Title: SolarWinds and Challenges in Patching

Overall Abstract: The SolarWinds hack shows the limit 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.

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

Fabio Massacci

Abstract: As a follow-up of the SolarWinds hack, regulators should grant users the right to stay unpatched and move the responsibility for confining security spillovers to vendors. This will push our community towards *better* solutions that we *do* have but are so less convenient (for software vendors).

THE SOLARWINDS HACK is an eye opener of the current practices of the software industry. Elsewhere we discuss the issue of the security of the software supply chain, I would like to discuss here a point that seems to be missing in the current discussion.

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

We cannot even decide not to update. For example Microsoft Windows 10 only allows us to delay an update but not to forgo it. Updates are often cumulative (this is a feature and SolarWinds is no exception [3]) so that you cannot just skyjump to the hotfix you need and you must take the update lock, stock, and barrel\ldots and vulnerabilities.

Updates are bundled in the interest of the software vendor, and by adding new functionalities new vulnerabilities are also introduced. One could illustrate it on the forced updates by Microsoft or Google or Apple or Facebook, or… but we stick to SolarWinds to keep discussion focussed. Table 1 shows us the schedule of updates.

A PRIMER ON SOLARWINDS.

To sum the facts of the case [1,2], SolarWinds offers a set of network and infrastructure monitoring services that has slowly grown over several acquisitions. The OpenPlatform is actually an aggregation of 50+ subcomponents (out of those products 18 are vulnerable and the rest are not). As SolarWinds software supply chain has been compromised, an attacker has been able to smuggle malware within a legitimate signed SolarWinds update [1]. Given the sys admin nature of the SolarWinds software, the attackers have found themselves with high-level privileges. Lateral movements allowed to pollute the victims’ authentication infrastructures often beyond other repair than razing it to the ground and starting from scratch [2].

Table 1, reconstructed from the SolarWinds release notes [1,3], shows the schedule of updates for the OpenPlatform.. We see from this table that only a few components have been really the subject of conceptual updates out of 38 components covered by SolarWinds platform updates. Sometimes the same components is patched and repatched again. For example in 2020.2.1 the NTA component is patched three times.

TABLE 1 - SolarWinds Patching Schedule

This table shows the update schedule (yyy/mm/dd) for the patches of the SolarWinds platform, The "Newly Patched" shows which product (out the 38 making up this "aggregation" of components) were actually changed. The carried over patches are the components that were brought forward. A X in the last two columns means the version is vulnerable to SUNBURST (SB column) or SUPERNOVA (SN column). A (p) means a patch is available, for the other components the only solution is to upgrade to the latest hotfix on top of the table. Only a few components have been really the subject of conceptual updates out of 38 components covered by SolarWinds platform. Sometimes the same components is patched and repatched again (e.g. in 2020.2.1 the NTA component is patched three times)

Ver.	**Patch**	**Date yyy/mm/dd**	**Newly Patched**	**Carried Patches**	**SB**	**SN**
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, 11/04-25	5/38			X
2020.2.1						X
2020.2	HF 1	2020/06/24-30, 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, 08/02	4/38	5/38		X
2019.2	HF 1	2019/06/26, 07/11	6/38			X
2019.2						X

Focus on version 2019.4 and 2019.2. These versions are not vulnerable to SUNBURST. If a customer did not need to use any of the (nine out of 38) products subject to Hotfix 5 it might not have needed to do an update and thus stayed out of trouble (and some customers even complained that they had lost useful features when they upgraded).

From the perspective of resistance against SUNBURST *not updating was more secure*. From the perspective of SUPERNOVA *any update was irrelevant*.

UPDATES, CUI BONO?
According to the Merriam-Webster dictionary, this Roman principle points the likely responsibility of an act to those having something to gain.

- **Question (revised)**: Are updates really necessary for the specific customer who is doing the update, to be compliant with all possible security regulations and best practices?

I would argue that the answer would be mostly never. Few customers would benefit, most don't, *for the simple reason they don't actually use the (sub)components that are being updated.*

Indeed, one of the major security features in the 2019.2 version of the platform was disabling admin access without passwords. . . Before sending SolarWinds to the gallows, we should look at SolarWinds' users blog. Out of the total 535 posts about "Vulnerability", a comment on default passwords dates back to 2014 as a failure of compliance. Yet, no follow-up on this, no user said "I also have this problem". In 2015 when the company posted a plan for product release in 2020 of the Network Configuration Manager the user interface was the top concern (nicely drawing
one's network with icons for up and down nodes).
It is only in 2015-16 that SolarWinds customers started to pentest SolarWinds applications in the framework of compliance tests as well (as opposed to services monitored by SolarWinds and asked for fixes to SolarWinds *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'm accepting entries from readers.
My own experience with other corporations both as an individual customer and a deputy rector in charge of a Metropolitan Area Network, 70+ people and few MEuro of budget has always been that after "For English, press 1" there was a "For Support, update to the Next Version and ***only then*** press 2". Curiously, updating typically requires "To pay the New License, press 1" for features you didn't even know existed and will never use.
The most fascinating invoice from a multinational corporation we got was for "License Maintenance Fee" (Warning to the English purist, the order of words is correct, the Maintenance License Fee was another invoice).
As customers we may expect updates to bugs and possibly new functionality. In contrast with hardware, software makes that possible. If our old car was a software, it could be automatically retrofitted with proximity sensors and our ugly seat cover replaced. Yet, we will also have to accept that brake and accelerator pedals be swapped on a moment's notice, and a skies container added on the roof. In other words ***all users*** are facing "generic updates" in which they are given ***only one choice***: "accept ***all***

changes”. This is not necessary and software vendors can perfectly check that a component in the bundle has never been invoked and there is no need to change it or install it.

NOT UPDATING CAN MAKE (SCIENTIFIC) SENSE

While this idea of not updating seems unscientific, in several empirical studies [4,5] I have performed with my colleagues on open source software vulnerabilities (from the major browsers [4] to the FOSS ecosystem [5]) we found out that this is a sensible idea. Indeed the key observation from [5] is

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

Code changes dramatically and this can also be beneficial for security. For example, in Apache Tomcat a calendar year might yield hundreds, if not thousands of API and code changes (See Figures 2, 10, and 11 in [5]). In 2014, when a vulnerability was discovered in the (then latest) version of Apache Tomcat 6 (CVE-2014-0033) and fixed at revision 1558822 on 16/01/2014, the revisions prior and including 1149130 from 21/07/2011 are not vulnerable to CVE-2014-0033, as the vulnerable feature is not present in these revisions. ***Old age can by itself be a cure.***

Obviously, if vulnerable code is there, you might be vulnerable to old vulnerabilities but it is not necessarily true that a vulnerability is also exploitable. From the perspective of compliance, it is far simpler to say Version X is vulnerable and ``all previous versions’’. A vendor or a security auditor covers one’s back at no cost.

DESIGN FOR FAILURES AS THE SOLUTION

Even if you have a vulnerable component, I argue that it should still be possible to run it without a catastrophe. In the same way we can still run a car on a sunny day with a broken windshield wiper without all four tires exploding.

Software should be designed with failures in mind so that if a component is exploited, the exploit should stay confined. A hacker can hack SolarWinds Network Maps? Nice, it should only be able to redraw poor maps and show funnier icons. It should not be able to get control of your authentication infrastructure. A hacker created a document that gets control of Microsoft Word (CVE-2019-1201)? Cool, so what? One should not be able to do anything besides misformatting your documents…

The right solution of a security vulnerability in a word processing document should not be to update to the next version of the entire productivity suite including the email client, it is a software environment where Word can fail without dragging the World with it.

As a security community we do have alternatives: for example, automatic network segmentation [6], monitoring and restarting an application fresh [7], running services with capabilities to limit escalation [8], automated generation of diverse applications instances [9], or execution confinement [10] etc. so that even if a single software application is exploited, an attacker cannot get much besides exploiting the single part of the kit. Yet software updates are so more convenient and cheaper for software vendors

THE RIGHT TO STAY UNPATCHED

Regulators should make software vendors liable for spillover of security that go beyond the vulnerable application component. As soon as this liability stick will be available then we will see how the solutions above that are discarded as not practical will see a boost in engineering to address the practical issues. Update to the next version as the only solution only serves a poor software industry. Unbundling functionality and security should make it clear what is the purpose of an update and will give users a choice. Give users a legal “Right to Stay Unpatched” and the software industry will find a (better) solution.

Software Updates: 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 compromises would be avoided if customers did not apply software updates.

However, in the current software ecosystem, vendors expect to produce software 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 benefit both to vendors and their 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 the software products (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: Both vendors and customers benefit from updates that remove flaws from software products. To customers, such updates are largely invisible, as they do not aim to impact the expected functionality, but all customers could benefit from such updates by avoiding exploitation of these latent vulnerabilities. Vendors benefit from updates that reduce the likelihood that their products will be compromised, when their updates actually achieve that goal, but vendors also generally aim to keep such flaw repair invisible beyond the broad statements of keeping your system more secure. From my discussions with vendors, my understanding is that vendors proactively combine flaw repair with functionality enhancements in updates to make it more difficult for adversaries to identify flaws worthy of investigation.

Software updates provide desirable new features: An advantage that software products have over hardware products, such as cars, is that new features can be introduced incrementally via updates. Customers have come to expect new features via updates and vendors certainly promote updates for the features that 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 the update. Using software updates to obtain new functionality is certainly an improvement for customers over having to buy a new release. I would have loved to get heated seats or 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 will be impractical to expect users to stick with old feature sets when new features 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 software 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 of software, whether in major releases versus updates. Rather, these problems are inherent to our current approach in managing software development, where software may be released (either in releases or updates) with flaws. In the SolarWinds case, a particular update introduced the malware, but it could have been introduced in a major release instead. The recent Cyberpunk 2077 release infamously includes buggy code, but was not an update.

Fabio raises an interesting point that customers may not need many of the features in an update. However, this problem is also not specific to updates. Back when SQL Slammer 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 PC was running an SQL Server that they never used, installed with the OS 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 by customers. However, such an approach to enable features would need to avoid usability problems, such as frequent notification of the kind users faced when granting permissions to mobile apps in the recent past.

Unfortunately, current technologies to validate code provenance, such as code signing or measured boot, were insufficient to detect the SolarWinds hack because the supply chain of the software was compromised. A question is how technologies being investigated now may be brought to bear to aid vendors in protecting their supply chain and customers in restricting new features.

For example, to help customers avoid compromise from updated features, existing features may be protected from new or modified features using isolation techniques, such as *privilege separation* [12]. Automated support for privilege separating programs is advancing. For example, we have developed techniques recently that automate marshaling of dynamically-sized data structures (e.g., arrays) [13] and enable developers to balance performance and security [14].

However, if the updated features require access to sensitive data, privilege separation cannot protect that data if the updated features are malicious. In this case, vendors must vet their software and any updates comprehensively, even from insiders. 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 (IDS) also failed to detect the SolarWinds attack. According to a summary of the SUNBURST Backdoor by FireEye [16], the backdoor communicated with third-party servers via HTTP. Since HTTP requests to arbitrary servers are common, the firewall and IDS did not flag this behavior. However, such behavior was likely unexpected in the context of any updated feature of the SolarWinds product. This shows that there is still a significant gap between application anomalies and what can be recognized by IDS. We have recently proposed an approach that makes IDS sensitive to threats at the program, host, and network layers in conjunction [17] to improve the context-awareness of detection methods.

However, each of these directions are still just points in a multi-dimensional space of defense-in-depth that will be required to prevent future attacks. Software vendors are slowly adopting these kinds of defenses, but the rate of improvement of defenses continues to lag behind the threats. How vendors can adopt defenses into their development processes more quickly and effectively remains as a major challenge.

Conclusions by Fabio Massacci and Trent Jager

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

ACKNOWLEDGMENTS

Dr. Massacci's work was supported in part by the European Commission by grants num. 830929 (H2020-CyberSec4Europe - https://cybersec4europe.eu) and num. 952647 (H2020-AssureMOSS - https://assuremoss.eu)

REFERENCES

1. CISA. Alert (AA20-352A) Advanced Persistent Threat Compromise of Government Agencies, Critical Infras-tructure, and Private Sector Organizations. Available at https://us-cert.cisa.gov/ncas/alerts/aa20-352a

2. CISA Emergency Directive 21-01. Available at https:// cyber.dhs.gov/ed/21-01/

3. SolarWinds. Orion Platform Release Notes. https://support.solarwinds.com/SuccessCenter/s/article/ Orion-Hotfix-Release-Notes

4. Nguyen, V.H., Dashevskyi, S. and Massacci, F., 2016. An automatic method for assessing the versions affected by a vulnerability. Empirical Software Engineering, 21(6), pp. 2268-2297.

5. Dashevskyi, S., Brucker, A.D. and Massacci, F., 2018. A screening test for disclosed vulnerabilities in FOSS components. IEEE Transactions on Software Engineer- ing, 45(10), pp. 945-966.

6. Wagner, N., Sahin, C.S., Winterrose, M., Riordan, J., Pena, J., Hanson, D. and Streilein, W.W., 2016, December. Towards automated cyber decision support: A case study on network segmentation for security. In 2016 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1-10). IEEE.

7. Herder, J.N., Bos, H., Gras, B., Homburg, P. and Tanen- baum, A.S., 2007, June. Failure resilience for device drivers. In 37th Annual IEEE/IFIP International Confer- ence on Dependable Systems and Networks (DSN'07) (pp. 41-50). IEEE.

8. Watson, R.N., Anderson, J., Laurie, B. and Kennaway, K., 2010, August. Capsicum: Practical Capabilities for UNIX. In USENIX Security Symposium (Vol. 46, p. 2).

9. Homescu, A., Jackson, T., Crane, S., Brunthaler, S., Larsen, P. and Franz, M., 2015. Large-scale automated software diversity—program evolution redux. IEEE Transactions on Dependable and Secure Comput- ing, 14(2), pp.158-171.

10. Xu, X., Ghaffarinia, M., Wang, W., Hamlen, K.W. and Lin, Z., 2019. CONFIRM: Evaluating Compatibility and Relevance of Control-flow Integrity Protections for Mod- ern Software. In 28th USENIX Security Symposium (USENIX Security) (pp. 1805-1821).

11. Apple, Inc., 2020. macOS Big Sur. https://www.apple.com/macos/big-sur/.

12. Niels Provos, Markus Friedl, Peter Honeyman, 2003. Preventing Privilege Escalation. In *USENIX Security Symposium*.

13. Shen Liu, Gang Tan, Trent Jaeger, 2017. PtrSplit: Supporting General Pointers in Automatic Program Partitioning. In *ACM CCS*, pp. 2359-2371.

14. Shen Liu, Dongrui Zeng, Yongzhe Huang, Frank Capobianco, Stephen McCamant, Trent Jaeger, Gang Tan, 2019. Program-mandering: Quantitative Privilege Separation. In *ACM CCS*, pp. 1023-1040.

15. Zhen Huang, David Lie, Gang Tan, Trent Jaeger, 2019. Using Safety Properties to Generate Vulnerability Patches. In *IEEE Symposium on Security and Privacy*, pp. 539-554.

16. FireEye, Inc., 2020. FireEye Blogs: Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor.
https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html.

17. Frank Capobianco, Rahul George, Kaiming Huang, Trent Jaeger, Srikanth V. Krishnamurthy, Zhiyun Qian, Mathias Payer, Paul Yu, 2019. Employing attack graphs for intrusion detection. In *New Security Paradigms Workshop*, pp. 16-30.

Bios:

Fabio Massacci (MSc'93, PhD'98) is professor at the University of Trento and the Vrije Universiteit Amsterdam. He has published over 200 peer review articles and is the recipient of the Ten Years Most Influential Paper Award in 2015 by the IEEE Requirement Engineering Conference for his work on security and trust in socio-technical systems. His current research interests are in experimental methods for security and privacy. He is the leader for education and professional skills of the

CyberSec4Europe pilot for the forthcoming EU CyberSecurity Competence Center and Network and the coordinator of the AssureMOSS project on the security of open source software. He is Department Editor for the ‘Building Security In’ column of the IEEE Security & Privacy Magazine. He is member of the ACM, AEA, IEEE, and the Society for Risk Analysis. Contact him at fabio.massacci@ieee.org.

Trent Jaeger is a Professor in the Computer Science and Engineering Department at The Pennsylvania State University. Trent's primary research interests are systems and software security. He has published over 150 research papers and the book, "Operating Systems Security," which has been taught in universities worldwide. Trent has made significant contributions to the Linux community, including mandatory access control, integrity measurement, process tracing, and namespace services. Trent is currently the Consortium Lead for Army Research Lab's Cybersecurity Collaborative Research Alliance (ARL CSEC-CRA). Trent also serves the computer security research community as an Executive Committee member of ACM SIGSAC as Past Chair (Chair from 2013-2017), as General Chair and Steering Committee Chair of NDSS, as an Editorial Board Member for the Communications of the ACM, as an Associate Editor-in-Chief of the IEEE Security & Privacy Magazine, and as an Academic Advisory Board member of the UK's Cyber Body of Knowledge (CyBoK) project.