Cybersecurity Risks Unique to Open Source and What Communities Are Doing to Reduce Them

Matthew L. Levy, U.S. Naval Information Warfare Center

This article delineates four categories of risk unique to open source projects and communities. We also discuss the communities actively seeking to mitigate these risk areas and suggest areas to elevate cybersecurity risk in open source.

Digital Object Identifier 10.1109/MC.2023.3262903
Date of current version: 31 May 2023

Consumers of open source consider it to be more secure than hybrid or closed source software.[10] Security awareness in the design phase can make it more secure. Transparency and openness in code development can mitigate outside concerns about security. The rigorous testing processes in large open source communities can reduce bugs and increase responsiveness in patching vulnerabilities. And of course, there is Linus's Law, "given enough eyeballs, all bugs are shallow," effectively promulgating the idea that open source contains fewer bugs, and fewer bugs mean fewer potential exploits.

INTRODUCTION

Indeed, a well-run open source software project can signal all of the aforementioned advantages and

EDITOR DIRK RIEHLE
Friedrich Alexander-University of Erlangen Nürnberg;
dirk.riehle@fau.de

more. But, with any decision, there is risk, and thus, we offer this article to bring some of the risks unique to open source to the fore. Accordingly, we delineate and discuss four major risk areas as consideration for open source usage. Following this, we highlight the technologists and researchers who are evolving the knowledge and tools to mitigate risk.

FROM THE EDITOR

Welcome back to Open Source Expanded! Security is on the forefront of many folks' mind, and open source is in the middle of it. Does transparency or obscurity make code more secure? Or is it the processes? What does the complexity of the dependency graph and the software supply chain mean? How to deal with adversaries and not just unintended bugs? In this column's article we will get a first idea of the complexity of the situation, thanks to Matthew ("Matt") Levy of the U.S. Navy. As always stay happy and healthy, and keep on hacking, securely. —*Dirk Riehle*

INFRASTRUCTURE RISK

Threats to open source infrastructure come mainly from attacks on the open source supply chain. Open source supply chains consist of a product's dependency graph—the set of components a program depends on and the way these components rely on each other—and the chain of suppliers that provide and pass on a component to their clients.[13] Open source dependencies are vast but often rooted in a small number of packages widely used throughout thousands of open source communities. We conceptualize open source software dependency risk along three dimensions: dependencies, dependents, and package managers (Figure 1).

Dependencies are packages a project depends on, making it susceptible to upstream exploitation. Dependents are packages depending on that project, making the dependents susceptible to upstream exploitation. Package managers are online services that manage massive collections of packages and automate package installation, upgrading, configuring, and removal.

To understand the vastness of open source infrastructure, we use the example of Express.js, a web application framework for building Node.js RESTful application programming interfaces (APIs). At the time of this writing, Express.js has 31 direct dependencies, 25 indirect dependencies, and many more when one considers node.js runtime calls—also open source projects. Express.js also has 551,575 direct dependents and 58,521 indirect dependents (Figure 2).[7]

Delving deeper into the lineage of Express.js, we see it depends on a component called body-parser.js, which has 29 dependencies and more than 700,000 dependents. Further, body-parser depends on a package called inherits.js, which has more than 700,000 dependents. Thus, if inherits.js get infected, it would affect millions of dependents. Fortunately, from a risk perspective, core packages are typically smaller in code size as one moves upstream, making malicious commits easier to catch. However, the story of the corruption of core components like colors.js, a seemingly simple project that provides coloration for node.js consoles, provides an example of potential downstream risk. When colors.js was exploited, it had more than 20 million weekly downloads and had Winston, the popular logging package, as a dependent.

Package management services also pose distinct risks in open source. On one hand, package management services are indispensable for project consistency. On the other, an attack on a single project can affect millions of package management users. Package management also execute preinstall and postinstall scripts that can run remote code. In a recent attempt to hack into PayPal, security researchers demonstrated that by searching for private repository names in public

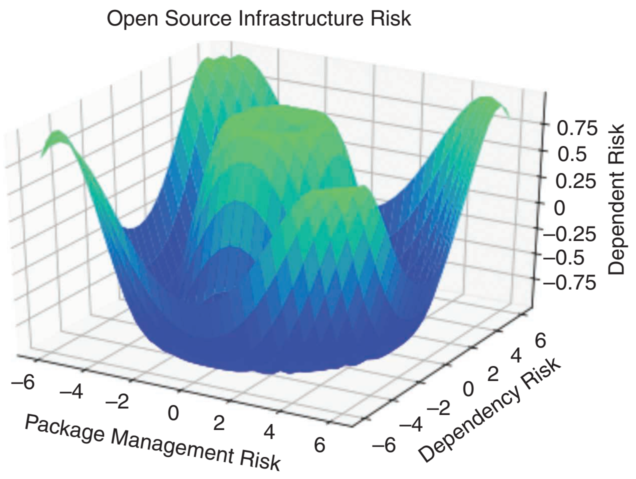


FIGURE 1. Dimensions of infrastructure risk.

projects by parsing the package.json configuration file, they could create malicious packages on node package manager (NPM) with the same name. Because node.js compilation searches public projects first when collecting dependencies, the project then downloaded the public (malicious) package instead of the private one. Consequently, using Domain Name System exfiltration techniques in a preinstall script, the newly installed package then phoned home from each computer. The researchers began by infiltrating PayPal but quickly discovered they could use the same techniques against Apple and Microsoft servers, and dozens of other large organizations.

Indeed, recent articles have illustrated how difficult it has been for package managers to verify packages on their platform, allowing for exploitation tactics like typosquatting and brandjacking, which result in dependency confusion and dependency injection. Indeed, a recent report revealed that PyPI, the most widely used Python package management service, removed more than 3653 typosquatted package names.[5]

PROCESS RISK

Like any other type of software, the maturity of open source software processes is gradual, sometimes taking years to reach the latter stages of maturity where they are stable and flexible, and the focus can be on responding to change. Conversely, software projects lacking contributors, resources, and tools to mature software processes are likely to be less reactive in their response to improvements or patches and experience difficulties that may impact code quality—to rapidly deploy improvements and quickly release patches that extricate vulnerabilities and software bugs.

Beyond top-tier projects, many open source projects have small numbers of active contributors. Indeed, many projects, some with thousands of downloads daily, are surprisingly small and lack the resources to develop mature software processes. A 2015 study of popular GitHub projects across language ecosystems found that nearly two-thirds of open source projects have only one or two maintainers, and only three projects in the study had more than 50 contributors.[1] Indeed, from a process-maturity perspective, many open source projects may lack the resources to adequately respond to its users, presenting insidious risk and opportunity for those seeking to exploit immature release processes, especially given that these processes are open for all to see.

Prior process deficiencies in the event-stream community provide us with a prescient example. Event-stream, a library for working with streaming data, has more than 1900 dependents and, at the time of this writing, has more than 3.3 million weekly downloads. In 2018, a single user compromised event-stream by injecting a malicious package that targeted the developers of the CoPay bitcoin wallet application. When Copay developers ran their build scripts, it modified the code before being bundled into the application. Then upon deployment,

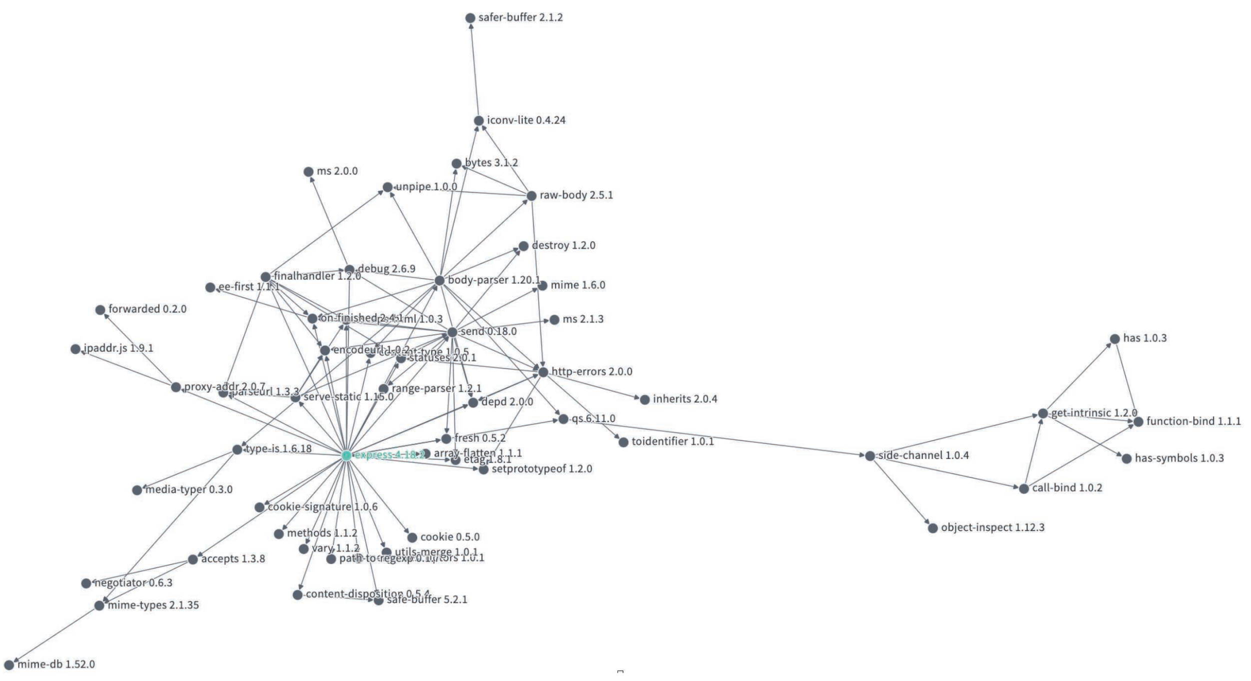


FIGURE 2. Dependencies for Express.js (version 4.18.2).

the code harvested account details and private keys from accounts having a balance of more than 100 Bitcoin or 1000 Bitcoin Cash.[11] The risk from immature processes is illustrated by how a single nefarious user, Right9ctrl, was granted full commit access by event-stream's principal maintainer, Dominic Tarr, by simply asking for it. There was no identity verification, no previous history of committing to the project, and no automated process that checked dependencies or dependents, upstream or downstream, for malicious code. In contrast, with mature governance processes and corresponding tooling around providing users with commit access or a release management process that scans for commits by new members, malicious code, or malicious design patterns, the event-stream hack could have been avoided. Reinforcing this, in a recent study, nearly half of all open source communities were missing either an openly described release management process or security auditing procedures.[8]

While many major open source communities now have the advantage of corporate sponsorships and corporate employees paid for community participation, many widely used open source projects still do not. Moreover, in these unsubsidized communities, consumers have tended to experience bugs after each release, making programs less useful in the tenuous days to weeks after an update. Indeed, without mature processes, research on open source communities has found a statistically significant association between prerelease bugs and postrelease vulnerabilities.[4]

METADATA RISK

Open source metadata includes commit data, bug lists, merge requests, discussions, continuous integration configurations, documentation, and active contributor descriptions. Open availability of metadata benefits those seeking to contribute to a community and those seeking to consume its products. But security research is also sounding the alarm on open source metadata.[9] For example, commit git histories can be rewritten to distort trust and gain access. While such a feature enables a contributor to have fine-grained control over the information she submits along with a code commit, user authentication is not enabled by default which opens the door to forging information about collaborators. A git commit creates a record consisting of a unique identifier in the form of a hash created from the specific changes, the date and time of the changes, and who made them. By design, a committer can manually change this record with commands like *set git*. With this ability, a nefarious actor can change commit dates and subsequently push changes to git for dates and times of their choosing that can even predate the creation of a user account or a repository since git lacks the mechanisms that check these data.

Commands like *set git* for these purposes may seem benign or far-fetched, but it allows a nefarious actor to create their own record of legitimacy—such as filling in one's own activity graph. Since the activity graph displays activity indiscriminately on public and private repositories, it is difficult to discredit fake commits, making this deception technique difficult to detect (Figure 3).

Similarly, a nefarious committer could also spoof their identity and

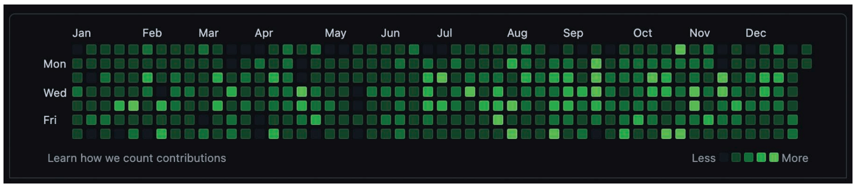
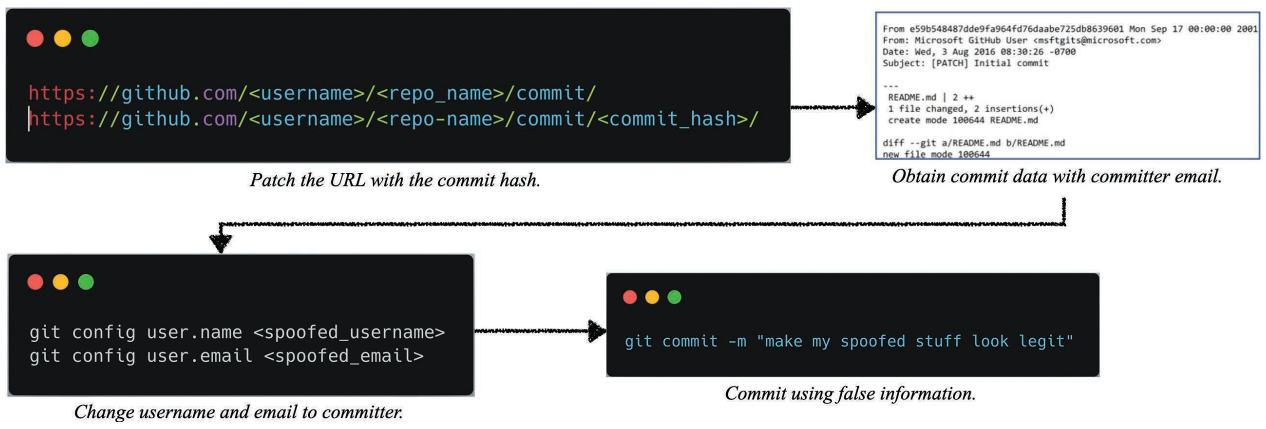


FIGURE 3. 2012 commit history of T.J Holowaychuk, Creator of the Express.js Framework.



```
https://github.com/<username>/<repo_name>/commit/
https://github.com/<username>/<repo-name>/commit/<commit_hash>/
```

Patch the URL with the commit hash.

```
From e59b548487dde9fa964fd76daabe725db8639601 Mon Sep 17 00:00:00 2001
From: Microsoft GitHub User <msftgits@microsoft.com>
Date: Wed, 3 Aug 2016 08:30:26 -0700
Subject: [PATCH] Initial commit

---
 README.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 README.md

diff --git a/README.md b/README.md
new file mode 100644
```

Obtain commit data with committer email.

```
git config user.name <spoofed_username>
git config user.email <spoofed_email>
```

Change username and email to committer.

```
git commit -m "make my spoofed stuff look legit"
```

Commit using false information.

FIGURE 4. How to change commit information.

attribute commits to other users by pushing commits on their behalf. Indeed, default project settings on popular code repository hosting sites allow this to happen. Given a commit hash, a nefarious actor can obtain the e-mail address of the project committer by patching the URL string of the commit, revealing the e-mail address from the committer username. Subsequently, the nefarious committer can change the e-mail address with *git config* and commit code changes to git using the manipulated information (Figure 4).

What makes the aforementioned example even more alarming is that the user whose e-mail address used in the attack vector will likely never know someone is using it. What is further insidious about it is how undermines legitimacy and trustworthiness to achieve power. Indeed, when observing the myriad ways state-sponsored hacking organizations have sought to undermine systemic trust as one of their main objectives, this example is notable, as it is now quite clear that state-sponsored hacking organizations are not only tactically attempting to alter specific outcomes (like an election) but seeking to sow the seeds of division for future exploitation.[3]

NARRATIVE RISK

Underpinning our examples in the preceding sections are the narratives that nefarious actors exploit. Concerning cybersecurity, narratives have been given increasing attention as a growing body of researchers point to the persuasive and identity-building capacities of how narratives can be weaponized to shape understanding and mobilize individuals. Narratives are the speech-acts that constitute spoken or written accounts of interconnected events and allow for the interpretation and analysis of human experience, meaning, knowledge, social action, human agency, and the complexity of social elements of human life. The creation of narratives is powerful in that they are open to interpretation, inviting others to attribute meaning, and can thereby make fringe ideas less antagonistic or threatening and engage recruits. Thus, if those seeking to craft narratives in open source communities adhere to the dominant cultural codes of that community, they stand a greater chance of being successful at shaping strategic decision-making and making others in the group reluctant to challenge it as they seek to secure a sympathetic hearing for positions unlikely to gain such a hearing otherwise.[12]

Because node.js compilation searches public projects first when collecting dependencies, the project then downloaded the public (malicious) package instead of the private one.

Dominating narratives in open source have to do with extraordinary levels of implicit trust. For decades, high levels of trust had little impact on community legitimacy since, even in hacker communities, tampering with open source was seen as taboo. However, as nation-state actors have become the driving force behind advanced cyberattacks, high levels of trust represent targets ripe for exploitation. Underpinning the attack vectors mentioned in previous sections are exploitations of the trust narrative. For example, the attack on the event-stream community was an attack on the trust narrative. The lead developer, Dominic Tarr, implicitly trusted right9ctrl with contributor access, and given the culture of trust in open source community identity, they exploited it to steal Bitcoin wallets. The attack vector described that takes advantage of the purposely open structure of git, where nefarious actors can modify usernames and e-mail addresses on a per-commit basis, also illustrate the exploitation of high levels of trust in community identity and commit history as a form of discourse. There are also high levels of trust in the integrity of package management services that coincidentally are now the principal vector for open source supply chain attacks. And there are numerous other examples: One being the controversial case of the hypocrite commits, where University of Minnesota researchers pushed bogus commits after establishing a trusted identity on the Linux project, or how the trust narrative was undermined when commit information was altered and two malicious commits were added to the PHP-src repository in the creator's name, Rasmus Lerdorf, where the code planted a backdoor for obtaining remote code execution for any website running this hijacked version of PHP. These are scary propositions. One is at the heart of the open source movement. The other is a project that runs on roughly 79% of websites on the Internet.

DISCUSSION

Not until recently have open source projects, participants, researchers, and teams begun to actuate awareness of these risks. As recently reported in this journal, roughly five years ago, university researchers created the Community Health Analytics in Open Source Software (CHAOSS) project to provide a compendium of health metrics that go far beyond those based on user and project commit history. CHAOSS has evolved significantly and now delineates five categories of health metrics: evolution, common, diversity, equity, and inclusion (DEI), value, and risk. In addition, adjacent projects also develop tooling for organizations to evaluate projects using CHAOSS metrics.[6] Given our aforementioned examples, future CHAOSS metrics could also include cybersecurity risk. For example, metrics delineating the security controls (for example, vigilant mode, commit signature verification) used in an open source project, metrics on supply chain vulnerabilities that include native libraries, and metrics that investigate discourse and narrative emotives in these communities.

Indeed, CHAOSS has begun to take similar strides in adopting best practices from the Open Source Security Foundation (OpenSSF). But specificity pertinent to cybersecurity risk is needed. At present, CHAOSS contains metric specifications to evaluate whether OpenSSF best practices are used, but needed are metrics on the details of how they are employed—such as whether privileged developers use multifactor authentication (MFA) tokens, whether the code contains secrets such as hashes or other resource-identifying information, or if the project has active efforts to identify and disclose vulnerabilities.

DARPA's SocialCyber project is another area of security research that could be used to elevate CHAOSS. SocialCyber is developing analytics that can detect and counteract cybersocial operations, such as those that may target open source developer communities through combinations of submissions of flawed code or designs, social media campaigns against OSS developers and maintainers critical of the flaws, as well as via misleading bug reports, obfuscating technical discussions, and social capture of functional authority on OSS projects.[2]

In sum, we make the clarion call for the larger network of open source researchers and technologists to elevate CHAOSS and other communities that elucidate and mitigate open source risk.

We hope this article illuminates open source risk without dissuading open source usage. We further hope this article serves to persuade open source communities to adopt additional security controls. And we hope critical communities like CHAOSS, OpenSSF, and SocialCyber continue to receive increased attention. To conclude, and paraphrase Louis Brandeis, we are attempting to use sunlight as the best disinfectant to elucidate the security risks in open source. Accordingly, the point of this article is not to say that open source is any more or less risky than using other forms of proprietary or hybrid software. It is to illustrate that open source, and its underlying social structures, carry their own *unique* risks. C

REFERENCES

1. G. Avelino, M. T. Valente, and A. Hora, "What is the truck factor of popular GitHub applications? A first assessment," *PeerJ PrePrints*, vol. 3, Jan. 2017, Art. no. e1233v2, doi: 10.7287/peerj.preprints.1233v3.
2. S. Bratus, "Hybrid AI to protect integrity of open-source code (SocialCyber)," Defense Advanced Research Projects Agency, Arlington, VA, USA, 2020. [Online]. Available: https://www.darpa.mil/program/hybrid-ai-to-protect-integrity-of-open-source-code
3. P. Burkhart and T. McCourt, *Why Hackers Win: Power and Disruption in the Network Society*. Berkeley, CA, USA: Univ. California Press, 2019, pp. 31–34.
4. F. Camilo, A. Meneely, and M. Nagappan, "Do bugs foreshadow vulnerabilities? A study of the chromium project," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, 2015, pp. 269–279, doi: 10.1109/MSR.2015.32.
5. T. Claburn. "About half of Python libraries in PyPI may have security issues, boffins say." The Register. Accessed: Nov. 10, 2021. [Online]. Available: https://www.theregister.com/2021/07/28/python_pypi_security/
6. S. Goggins, M. Germonprez, and K. Lumbard, "Making open source project health transparent," *Computer*, vol. 54, no. 8, pp. 104–111, Aug. 2021, doi: 10.1109/MC.2021.3084015.
7. Google, Inc. "Open-source insights." Open-Source Insights. Accessed: Dec. 8, 2022. [Online]. Available: https://deps.dev
8. M. Gresham. "Addressing cybersecurity challenges in open-source software." Snyk. Accessed: Feb. 18, 2023. [Online]. Available: https://snyk.io/blog/addressing-cybersecurity-challenges-in-open-source-software/
9. M. Kapko. "Fake GitHub commits can trick developers into using malicious code." Cybersecurity Dive. Accessed: May 8, 2022. [Online]. Available: https://www.cybersecuritydive.com/news/github-commits-malicious-code/627466/
10. L. Morgan and P. Finnegan, "How perceptions of open source software influence adoption: An exploratory study," in *Proc. 15th Eur. Conf. Inf. Syst. (ECIS)*, 2007, pp. 973–984.
11. "Details about the event-stream incident." npm Blog. Accessed: Jul. 3, 2022. [Online]. Available: https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident
12. F. Polletta, *It Was Like a Fever: Storytelling in Protest and Politics*. Chicago, IL, USA: Univ. Chicago Press, 2009.
13. D. Riehle. "Dependency graph vs. software supply chain." Bayave. Accessed: Mar. 16, 2023. [Online]. Available: https://bayave.com/2023/02/14/dependency-graph-vs-software-supply-chain/

MATTHEW L. LEVY is a research scientist at the U.S. Naval Information Warfare Center, San Diego, CA 92152 USA. Contact him at matthew.l.levy.civ@us.navy.mil.