Editor: Sean Peisert, sppeisert@lbl.gov

SolarWinds and the Challenges of Patching: Can We Ever Stop Dancing With the Devil?

Fabio Massacci | University of Trento and Vrije Universiteit Amsterdam
Trent Jaeger | Pennsylvania State University

Editor's Note

The SolarWinds hack shows the limits of our security practices: damned if you patch, damned if you don't. Fabio Massacci and Trent Jaeger discuss whether we should change our current attitude to patching by debating at the two ends of the spectrum.

Point

The Right to Stay Unpatched and the Need to Design for Failures

Fabio Massacci

The SolarWinds hack is an eye-opener to the current practices of the software industry. In the "Perspectives" department in this issue, some of *IEEE Security & Privacy*'s editorial board members discusse the issue of software supply chain security. Here, I would like to discuss a point that seems to be missing, including the following observations and question:

- *Observation 1*: "Update your software" is the strongest commandment of the current security religion.
- *Observation 2*: A legitimate update can introduce a new vulnerability into a system.
- *Question*: Are updates really necessary?

©SHUTTERSTOCK/ASCANNIO

We cannot even decide not to update. For example, Windows 10 allows us only to delay an update but not to forgo it. Updates are often cumulative (this is a feature, and SolarWinds is no exception[3]), so we cannot just sky jump to the hotfix we need; we must take an update lock, stock, and barrel, including its vulnerabilities. As a follow-up to SolarWinds, regulators should grant users the right to stay unpatched

Digital Object Identifier 10.1109/MSEC.2021.3050433
Date of current version: 15 March 2021

 Copublished by the IEEE Computer and Reliability Societies 1540-7993/21©2021IEEE

and move responsibility for confining security spillovers to vendors. This would push our community toward better solutions—which we do have but that are less convenient for software companies.

Updates are bundled in the interest of vendors, and by adding functionalities, new vulnerabilities are introduced. One could illustrate this with forced updates from Microsoft, Google, Apple, and Facebook, but we will stick to SolarWinds to keep the discussion focused. See "The SolarWinds Patching Schedule and Its Demise" for a summary; Table S1 provides us with a schedule of updates. Focus on versions 2019.4 and 2019.2. They are not vulnerable to the SUNBURST malicious code. If a user did not need any of the (nine out of 38) products subject to hotfix 5, he or she might not have required an update and thus would have avoided trouble (some users even complained that they lost useful features when they upgraded). From the perspective of resistance to SUNBURST, not updating was more secure. From the perspective of SUPERNOVA, any update was irrelevant.

Updates: Cui Bono? According to *Merriam–Webster*, the Roman principle of cui bono suggests that perpetrators of an act have something to gain.

- *Question (revised)*: Are updates really necessary for compliance with all possible security regulations and best practices?

I argue that the answer is mostly never. Few people benefit from updates for the simple reason that most customers don't actually use the (sub)components that are being upgraded. Indeed, one of the major security features in the 2019.2 version of SolarWinds concerned disabling administrator access without passwords. Before sending SolarWinds to the gallows, we should look at its users' blog. Out of the 535 posts about "vulnerability," a 2014 comment described default passwords as a failure of compliance. Yet, there were no follow-ups; no user said, "I also have this problem." In 2015, when SolarWinds posted a product release plan for the 2020 network configuration manager, the user interface was the top concern (nicely drawing one's network with icons for up and down nodes). It was only in 2015–2016 that SolarWinds customers started penetration-testing applications in the framework of compliance evaluations (as opposed to monitored services) and requesting fixes for specific products. They did not ask for latent improvements.

The Real Reason for Updates. The answer to a problem (security or otherwise) always is, "Update to the next version." Pick your corporation of choice and find a different "solution." I am accepting entries from readers. My own experience—both as an individual customer and as a deputy director of a metropolitan area network, 70-plus people, and a budget worth a few million euros—has always been that after "For English, press one," there was a "For support, update to the next version, and only then press two." Curiously, updating typically requires "To pay the new license, press one" for features I did not know even existed and will never use. The most fascinating invoice my group received from a multinational corporation was for a "license maintenance fee." (To the English purists: the order of the words was correct; the maintenance license fee appeared on another invoice.)

As customers, we may expect updates to fix bugs and possibly introduce new functionalities. In contrast to hardware, software makes that possible. If an old car were software, the vehicle could be automatically retrofitted with proximity sensors, and an ugly seat cover could be replaced. Yet, we would have to accept that the brake and accelerator pedals could be swapped at a moment's notice and that a luggage rack could be added to the roof. In other words, all software users face "generic updates" in which they are given only one choice: "accept all changes." This is not necessary. Software vendors can check whether a component in a bundle has ever been used and whether there is a need to change it.

Not Updating Can Make (Scientific) Sense. While the idea of not updating seems unscientific, in several empirical studies[4,5] I have performed with my colleagues to examine open source software vulnerabilities (from the major browsers[4] to the free and open source software ecosystem[5]), we found that it is sensible. Indeed, the key observation from Dashevskyi et al.[5] can be summarized as

- *Observation 3*: Vulnerabilities are discovered in the latest versions of software, and if your version is old enough, the vulnerable code is simply not there. No code, no exploit.

Code changes dramatically, which can be detrimental to security. For example, in Apache Tomcat, a calendar year might include hundreds, if not thousands, of application programming interface and code changes (see Dashevskyi et al.,[5] Figures 2, 10, and 11). In 2014, a vulnerability, CVE-2014-0033, was discovered in the then-latest version of Tomcat and fixed through revision 1558822. However, revisions prior to that, including 1149130 from 2011, were not vulnerable to CVE-2014-0033, as they did not include the exposed feature. Old age can itself be a cure. Obviously, if vulnerable code is there, you might be exposed, but it is not necessarily true that a vulnerability is exploitable.

The SolarWinds Patching Schedule and Its Demise

To summarize the facts of the case,[1,2] SolarWinds offers a set of network and infrastructure monitoring services that has slowly grown through several acquisitions. OpenPlatform is actually an aggregation of 50-plus subcomponents (out of those products, 18 are vulnerable, and the rest are not). Because the SolarWinds software supply chain has been compromised, attackers can smuggle malware within a legitimate signed update.[1] Given the system administration nature of SolarWinds, bad actors find themselves with high-level privileges. Lateral movements enable them to pollute victims' authentication infrastructures, often beyond any repair other than razing the systems and starting from scratch.[2] Table S1, reconstructed from SolarWinds release notes,[1,3] shows the schedule of updates for OpenPlatform. We see from the table that only a few components out of 38 have been the subject of conceptual updates. Sometimes the same components have been patched and repatched. For example, in version 2020.2.1, the network traffic analyzer component was patched three times.

Table S1. The SolarWinds patching schedule.

Version	Patch	Date (yyyy/mm/dd)	Newly patched	Carried patches	SUNBURST	SUPERNOVA
2020.2.1	HF 2	2020/12/15	6/38	2/38		
2019.4	HF 6	2020/12/14		9/38		
2020.2.1	HF 1	2020/10/29, 2011/04/25	5/38			X
2020.2.1						X
2020.2	HF 1	2020/06/24–30, 2020/07/08	5/38		X	X
2020.2					X	X
2019.4	HF 5	2020/03/26		9/38	X	X
2019.4	HF 4	2020/02/05–07	3/38	8/38		X
2019.4	HF 3	2020/01/09		8/38		X
2019.4	HF 2	2019/12/18–20	3/38	5/38		X
2019.4	HF 1	2019/11/25	5/38			X
2019.4						X
2019.2	HF 3	2019/09/23–30	3/38	9/38		X (P)
2019.2	HF 2	2019/07/31, 2019/08/02	4/38	5/38		X
2019.2	HF 1	2019/06/26, 2019/07/11	6/38			X
2019.2						X

This table lists the update schedule for SolarWinds patches. The "Newly patched" column shows which product (out the 38 making up this "aggregation" of components) were actually changed. The "Carried patches" column indicates the number of components that were brought forward. An X *in the last two columns means a version is vulnerable to SUNBURST or SUPERNOVA. A parenthetical* P *specifies that a patch is available; for other components, the only solution is to upgrade to the latest hotfix.*

From the compliance perspective, it is far simpler to say version X is vulnerable and that "all previous versions" are, too. A vendor or a security auditor can cover one's back.

Design for Failures as the Solution. Even if there is a vulnerable component, I argue that it should still be possible to run it without catastrophe. In the same way, on a sunny day, we can drive a car that has a broken windshield wiper without all four tires exploding. Software should be designed with failures in mind so that if a component were exploited, the breach would be confined. A hacker can infiltrate SolarWinds network maps? Nice. He or she should be able only to redraw poor maps and show funnier icons, not gain control of authentication infrastructure. A hacker created a document that takes control of Microsoft Word (CVE-2019-1201)? Cool, so what? He or she should not be able to do anything besides introducing errors to text formats. The right solution to a security vulnerability in a word processor does not include updating an entire productivity suite, including the email client. Word can fail without dragging the world down with it.

As a security community, we have alternatives—for example, automatic network segmentation,[6] monitoring and restarting an application,[7] running services that can limit escalation,[8] automatically generating diverse applications instances,[9] and execution confinement[10]—so that even if a single software application is exploited, an attacker cannot achieve much beyond exploiting one part of the kit. Yet, software updates are so much more convenient and cheaper for vendors

The Right to Stay Unpatched. Regulators should make software vendors liable for security spillovers that go beyond a vulnerable application component. As soon as that happened, we would see solutions that were discarded as impractical be revived and receive an engineering boost. Updates as the sole solution serve only the software industry. Unbundling functionality and security should make the purpose of an update clear and provide choices. Giving users a legal "right to stay unpatched" would prompt vendors to find a better solution.

Counterpoint

Software Updates: We Can't Live Without Them, but How Do We Live With Them?

Trent Jaeger

Fabio's premise is that generic software updates are almost never beneficial for individual customers and hence are not necessary in many (nearly all?) cases. Thus, the exposure to the SolarWinds Orion Code compromise and many other future attacks would be avoided if customers did not apply updates. However, in the current software ecosystem, vendors expect to produce updates, and customers expect to apply those updates at some point in the not-too-distant future, albeit not necessarily immediately. Why is this the case? I find two valid reasons for software updates that provide benefits both to vendors and customers to maintain this equilibrium. However, the process of software updating is still fraught with peril. Ultimately, just as product development is evolving to apply techniques to reduce the number of flaws in software (e.g., by fuzz testing), software maintenance will also need to evolve to enforce discipline on updating to restrict its attack surface.

Software updates provide an opportunity to remove latent flaws. Vendors and customers both benefit from updates that remove such flaws. To customers, such updates are largely invisible, as they do not aim to impact the expected functionality, but all users could gain from them by avoiding exploitation of these hidden vulnerabilities. Vendors profit from updates that reduce the likelihood that their products will be compromised—when their updates actually achieve that goal—but they also generally aim to keep such repairs invisible beyond the broad statement of keeping systems more secure. From discussions with vendors, my understanding is that companies proactively combine flaw repair with functionality enhancements to make it more difficult for adversaries to identify susceptibilities worthy of investigation.

Software updates also include desirable new features. An advantage that software has over hardware, such as cars, is that new features can be incrementally introduced. Customers have come to expect new features via updates, and vendors certainly promote updates for the features they provide. As one recent example, the Mac OS X Big Sur update[11] highlights several "all-new features" as the main reason to apply it. Using updates to obtain new functionality is certainly an improvement for customers over having to buy new releases. I would have loved to get heated seats and proximity sensors as an update to my old car, rather than having to buy a new one. Now that software updates are the norm, it would be impractical to expect users to stick with old feature sets when new functions are easily obtained.

The problem in the SolarWinds case and for software updating in general is that software product development and its maintenance present a significant attack surface that vendors fail to track systematically, leaving opportunities for adversaries. While updates may introduce new features that are buggy and/or

malicious, as Fabio indicates, it is unclear that this problem is changed by the mode of delivery, whether in major releases versus incremental changes. Rather, these problems are inherent to our current approach in managing software development, where products may be released (either in releases or updates) with flaws. In SolarWinds, a particular update introduced the malware, but the malicious code could have been introduced in a major release instead (see "The SolarWinds Patching Schedule and Its Demise"). Reviews of the recent Cyberpunk 2077 release called the release buggy, but it was not an update.

Fabio raises an interesting point in that customers may not need many of the features in an update. However, this problem is also not specific to updates. Back when the SQL Slammer worm hit, a number of my colleagues at IBM Research were surprised to find that their computers were compromised, but they seemed more surprised that their PCs were running an SQL server they never used, which was installed with the operating system distribution of the time. Thus, we have come to find that unnecessary functionality should be turned off. However, rather than forgoing all features to avoid some, perhaps other solutions are warranted. Perhaps features can remain inoperable until explicitly needed. Such an approach to enabling features would need to avoid usability problems, such as frequent user notifications.

Unfortunately, current technologies to validate code provenance, such as code signing and the measured boot, were insufficient to detect the SolarWinds hack because the software supply chain was compromised. A question is how technologies being investigated now may be brought to bear to aid vendors in protecting their supply chains and customers in restricting new features. For example, to help customers avoid compromises from updated features, existing functionalities may be protected from new and modified ones by using isolation techniques, such as privilege separation.[12] Automated support for privilege-separating programs is advancing. For example, we have developed techniques that automate the marshaling of dynamically sized data structures (e.g., arrays)[13] and enable developers to balance performance and security.[14]

However, if updated features require access to sensitive data, privilege separation cannot protect that information. In this case, vendors must comprehensively vet those updates. One approach is to automate patching mechanisms to meet security properties. For example, we have recent work to validate that patches comply with memory safety,[15] although a more extensive set of properties will be required.

In addition to failings in the supply chain, intrusion detection systems (IDSs) also failed to detect the SolarWinds attack. According to a summary by FireEye,[16] the SUNBURST back door communicated with third-party servers via HTTP. Since HTTP requests to arbitrary servers are common, the firewall and the IDS did not flag this behavior. Such conduct was likely unexpected in the context of any updated SolarWinds feature. This shows that there is still a significant gap between application anomalies and what can be recognized by IDSs. We have proposed an approach that makes IDSs sensitive to threats in the program, host, and network layers[17] to improve the context awareness of detection methods. However, each of these directions remains a single point in a multidimensional space of in-depth defense that will be required to prevent future attacks. Software vendors are slowly adopting these defenses, but the rate of improvement continues to lag behind the threats. How vendors can adopt safeguards into their development processes more quickly and effectively remains a major challenge.

Joint Conclusions

Ignoring updates is a gamble, much as applying updates is a gamble. In either case, this roll of the dice is a symptom of our insufficient approaches to software development and maintenance on one side and intrusion detection and confinement on the other. We all have more work to do to gain the benefits of software and its updates without the risk. The SolarWinds hack is a wakeup call that a silver bullet does not exist and that innovative mixes of technical, organizational, and regulatory solutions might be the way forward. We look forward to hearing your opinions. ■

Acknowledgments

Fabio Massacci's work was supported, in part, by the European Commission, through grants 830929 (H2020-CyberSec4Europe, https://cybersec4europe.eu) and 952647 (H2020-AssureMOSS, https://assuremoss.eu). Trent Jaeger's work was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement W911NF-13-2-0045 (ARL Cyber Security CRA) and National Science Foundation grants CNS-1801534 and CNS-1801601. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory, the U.S. Government, or the European Commission. The U.S. Government is authorized to reproduce and distribute reprints for Government

purposes notwithstanding any copyright notation here on.

References

1. "Alert (AA20-352A): Advanced persistent threat compromise of government agencies critical infrastructure, and private sector organizations." Cybersecurity and Infrastructure Security Agency. https://us-cert.cisa.gov/ncas/alerts/aa20-352a (accessed Dec. 29, 2020).
2. "Emergency directive 21-01." Cybersecurity and Infrastructure Security Agency. https://cyber.dhs.gov/ed/21-01/ (accessed Dec. 29, 2020).
3. "Orion platform release notes." Solar Winds. https://support.solarwinds.com/SuccessCenter/s/article/Orion-Hotfix-Release-Notes (accessed Dec. 29, 2020).
4. V. H. Nguyen, S. Dashevskyi, and F. Massacci, "An automatic method for assessing the versions affected by a vulnerability," *Empir. Softw. Eng.*, vol. 21, no. 6, pp. 2268–2297, 2016. doi: 10.1007/s10664-015-9408-2.
5. S. Dashevskyi, A. D. Brucker, and F. Massacci, "A screening test for disclosed vulnerabilities in FOSS components," *IEEE Trans. Softw. Eng.*, vol. 45, no. 10, pp. 945–966, 2018. doi: 10.1109/TSE.2018.2816033.
6. N. Wagner et al., "December. Towards automated cyber decision support: A case study on network segmentation for security," in *Proc. 2016 IEEE Symp. Series Computat. Intell. (SSCI)*, pp. 1–10. doi: 10.1109/SSCI.2016.7849908.
7. J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Failure resilience for device drivers," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN'07)*, 2007, pp. 41–50.
8. R. N. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for UNIX," in *Proc. USENIX Security Symp.*, vol. 46, 2010, p. 2.
9. A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, and M. Franz, "Large-scale automated software diversity—program evolution redux," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 2, pp. 158–171, 2015. doi: 10.1109/TDSC.2015.2433252.
10. X. Xu, M. Ghaffarinia, W. Wang, K.W. Hamlen, and Z. Lin, "CONFIRM: Evaluating compatibility and relevance of control-flow integrity protections for modern software," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 1805–1821.
11. "macOS Big Sur." Apple. 2020. https://www.apple.com/macos/big-sur/
12. N. Provos, M. Friedl, and P. Honeyman, "Preventing privilege escalation," in *Proc. USENIX Security Symp.*, 2003.
13. S. Liu, G. Tan, and T. Jaeger, "PtrSplit: Supporting general pointers in automatic program partitioning," in *Proc. ACM CCS*, 2017, pp. 2359–2371. doi: 10.1145/3133956.3134066.
14. S. Liu et al., "Program-mandering: Quantitative privilege separation," in *Proc. ACM CCS*, 2019, pp. 1023–1040. doi: 10.1145/3319535.3354218.
15. Z. Huang, D. Lie, G. Tan, and T. Jaeger, "Using safety properties to generate vulnerability patches," in *Proc. IEEE Symp. Security Privacy*, 2019, pp. 539–554. doi: 10.1109/SP.2019.00071.
16. "Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with SUNBURST backdoor." FireEye. 2020. https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html
17. F. Capobianco et al., "Employing attack graphs for intrusion detection," in *Proc. New Security Paradigms Workshop*, 2019. pp. 16–30. doi: 10.1145/3368860.3368862.

Fabio Massacci is a professor at the University of Trento, Trento, 38123, Italy, and Vrije Universiteit Amsterdam, Amsterdam, 1081 HV, The Netherlands. His research interests include experimental methods for security and privacy. He has published more than 200 peer-reviewed articles and is the editor of the "Building Security In" department in *IEEE Security & Privacy*. In 2015, he received the Ten-Year Most Influential Paper Award at the IEEE Requirements Engineering Conference, for his work on security and trust in sociotechnical systems. He is a Member of IEEE, the Association for Computing Machinery, the American Economic Association, and the Society for Risk Analysis. Contact him at fabio.massacci@ieee.org.

Trent Jaeger is a professor in the Department of Computer Science and Engineering, Pennsylvania State University, State College, Pennsylvania, 16801, USA. His research interests include systems and software security. He has published more than 150 research papers and the book *Operating Systems Security*. He is the consortium lead for the Army Research Laboratory's Cybersecurity Collaborative Research Alliance and a member of the Association for Computing Machinery Special Interest Group on Security, Audit, and Control executive committee; the Network and Distributed System Security Symposium steering committee; the *Communications of the ACM* editorial board; and the United Kingdom's Cyber Body of Knowledge project academic advisory board. He is an associate editor in chief of *IEEE Security & Privacy*. Contact him at tjaeger@cse.psu.edu.