Editor: Fabio Massacci, f.massacci@vu.nl

"Free" as in Freedom to Protest?

Fabio Massacci | University of Trento and Vrije Universiteit Amsterdam
Antonino Sabetta | SAP Security Research
Jelena Mirkovic | University of Southern California Information Sciences Institute
Toby Murray | University of Melbourne
Hamed Okhravi | MIT Lincoln Laboratory
Mohammad Mannan | Concordia University
Anderson Rocha | University of Campinas
Eric Bodden | Heinz Nixdorf Institute at Paderborn University and Fraunhofer IEM
Daniel E. Geer, Jr. | In-Q-Tel

Editor's Note

Open source software development is a technical job, which should make it kind of immune to bias. We trust developers to do the right (technical) thing, and this trust has become even more important as the software supply chain has become critical for software vendors who bundle or rely on open source components. The recent days proved us wrong. We discuss in this piece several different perspectives from the editorial board.

©SHUTTERSTOCK.COM/I_PHOTOS

What Happened?

Fabio Massacci and Antonino Sabetta

To kick-start the discussion, let's first review some of the recent attacks. In the node-ipc case[1] a developer pushed an update that deliberately but stealthily included code that sabotaged the computer of the users who installed the updated component. Such an attack was selective: a DarkSide in reverse. If the computer Internet Protocol (IP) was geolocated in Russia, the attack would be launched. Several days and a few million downloads later, the "spurious code" was actually noticed and investigated. Linus's law on the many eyes eventually made the bug shallow,[2] and the developer pulled back the changes.

The case of styled-components[3] is an interesting combination of a malicious but ultimately innocuous code change whose intended effect was to print a message on the console of users whose system locale was set to "ru_RU," with an accidental but disastrous mistake: the developer forgot to include a required file (postinstall.js) without which the package installation would fail, causing a chain of failures in the build of its dependent projects.[4] Only after the missing file was added, the (malicious) code could eventually "work as designed."

Pushed by a different motivation, the developer of the npm packages colors and faker[5] deliberately and openly introduced errors in their code, thus making them de facto unusable. In this case, the developer even created an issue on the software repository explaining that he could not continue working on the project for free while not actually getting any revenues out of it. So he challenged other developers to actually come to the rescue. None arrived. Apparently, the

Digital Object Identifier 10.1109/MSEC.2022.3185845
Date of current version: 6 September 2022

"free as in freedom"[6] had become "free as in free beer,"[7] or this is a sign of deeper issues in the model through which commercial vendors have interacted with the open source community so far.[8]

Editorial Board Members' Perspectives

The Polarization of Code

Jelena Mirkovic

In the past several years, we have seen a tendency in public opinion to sway widely into light-and-shadow, polarizing depictions of various social issues ranging from immigration, vaccines, and climate change to more lightweight quarrels at the Oscars. It is not surprising that wars are another polarizing issue where sides are firmly taken, and opponents are vilified and damages are bestowed upon them because "they deserve it."

Anderson Rocha

The more society lives in so-called social bubbles and opinions are in different extremes, the more we will see different forms of protest, sometimes harming public confidence even in pieces of software that historically were considered clean. It is paramount to devise forms of protecting the development of high-quality software from malicious pieces of "demonstrators." Perhaps adopting some sort of wiki-like development with more trustful developers acting as gatekeepers to guarantee good software development practices would be a starting point.

Fabio Massacci and Antonino Sabetta

We expect black-hat hackers to have mischievous tactics to avoid detection and possibly collude with corrupt or inefficient law enforcement. DarkSide, the group behind the Colonial Pipeline attack, has been famous for not installing its ransomware on machines having a Ukrainian or Russian keyboard. The final list of safe haven keyboards has been published in Brian Krebs' blog.[9] Some of us might even accept state -or police-sponsored hacking as a technically appropriate solution. Stuxnet is an example[10] (good or bad depending on which side you are on); the EncroChat hacks for the surveillance of drug cartels[11] is another one; and we might argue that the Association for Computing Machinery Code of Ethics is good in theory[12] but pretty bad on the concretely chosen example "The Rogue," where sending worms on another country's Internet service provider (ISP) received a bill of health.[13] What we have underestimated is that the "good" developers upon which the open source community is made are actually humans, and they can themselves do the equivalent of hurling bottles at the despised members of the "other party."

> In the node-ipc "impulse hacktivism attack," the service invocation looked like this:
>
> aHR0cHM6Ly9hcGkuaXBnZW9sb2NhdGlvbi5pby9pcGdlbz9hcGlLZXk9YWU1MTFlMTYyNzgyNGE5NjhhYWFhNzU4YTUzMDkxNTQ=
>
> which is the base-64 encoded form of this:
>
> https://api.ipgeolocation.io/ipgeo?apiKey=ae511e1627824a968aaaa758a5309154

Jelena Mirkovic

This is a slippery slope. Nothing in life is black-and-white, and wars especially are messy, complicated, serious, and leave long-lasting scars in generations of both winners and losers.

Hamed Okhravi

Even further, how can one ensure that the implanted version of the code base does not spread to other regions? Attackers do not have a successful track record of ensuring that a piece of malware can impact only the intended target, even when it comes to highly targeted malware; think Stuxnet.[10] How can one ensure that a piece of software that is meant to be shared does not spread beyond a certain region? IP geolocation is way too inaccurate. With the widespread usage of consumer-grade virtual private networks (VPNs) these days, IP geolocation can be even more inaccurate. Consider this scenario: a user connects to a VPN service to bypass a regional lock for a movie, and while they are watching the movie, a service on that user's machine automatically updates but grabs the infected version of a library by mistake because the machine looks like it is in a different region.

The Future(s) Ahead

Toby Murray

We might decide to consider these events exceptions; history to date has demonstrated that developers are likely to have a vulnerable product because either they didn't fix their own code or they did not keep a third-party dependency up to date, the former case being more frequent than the latter once the code under control of the developers has been properly counted.[14] Having a vulnerability because they updated to a version that included unwanted or malicious

updates is way less likely. After all, developers with the best of intentions who unwittingly introduce vulnerabilities are far more common than those who intentionally insert vulnerable or unwanted functionality.

Fabio Massacci and Antonino Sabetta

If past is prologue, these events might also indicate a new factor of fragility of the overall software infrastructure. With protestware, there is a new reason for open source software components to become the single point of failure of several companies, and it has to do with political stances, personal beliefs and motivations, and even psychological and sociological factors: a risk that doesn't fare well on a company's 10-K form.

The coming of age of the open source world also brings the full realization of its key role in the economics of today's software industry. Many cornerstone packages used in thousands of commercial products rely on one-developer shows that receive no financial support from the corporations that benefit from them. The frustration that ensues risks pushing more open source developers to pull the plug on their projects.

So, the protests on too many free riders or the enemies of the country next door are just examples of the future likely failure of the governance of open source processes. The threat from individual developers making changes to their products to limit or degrade their functionality for users in certain geographic regions, or to include outright hostile functionality, is magnified when that product is widely relied upon.

Hamed Okhravi

What would be the impact of silently implanted, intentional bugs in a code base that may be used or deployed in various systems with various missions? Given the risk of spreading beyond the targeted region, how can one even begin to understand the secondary and tertiary impact of such bugs? Could they result in system crashes and possibly damages to safety-critical systems or even lives outside the targeted region? These questions are hard to answer because they get to the core of why cybersecurity is hard: reasoning about complex systems with many interdependencies is exceedingly difficult even in specific cases and intractable in the general case.

Many cornerstone packages used in thousands of commercial products rely on one-developer.

Fabio Massacci and Antonino Sabetta

While presumably the node-ipc modification was meant to go unnoticed (it was encoded in such a way as not to be immediately readable), it was not "professionally stealthy" given that it used an obfuscation method that is rather primitive, to say the least. Also, the geolocation service used is rather accurate but not infallible (the provider of the geolocation application programming interface used in the node-ipc case claims that, at the granularity of countries, it has a 99% accuracy),[15] so some end users might have also been impacted outside of Russia and Belarus. The characteristics of the changes suggest that the technical execution of this action as well its actual (as opposed to the intended) consequences were not really thought through, but they were the result of some sort of "impulse hacking."

Developers who rely on third-party components may wish to keep their dependencies up to date to ensure that they are taking advantage of recent security patches; yet when third-party developers introduce unwanted changes, those undermine trust in the supply chain and the virtues of regularly patching third-party components. We are back to the conundrum discussed in the previous article on SolarWinds.[16] How can we distinguish the "normal" (unintentional) security bugs introduced in a software update from the "intentional" security bugs?

Anderson Rocha

As new forms of protest embedded into software take place, so does the need of developing checking software and protocols that could employ intelligent techniques for detecting security flaws in addition to a wiki-like development pipeline as mentioned previously. Perhaps an area of research that could be supported in this regard would be artificial intelligence-based security flaw and protestware detection, both in terms of checking written code and real-time software testing.

Direct and Cascading Impacts

Hamed Okhravi

To understand how much intentional bugs can complicate the operation of the existing bug-finding tools, consider why bug finding is hard. The difficulty of finding normal (unintentional) bugs is traditionally attributed to the corner cases in programming languages that hinder an accurate analysis.[17] Now consider that most research has focused on unintentional bugs that are introduced as a

result of programmer error, which may occasionally face such corner cases. Intentional bugs introduced by protestware can make this analysis vastly more difficult and inaccurate by explicitly targeting the corner cases to further impede detection. Furthermore, with unintentional bugs, analysis has to be done once on a given code base; with intentional bugs, the analysis needs to be repeated N times, where N is the number of different regional versions of the code, to ensure that the specific versions of the code base are not implanted with vulnerabilities.

Dan Geer

Poisoning a common resource, be it an open source code base or a municipal water plant, is a consequence of interdependence. In any such case of interdependence, any solution must be presumed to have important side effects; for example, faultlessly geocoding the Internet by global treaty might make extending the Westphalian principle whereby a sovereign is culpable for attacks emanating from its territory into a matter of settled international law, yet the side effects might well favor authoritarians. Similarly, making a service provider culpable for its use of code regardless of whether it wrote it itself or incorporated it from an open source pool might well accelerate the balkanization of the Internet while boosting cyber-industrial consolidation. As others have implied, keeping honest people honest is a high goal but one that is feasible. Keeping dishonest people honest is far harder and may not be feasible. As the line between data and code blurs with the proliferation of machine learning, provenance likely becomes undecidable.

Eric Bodden

Data models that have been pretrained by third parties are a new type of common resource. Their increasingly widespread use poses challenges that have not been discussed in enough depth in the security community so far. Like software components, they too build an increasingly important supply chain, yet at the same time, the trust that we can put in these models is quite unclear. Who trains these models and to what extent? How may these people be influenced, and how may the resulting models be biased? What may be the impact of these biases? These are questions that we need to discuss, also from a geostrategic point of view.

Do We Need New Rules?

Jelena Mirkovic

One opportunity is to rethink open source and supply chain verification and governance systems. We are in need of the "Three Laws of Robotics," but for computer code.

Fabio Massacci and Antonino Sabetta

Discussions of changes to the source code driven by considerations that are political rather than technical have taken place in other projects as well. A proposal was made on 2 March 2022 in the OpenBLAS project repository[18] to drop the support for the Russian-produced Elbrus E2000 processor family, based on the fact that "the Elbrus processor is a so-called homegrown processor, with the primary use case of circumventing sanctions"[19] and that it is used by the Russian military and intelligence organizations. Along with the ingrained expectation that Internet standards should be based on technical merit,[20] members of the OpenBLAS community have pointed out that: (1) the Elbrus processor has a compatibility mode that allows it to run even code built for other architectures; (2) dropping Elbrus support would have no effect on any forks that might exist internally to the Russian military and intelligence organizations; and (3) there are indeed legitimate civilian uses of that architecture. Also, they stated their belief that the decision to include or drop support for a certain architecture should be driven purely by technical considerations and not political ones.

Protestware should be treated as a new security problem by the security community that needs to be addressed.

... and New Technical Solutions?

Jelena Mirkovic

A solution might also be on the technical side: can we develop methods that detect and interpret functional changes in code between its versions, especially when focusing on data exfiltration or data modification? Can we develop better code integration processes to detect and weed out unwanted additions? How can we develop code development systems that uphold some code functionality principles by themselves rejecting insider attacks?

Mohammad Mannan

Protestware should be treated as a new security problem by the security community that needs to be addressed. There are decent tools now for detecting

(exploitable) software bugs due to developers' mistakes, both at the source/machine-code levels, and some tools also exist to detect back doors and covert communication channels (with varying levels of success). There isn't anything on protestware, and the challenges introduced by protestware are also somewhat unique. For example, we need to consider unusual resource consumption, software/hardware corruption, and even the possibility of malicious or intentionally incorrect computation/logic bugs. Checking for context-based (for example, ISP, geolocation, or keyboard settings) exceptional execution paths could be a starting point for detection. However, developing an effective solution would take time and effort (albeit exciting for researchers!), but spending resources on this may be essential as open source projects are used by many government and business operations.

Checking for context-based (for example, ISP, geolocation, or keyboard settings) exceptional execution paths could be a starting point for detection.

We may debate whether it is appropriate or not—some developers will take an activist role and make malicious changes to their code in situations like the current war in Ukraine. Even if there are serious consequences to people outside the developers' target, this may continue to happen in the future—as developers are humans too, and humans aren't always rational. Working on these complex issues can help us prevent future supply chain attacks, which can bite friend and foe alike. ■

Acknowledgments

Part of this material is based upon work supported under U.S. Air Force Contract FA8702-15-D-0001 and the European Commission H2020 Program under Grant 952647 (AssureMOSS). This work is also supported by the São Paulo Research Foundation (FAPESP) under Grant DéjàVu 2017/12646-3.

Disclaimer

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the U.S. Air Force or the European Commission.

References

1. "Sabotage: Code added to popular NPM package wiped files in Russia and Belarus." Ars Technica. https://arstechnica.com/information-technology/2022/03/sabotage-code-added-to-popular-npm-package-wiped-files-in-russia-and-belarus/ (Accessed: Jun. 15, 2020).
2. A. Meneely and L. Williams, "Secure open source collaboration: An empirical study of Linus' law," in *Proc. 16th ACM Conf. Comput. Commun. Security*, Nov. 2009, pp. 453–462, doi: 10.1145/1653662.1653717.
3. "Styled-components." npm. https://www.npmjs.com/package/styled-components (Accessed: Jun. 15, 2020).
4. "New protestware found lurking in highly popular npm package." Checkmarx. https://checkmarx.com/blog/new-protestware-found-lurking-in-highly-popular-npm-package/ (Accessed: Jun. 15, 2020).
5. "Open source maintainer pulls the plug on npm packages colors and faker, now what?" Snyk. https://snyk.io/blog/open-source-npm-packages-colors-faker/ (Accessed: Jun. 15, 2020).
6. R. Stallman, "Viewpoint why 'open source' misses the point of free software," *Commun. ACM*, vol. 52, no. 6, pp. 31–33, 2009, doi: 10.1145/1516046.1516058.
7. S. Greenstein, "Free software without a free lunch or free beer," *IEEE Micro*, vol. 38, no. 5, pp. 94–96, 2018, doi: 10.1109/MM.2018.053631146.
8. "'Open Source' is broken." Christine. https://christine.website/blog/open-source-broken-2021-12-11 (Accessed: Jun. 15, 2020).
9. "Try this one weird trick Russian hackers hate." KrebsOnSecurity. https://krebsonsecurity.com/2021/05/try-this-one-weird-trick-russian-hackers-hate/ (Accessed: Jun. 15, 2020).
10. R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, 2011, doi: 10.1109/MSP.2011.67.
11. C. O'Rourke, "Is this the end for 'encro' phones?" *Comput. Fraud Secur.*, vol. 2020, no. 11, pp. 8–10, 2020, doi: 10.1016/S1361-3723(20)30118-4.
12. D. Gotterbarn, A. Bruckman, C. Flick, K. Miller, and M. J. Wolf, "ACM code of ethics: A guide for positive action," *Commun. ACM*, vol. 61, no. 1, pp. 121–128, 2017, doi: 10.1145/3173016.
13. M. S. Kirkpatrick, "Case studies," in *ACM Code of Ethics and Professional Conduct*, 2018.
14. I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vuln4Real: A methodology for counting actually vulnerable dependencies," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1592–1609, 2020, doi: 10.1109/TSE.2020.3025443.
15. "IP Geolocation FAQs." ipgeolocation. https://ipgeolocation.io/faq.html (Accessed: Jun. 15, 2020).

16. F. Massacci, T. Jaeger, and S. Peisert, "Solarwinds and the challenges of patching: Can we ever stop dancing with the devil," *IEEE Security Privacy*, vol. 19, no. 2, pp. 14–19, 2021, doi: 10.1109/MSEC.2021.3050433.
17. M. Miller "Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape." GitHub. https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%20-%20BlueHatIL%20-%20Trends%2C%20challenge%2C%20and%20shifts%20in%20software%20vulnerability%20mitigation.pdf (Accessed: Jun. 15, 2020).
18. "OpenBLAS." GitHub. https://github.com/xianyi/OpenBLAS (Accessed: Jun. 15, 2020).
19. "OpenBLAS." GitHub. https://github.com/xianyi/OpenBLAS/issues/3551 (Accessed: Jun. 15, 2020).
20. A. L. Russell, "Rough consensus and running code' and the Internet-OSI standards war," *IEEE Ann. Hist. Comput.*, vol. 28, no. 3, pp. 48–61, 2006, doi: 10.1109/MAHC.2006.42.

Fabio Massacci is a professor at the University of Trento, Trento, 38123, Italy, and Vrije Universiteit, Amsterdam, 1081 HV, The Netherlands. Massacci received a Ph.D. in computing from the University of Rome "La Sapienza." He received the IEEE Requirements Engineering Conference Ten Year Most Influential Paper Award on security in sociotechnical systems. He participates in the FIRST special interest group on the Common Vulnerability Scoring System and the European pilot CyberSec4Europe on the governance of cybersecurity. He coordinates the European AssureMOSS project. He is a Member of IEEE, the Association for Computing Machinery, and the Society for Risk Analysis. Contact him at fabio.massacci@ieee.org.

Antonino Sabetta is a senior researcher with SAP Security Research. His research interest is in the security of open source software, especially using machine learning to analyze source code. Sabetta received a Ph.D. in computer science and automation engineering from the University of Rome Tor Vergata, Italy. He is the technical leader of the European AssureMoss project.

Jelena Mirkovic is a project leader at the Information Sciences Institute of the University of Southern California, Los Angeles, California, 90292, USA. Mirkovic received a Ph.D. in computer science from the University of California, Los Angeles. Contact her at mirkovic@isi.edu.

Toby Murray is a researcher in the Software Systems Research Group at NICTA and a conjoint lecturer in the School of Computer Science and Engineering at the University of New South Wales. Contact him at toby.murray@nicta.com.au.

Hamed Okhravi is a senior staff member at the Massachusetts Institute of Technology (MIT) Lincoln Laboratory, Lexington, Massachusetts, 02421, USA. Okhravi received a Ph.D. in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He is also the recipient of two Best Paper Awards, two R&D 100 Awards, MIT Lincoln Laboratory's Best Invention and Early Career Technical Achievement Awards, and NSA's Best Scientific Cybersecurity Paper Award. He is a member of DARPA ISAT and a Senior Member of IEEE. Contact him at hamed.okhravi@ll.mit.edu

Mohammad Mannan is an associate professor at the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Quebec, H3G 1M8, Canada. Mannan received a Ph.D. in computer science in the area of Internet authentication and usable security from Carleton University. Contact him at m.mannan@concordia.ca.

Anderson Rocha has been an associate professor at the Institute of Computing, Unicamp, since 2009. Since 2010, he has been a coordinator at the Artificial Intelligence (Recod.ai) Laboratory, Unicamp. His research interests include artificial intelligence, machine learning, and digital forensics. Rocha received a Ph.D. in computer science from the University of Campinas (Unicamp), Brazil. He is a Google and Microsoft Faculty Fellow. He was the chair of the IEEE Information Forensics and Security Technical Committee from 2019 to 2020. He is a Senior Member of IEEE.

Eric Bodden is a full professor for secure software engineering at the Heinz Nixdorf Institute of Paderborn University, Germany. He is also the director for software engineering at the Fraunhofer Institute for Engineering Mechatronic Systems. He has been recognized several times for his research on program analysis and software security, most notably with the German IT-Security Price and the Heinz Maier-Leibnitz Price of the German Research Foundation, as well as with several distinguished paper and distinguished reviewer awards.

Daniel E. Geer, Jr. is a Senior Fellow at In-Q-Tel, USA. His collected works are available at http://geer.tinho.net/pubs. Contact him at dan@geer.org.