this would cause too many other waterfalling issues. Here, there was a weighing of correct behaviours with compatibility and sometimes compatibility wins over correct behaviour.

Might cause provocation. Some bugs are left fearing worse scenarios to arise in their stead. It can be strategic to leave bugs if they don't cause significant economic damage to allow the small number of people to exploit them *and* the people who would exploit them would do worse things. For example, wifi providers often don't block known bugs like, e.g., DNS tunnelling. Blocking DNS tunnelling could result in people provoked into worse things. They might probe the network's publicly facing computer systems or e.g. clone the MAC address of a paid user and kick a legitimate user off to use the network for free. This is particularly troubling in common scenarios like on aeroplanes.

Workflow issues. Sometimes removing bad code can cause issues for developers who use the features in less common ways. Recently, Castellucci has been trying to argue for removing footgun options from major cryptography libraries. Some of them have decided to do this and others have decided not to, overwhelmingly citing workflows [24]. For example, RSA-512 should never be used at this point since it's insecurity renders the actual crypto fairly moot. But, because some use it in unit tests, the behaviour is allowed to remain. Changes aren't necessarily neutral for all.

Low impact - high cost. There's the class of bugs where addressing the bugs would require substantial engineering work or other tradeoffs in design. Particularly when the bug is difficult to abuse and the impact is relatively minor, it can be strategic not to patch. An example is HTTPS strict transport security which has a bug that can be used as a tracking vector because it isn't subject to the same partitioning as things like cookies and cacheing [30]. Browser vendors know about this bug and acknowledge it as a theoretical issue, proposing to monitor it and patch if it becomes actually weaponised. There's a tradeoff – fixing this one bug might cause issues in another area.

Similarly, a developer who is overworked might triage their bugs to patch and have such a high workload, that they might never get to all of the bugs reported. Those which are low impact with a high cost then might be triaged low enough to never be fixed.

Future impact - high cost. This is related to the last category, except for when the bug will only occur a long time in the future. We note that Y2k bugs were in this category for quite some time until the world scrambled in the late 1990's to find and quash them. `logrotate` has a bug like this[2]. The software makes some assumptions on the file format naming conventions which mean that it will break when the UNIX epoch timestamp gets another digit. This harkens back to Y2K bugs, except 264 years might be more reasonable of a time period.

Another instance here is in the case of a startup where an individual developer might not patch a bug because it will likely not be exploited during their tenure at the company. Similarly, an entire startup might decide not to patch known bugs in the rush to push a product before they run out of startup funds.

Jailbreaks. Jailbreak bugs allow users to bypass system boundaries to gain access above what the system allows. We classically think about these where users jailbreak their phones, often voiding their End User License Agreement with the operating system software. These are primarily used by users against their own devices and the legality is varied around the world.

There's a whole spectrum of things here. Some jailbreak "features" end up being incorporated back by the platform, like expanded emojis in iOS. Other times, there's no clear security boundary being breached by the bug. For instance, some jail break-type bugs require physical access or even just administrative access. These kinds of bugs are useful or extremely useful to a certain subset of users needing root access. A team found a bug that was useful for jailbreaking Nintendo, but sat on it hoping that the device would reach EOL and it could be a permanent bug. It had minimal security implications, though it was patched two years later after it was found by another [31]. This contrasts from times when a security-sensitive bug is weaponised for a short period of time. For example, a jailbreak bug for iOS ended up causing some rooted iPhones to form a botnet [3].

Bug door. Not all bugs are found by externals. Others are developed into the software for special, extra-purpose use, e.g. back doors which are bugs or "bug doors". By their very nature, they allow deniability. For example, `brainwallet.org` allowed user to create very insecure brainwallets, opening them up for draining by attackers. There were strong suspicions online via, e.g. Gregory Maxwell, who speculated that the owner of this website was cracking brainwallets on the side[3]. `Dual_EC_DRBG` was similarly speculated to be a bug door by the NSA [4, 9, 35].

Forced obsolescence. It can be a strategic decision not to patch security bugs in order to force users to buy new software or new devices. This is planned

[2] `https://github.com/logrotate/logrotate/blob/d57dff56edce193abf7a660da3635be89e57fc8e/logrotate.c#L1892`

[3] `https://web.archive.org/web/20140308200904/https://people.xiph.org/~greg/brainwallet.txt`

obsolescence but worse. Classically, vendors of low end Android phones use this method as do vendors for consumer and prosumer level network hardware (e.g. home routers, network-attached storage). It's unclear if this is an explicit strategy or just a time-limiting mechanism without any further thought. The EU Cyber Resilience Act and similar UK regulation aims to fix this.

4 Discussion

The main thread in not patching bugs by legitimate codebases is liability.[4] Many of these had a cost benefit trade-off (not all as explicit as Microsoft) where it was deemed not worth the money to patch the system. Rather, the organisation would accept the risk, sometimes monitor it to reevaluate if it is being weaponised, and accept the underlying liability for knowing about the bug and not patching.[5]

This contrasts with the reasons for not reporting bugs. These are often delay tactics to allow the security research to monetise the bug beyond the bounty offered through an official program. Bug bounty programs are an attempt for companies to better align incentives [18]. They often make reporting easier, provide consistency, and offer a monetary reward. There is a lot of room for improvement, though. The constant flood of low quality reports can drown out legitimate bugs and issues, frustrating both organizations and researchers. Non-reproducible bugs make it tricky – is the bug report just poorly documented or is the bug a one-off? Ghosh et al. found this to be a substantive issue when considering `npm` repositories on GitHub [10]. Is it worth it to try to investigate this further or is the cost too high for strained open source maintainers?

We agree with the concerns held by Laube and Böhme with how this will play out in the Internet of Things [19]. The example of the pacemaker bug being witheld to invoke short-term stock market movement was not the end of this story; this was later patched. The patch was rolled out cautiously due to the implications of patching an implanted device and the 1% risk of the patch bricking the device [28]. Particularly, elderly people with pacemakers were much less likely to patch than others. If the bug was privately disclosed, it might not have been patched due to its *low impact - high cost* nature. Instead, it was weaponised and caused a large, perhaps unnecessary, strain on the medical system.

While we consider tradeoffs informing defender behaviour, others have already considered why attackers could strategically not attack a system [12]. This work shows that deterrence can work. We leave the task of weaving together this

[4] `https://www.schneier.com/essays/archives/2007/01/information_security_1.html`

[5] For instance, open redirects are a common bug exploited by SEO spammers or cybercriminals cloaking phishing websites or counterfeit goods websites. Google explicitly does not patch this despite awareness of the broader potential issues, explicitly outlining their tradeoffs in terms of other benefits [13]. Github's `git.io` service allowed these for awhile, but shut them down in 2022 due to the volume of abuse [11].

work with ours to future work. While attacker versus defender behaviour is often modelled as a cat and mouse game, considering alternate strategies could improve our understanding of the world in real terms beyond academic approximations.

Deployment of Patches. Much of the work considering tradeoffs in patching behaviour happens after the decision we consider does. That is, considering whether end users patch or do not patch their systems after the patch is available [33, 34]. These mirror hardships when system administrators deploy patches to their systems [14, 26]. In both cases, others find that patches are expensive to deploy, either in time to research and test patches [14] or paying a person (internal or external) to deploy patches to individual devices [16]. It is possible to write code with an explicit expiration date to get around the predisposition to not deploy patches – for example, the messaging app, Signal, forces the expiration of its binaries 90 days after release [29]. We do not explicitly consider the deployment of patches in our model. However, these decisions mirror the decisions captured in our ontology on why companies might not patch vulnerabilities in the first place – it is expensive, particularly if low or future impact.

5 Conclusion

Many strategies here are illegal. These are clearly bad and we strongly recommend against employing illegal strategies. Some surprisingly are legal – the Muddy Waters example is a legitimate company operating within the rules of the US. But, like we saw with jailbreaks, the rest are largely in a legal grey area.

Patching advice often feels like a rational astrology that happens to work [15]. We patch because of bureaucratic inertia, a term that Kelsey and Schneier coin for security advice we follow because "it's always been done this way". Our work attempts to break down when to ignore this belief system (or at least companies or individual software developers do).

At any given time, there is a significant number of known vulnerabilities that either have not been disclosed or have not been patched. Academics should consider these behaviours in their models of incentivising patching. Similarly, academics should consider engineering incentives to reduce these behaviours, particularly those which are security sensitive.

References

1. Anderson, R.: Why information security is hard – an economic perspective. In: Seventeenth Annual Computer Security Applications Conference. pp. 358–365. IEEE (2001)
2. Ayala, J., Tung, Y.J., Garcia, J.: A mixed-methods study of open-source software maintainers on vulnerability management and platform security features. In: Usenix Security (2025)
3. BBC News: New iPhone worm can act like botnet say experts (2009), `http://news.bbc.co.uk/1/hi/technology/8373739.stm`

4. Bernstein, D.J., Lange, T., Niederhagen, R.: Dual EC: A standardized back door. In: LNCS Essays on The New Codebreakers, pp. 256–281. Springer (2016)
5. Böhme, R., Eckey, L., Moore, T., Narula, N., Ruffing, T., Zohar, A.: Responsible vulnerability disclosure in cryptocurrencies. Communications of the ACM **63**(10), 62–71 (2020)
6. Byfield, B.: Why I rarely file bug reports. Linux Magazine (2014), https://www.linux-magazine.com/Online/Blogs/Off-the-Beat-Bruce-Byfield-s-Blog/Why-I-rarely-file-bug-reports
7. Castellucci, R.: Bitfi's hardware wallet is terrible (2018), https://rya.nc/bitfi-wallet.html
8. Caulfield, T., Ioannidis, C., Pym, D.: The US vulnerabilities equities process: An economic perspective. In: Decision and Game Theory for Security. pp. 131–150. Springer (2017)
9. Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohney, S., Green, M., Heninger, N., Weinmann, R.P., Rescorla, E., Shacham, H.: A systematic analysis of the Juniper Dual EC incident. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 468–479 (2016)
10. Ghosh, R., De, S., Mondal, M.: "I wasn't sure if this is indeed a security risk": Data-driven understanding of security issue reporting in GitHub repositories of open source npm packages. In: Usenix Security (2025)
11. Github: Git.io no longer accepts new urls (2022), https://github.blog/changelog/2022-01-11-git-io-no-longer-accepts-new-urls/
12. Heitzenrater, C., Taylor, G., Simpson, A.: When the winning move is not to play: Games of deterrence in cyber security. In: Decision and Game Theory for Security. pp. 250–269. Springer (2015)
13. Hunters, G.B.: Google and alphabet vulnerability reward program (vrp) rules, https://bughunters.google.com/about/rules/google-friends/6625378258649088/google-and-alphabet-vulnerability-reward-program-vrp-rules
14. Jenkins, A.D.G., Liu, L., Wolters, M.K., Vaniea, K.: Not as easy as just update: Survey of system administrators and patching behaviours. In: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. CHI '24, Association for Computing Machinery, New York, NY, USA (2024)
15. Kelsey, J., Schneier, B.: Rational astrologies and security. RossFest Festschrift (2025)
16. Kustosch, L., Gañán, C., van Eeten, M., Parkin, S.: Patching up: Stakeholder experiences of security updates for connected medical devices. In: Usenix Security (2025)
17. Langley, A.: Apple's SSL/TLS bug, https://www.imperialviolet.org/2014/02/22/applebug.html
18. Laszka, A., Zhao, M., Malbari, A., Grossklags, J.: The rules of engagement for bug bounty programs. In: Financial Cryptography and Data Security. pp. 138–159 (2018)
19. Laube, S., Böhme, R.: Strategic aspects of cyber risk information sharing. ACM Computing Surveys **50**(5), 1–36 (2017)
20. Leyden, J.: C'mon, if you say your device is 'unhackable', you're just asking for it: Bitfi retracts edgy claim. The Register (2018), https://www.theregister.com/2018/08/31/bitfi_reluctantly_drops_unhackable_claim/
21. Microsoft 365 Troubleshooting: Excel incorrectly assumes that the year 1900 is a leap year (2024), https://learn.microsoft.com/en-us/office/troubleshoot/excel/wrongly-assumes-1900-is-leap-year

22. Moore, T., Friedman, A., Procaccia, A.D.: Would a 'cyber warrior' protect us: exploring trade-offs between attack and defense of information systems. In: New Security Paradigms Workshop (NSPW). pp. 85–94. ACM (2010)
23. Muddy Waters Capital LLC: MW is Short St. Jude Medical (STJ:US) (2016), https://muddywatersresearch.com/research/stj/mw-is-short-stj/
24. Munroe, R.: xkcd: Workflow (2013), https://xkcd.com/1172/
25. ten Napel, G., van Eeten, M., Parkin, S.: Speedrunning the maze: Meeting regulatory patching deadlines in a large enterprise environment. In: 2025 IEEE Symposium on Security and Privacy (2025)
26. Nurse, J.: To patch or not to patch: Motivations, challenges, and implications for cybersecurity. In: International Conference on Human-Computer Interaction. pp. 265–281. Springer (2025)
27. Ruth, K., Obu, R.B., Shode, I., Li, G., Gates, C., Ho, G., Durumeric, Z.: A first look at governments' enterprise security guidance. In: Usenix Security (2025)
28. Saxon, L.A., Varma, N., Epstein, L.M., Ganz, L.I., Epstein, A.E.: Factors influencing the decision to proceed to firmware upgrades to implanted pacemakers for cybersecurity risk mitigation. Circulation **138**(12), 1274–1276 (2018)
29. Signal Support: Open signal on your phone to keep your account active, https://support.signal.org/hc/en-us/articles/9021007554074-Open-Signal-on-your-phone-to-keep-your-account-active
30. Syverson, P., Traudt, M.: HSTS supports targeted surveillance. In: 8th USENIX Workshop on Free and Open Communications on the Internet (2018)
31. thejsa: Hacker news (2024), https://news.ycombinator.com/item?id=39983964
32. UK National Cyber Security Center: The problems with patching (2019), https://www.ncsc.gov.uk/blog-post/the-problems-with-patching
33. Vaniea, K., Rader, E., Wash, R.: Betrayed by updates: how negative experiences affect future security. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. p. 2671–2674. CHI '14, Association for Computing Machinery, New York, NY, USA (2014)
34. Vaniea, K., Rashidi, Y.: Tales of software updates: The process of updating software. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. p. 3215–3226. CHI '16, Association for Computing Machinery, New York, NY, USA (2016)
35. Wyden, R., Lee, M.S., Booker, C.A., Nadler, J., Thompson, B.G., Lieu, T.W., Lofgren, Z., Payapal, P., Malinowski, T., Eshoo, A.G., Foster, B., Khanna, R., DelBene, S.K.: Letter to Rami Rahim (2020), https://www.wyden.senate.gov/imo/media/doc/061020%20Wyden%20Led%20Bicameral%20Juniper%20Letter.pdf