The Exploitation of the Open-Source Philosophy

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Eric Knocklein

Spring 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Joshua Earle, Department of Engineering and Society

1. Introduction

Complex digital systems are an invaluable part of modern life. This thesis explores the vulnerabilities of one crucial aspect of these systems: Open Source Software (OSS). OSS is "software developed and maintained via open collaboration, and made available, typically at no cost, for anyone to use, examine, alter and redistribute however they like" (IBM). Throughout this paper it will be made clear how crucial OSS and the open source ecosystem is for the tech ecosystem in both the production of proprietary software – that is software that is not open source – and open source software. Given the importance of OSS, it is crucial that those who work within that community are treated fairly.

A small part of this paper is concerned with the security vulnerabilities of the technology – how companies utilize OSS without maintaining it or properly researching it, thereby putting their software and, in turn, the users of their software at risk. A more substantial part of the paper will relate to the ecosystem of OSS developers and the companies that use OSS. It will question whether the interaction between these two groups in its current state is ethical.

A. The Open Source Philosophy

I start by analyzing the driving force behind the development of OSS. This is the so-called "Open Source Philosophy," which informs and builds the environment in which OSS is developed. Some of its foundations are laid out in *The GNU Manifesto*, a document written by Richard Stallman in 1985. This philosophy stands in stark opposition to traditional methods of software development and distribution. At its core, it is a unifying philosophy between developer and user, a coalition meant to stand against traditional software vendors, who "want to divide the users and conquer them" (Stallman, 1985). Stallman asserts that "the fundamental act of

friendship among programmers is the sharing of programs" (Stallman, 1985). These software should be shared in such a way to promote collaboration without fear of retaliation from the developer.

The first three and most important of the "Debian Free Software Guidelines" as described in the *Debian Social Contract*, which details the core concepts of the open source philosophy, are the "Free Redistribution," the "Source Code," and the "Derived Works" guidelines (*Debian Social Contract*, n.d.). Under these guidelines, users of OSS can edit their own software – since it is distributed with both the compiled and uncompiled versions of the source code – and distribute these modifications.

A large group of open source developers built a community around these and other similar guidelines. This community is crucial for the development of proprietary software. I explore the dynamics in the interplay between these two communities – only one bound and self regulated by the guidelines above – and how this can lead to vulnerabilities and exploitation.

2. Methods/Results

Before I make any judgments about any potential vulnerabilities and acts of exploitation, I perform a literature review. In this section I review general topics relating to OSS before identifying the stakeholders in the issues presented. Primarily, I examine the worker perspective on these issues, as well as the security vulnerabilities that are commonplace in the use of OSS within proprietary software. I discuss those who are affected by OSS, their desires, the impacts of these desires, and the system created by the confluence of these desires. After this core is established, I identify some ethical and social frameworks through which to analyze the issues, particularly, the Motivation-Practice Framework and Virtue Ethics.

A. Literature Review

i. Identification of Stakeholders and Their Desires

Since I am examining the interplay between two communities, I break the stakeholders into two parties. I make a distinction between those who consume or produce OSS under OSS guidelines and those who consume and produce OSS without being part of the OSS community, without being beholden to the OSS guidelines.

When describing the constituents of traditional development organizations, one faces little difficulty, for their developers are clearly listed and documented. The same cannot be said about OSS development. In order to examine the motivations of OSS developers and the contexts in which they operate, I must, however, make sense of this decentralized system. Khan et al show that it is a decentralized system. The study conducted by Khan concluded that "the community structure of the developer network is decentralized" (Khan et al., 2021, 351). Further, they were unable to show that there was an influential core of developers that formed the backbone of the community. Not only are these OSS developers decentralized on a global scale, they also do not usually form large or rigid teams, preferring rather to primarily work independently or in a small group.

This decentralized structure contrasts heavily against the structure of more traditional software development. Tech companies are clearly structured in a hierarchy, at least in global structure, though individual teams may be constructed more loosely.

In *Carrots and Rainbows*, the authors identify a key motivation that brings developers into OSS and keeps them there. This boils down to the standards of the OSS community and the internal and external good that it produces. The OSS community creates a social practice that is

self-sustaining in that it encourages developers to maintain it. Considering the code of conduct of the open source community that was discussed earlier in this paper, this makes sense. Development of OSS is not done primarily for monetary gain but due to a social pressure and a communal desire to upkeep the standards of the community. As Stallman would say, it is an expression of goodwill and friendship (Stallman, 1990).

In contrast, the motivation of traditional software vendors is to maintain their company and maximize the profits for its shareholders. A company that is not profitable is not viable. As such, the motivation of traditional software vendors is to minimize their expenses and maximize their revenue. While companies can act against this motivation, the forces of the system and the community in which they exist encourage them to follow it.

Lastly, I consider the users, who generally want software that is fairly priced and of high quality. These desires are quite simple, which makes the user group more easy to understand than the other two groups. A desire for high quality software encapsulates one for secure software. While the user may be a simple part of this complex system, they play an important role. In the case of proprietary software, the user is the customer and provides the software company with the resources to do their development. This changes somewhat in the OSS community, where the users are also often the developers. However, unlike with proprietary software, users are not crucial to the development or survival of OSS, since the desire for money or recognition is not the only motivation behind the development of software.

ii. Identification of Interactions

Interactions between the OSS community and proprietary organizations are most important in this analysis, so I will not be discussing interactions with users much in this section.

The OSS community is co-opted by proprietary organizations in two important ways: through the integration of OSS code and through integration of OSS developers. If any exploitation exists between the OSS community and proprietary software vendors, then it will be in one of these two interactions. According to Hauge et al, the former interaction can be broken down into a few distinct pathways.

Organizations could "integrate OSS components" (Hauge et al., 2010, 41). A component, in this context, is an almost self-contained piece of software able to or meant to be used in a larger software. Hauge and coauthors make clear the extent of this interaction. One study showed that 33% of companies it reviewed provide products or services that are based on OSS. "Moreover, 48% of 62 software companies use OSS in their business, and in a sample of 569 software companies, 46.8% integrate OSS in their software systems" (Hauge et al., 2010, 22). Choosing which components to integrate is an ongoing problem for proprietary software vendors. While methods exist to ease the integration process, "these methods and techniques are seldom applied in practice" (Merilinna & Matinlassi, 2006, 7). Instead, software vendors rely on "experience and rules of thumb" (Merilinna & Matinlassi, 2006, 1).

Hauge et. al. also show how these organizations often modify the OSS that they use, which can be partly attributed to a poor choice in component choice. Due to the nature of OSS, this is allowed and even encouraged, though vendors must be cautious to not breach their license. Depending on the method chosen to handle these modifications, organizations can contribute to the OSS community.

In closer collaboration with the broader OSS community, organizations could also contribute to OSS products which they do not control. They could also develop directly in the OSS community by creating OSS products. While Hauge and coauthors point out that this is

possible, most of their discussion focuses on how little companies contribute to the OSS community. The most common form of contribution is bug reporting, with more substantial involvement being rare. Releasing their own OSS products is another way in which companies can interact with the community. Hauge et. al. examine this possibility as well, claiming that it is primarily a marketing and distribution strategy rather than a development strategy.

iii. System Perspective

As self-contained as these interactions may seem, they are taken within a larger system. *Chokepoint Capitalism* by C. Doctorow and R. Giblin describes the role of systems in creative work, specifically how chokepoint capitalism – as they name it – interacts with creative works and workers. They argue that the current system's purpose is to consolidate culture and the creative works under that culture. After all, "the purpose of a system is what it does," (Doctorow and Giblin, 2022, Chapter 19) and the book describes many instances where this consolidation is exactly what the system has done. This connects well with the motivations I outlined earlier in this section because the system is a consolidation of motivations, and analysis of the system allows for a more clear understanding of the motivations that govern it and which it informs.

iv. Security Perspective

Much of the literature into OSS is done on how open-source software influences the security of software. Primary amongst this literature is the aforementioned *2022 Open Source Security and Risk Analysis (OSSRA) Report*. This report is a wide-ranging review of the state of the security of open source code and components. It emphasizes the need for continual maintenance of codebases, particularly in how they are affected by open source code and the

width of influence of open source code. According to these reports, 99% of codebases contain some form of OSS component and “80% of the codebases were composed of open source” (Synopsis, 2022, Page 12). Additionally, “Twenty-three percent of open source projects have only one developer contributing the bulk of code. Ninety-four percent of the projects have fewer than 10 developers accounting for more than 90% of the lines of code” (Synopsis, 2022, Page 20). These statistics reveal the scope of the technology and thereby the necessity of reports such as the others reviewed in this paper. They also emphasize the fact that many OSS projects are developed individually or in small groups instead of being developed by large companies.

While having OSS components or utilizing OSS is not inherently a risk to security, the report continues to provide troubling statistics for the state of OSS integration. For instance, 85 percent of codebases audited by the report contained open source elements that were more than four years out of date, and 88 percent used open source components which had an updated version available. It is not difficult to see why this could be a security issue. If a hacker or penetration tester discovers and reports a vulnerability in one of these components – which are not being updated regularly – it could linger in the software indefinitely even if a new version of the component has been released that addresses the issue. Hackers could then use the patch notes – the update information – of the component to examine how old components are vulnerable. Only by updating frequently can developers stay ahead of vulnerabilities.

The last portion of security that I see as a concern is independence and stability. In 2016, a developer by the name of Azer Koçulu almost broke the internet by taking an open source component off of a prominent library for such components called npm. This alone would not be a problem if it was not the case that a large number of other software and components were dependent on this component to function properly. A large chain of components all failed in

unison due to one developer revoking access to his code. “Loading your own app might require a certain set of packages from npm, but those packages may require their own sets of packages, and so on” (Collins, 2016). In this way, relying on OSS components could mean building on a shaky foundation.

B. Frameworks

To guide the judgements gleaned from this information, I discuss OSS in relation to the Motivation-Practice Framework proposed by von Krogh, Haefliger, Spaeth, and Wallin. Within this framework, actions and motivations are considered in how they interact with social practice, institutions, and internal and external good. Krogh and coauthors created the framework to formulate and answer three questions about OSS that will be summarized here.

First, Krogh and coauthors conclude that OSS developers interact with the social practice of OSS by producing internal good to that social practice (i.e. they enhance the social practice through their work). In turn, the enhancement of the social practice through the development of this good alters the standards of the work done for that social practice. This contrasts with work in regular institutions, in which more of a one-way relationship from worker to institution exists. In some ways, it seems more apt to think of OSS development not as individuals but as the social practice as a whole being composed of individuals.

The second conjecture of Krogh et. al. under the framework deals with the method by which OSS developers change institutions. As before, this is not a one-way relationship. Krogh and coauthors conjecture that “OSS developers change institutions when and where these institutions no longer protect sufficiently the standards of excellence of the social practice” (von Krogh, et al, 2012, Page 667).

Lastly, Krogh and coauthors the functioning of the social practice and how it is sustained by developers. They conjecture that social practice and its standards create the motivation for developers to uphold it. These motivations foster a community where actions have true and visible consequences upon the standards and course of the community as a whole. In such a community, actions are meaningful, even mundane ones like bug-fixing. Work with meaning is more appealing than work without, which encourages people to work in this field.

There are also the future prospects of the developers to consider. Many companies look for developers who have worked within OSS communities. In order to seem appealing to these companies, some developers might accept a low paying or volunteer role within OSS in hopes of landing a job at a large company. In this way, the company is benefiting both from the free software developed by the OSS community but also the training, the standards, and the culture of the OSS community. Now that the motivations and nature of the OSS community are more understood – or are able to be better understood through the use of the framework – it can be critically examined.

This examination will also focus on the Virtue Ethics of the OSS community and those of the companies who use OSS or hire OSS developers. Where motivation-practice theory considers the motivation of individual actors, virtue ethics examines their actions. In this analysis, we must ask whether the use of OSS software corresponds to virtuous human ideals such as fairness and justice.

3. Analysis

A. Impacts of Motivation

The differing motives between the OSS community and the more traditional software vendors creates possibilities for the exploitation of the OSS community. OSS is community focused, whereas traditional software is – for the most part – profit focused. Of course, traditional companies still have the capacity to act in a community-focused manner. Indeed, many companies set up their own OSS projects as outlined above. However, the motivation is only aligned with this pursuit if the pursuit promises profit. In contrast, many OSS projects are created without the need or the promise of any type of profit. The motivation is to do good by and for the community.

To summarize, proprietary software is defined by a closed motivation while OSS is defined by open motivation. Proprietary software benefits generally from taking and keeping, while OSS benefits from giving and taking in equal parts. This is the core imbalance in the interaction between these two systems: proprietary software takes what OSS gives and doesn't give much, if anything, back. It could be seen as an extraction of talent, a privatization of a public space.

B. Hiring and OSS

Having developers first cut their teeth in OSS is extremely beneficial for traditional software vendors. Working in OSS allows developers to show their prowess to potential recruiters, thereby decreasing the amount of time and money that has to be spent in the hiring and training process. This reduces the costs of hiring, aligning this sort of recruitment with the goals of traditional software vendors. If the precedent is set that OSS developers are hired much more frequently than developers who do not have OSS experience, then developers could see

OSS development as almost mandatory. While more OSS developers sounds like a good thing for both communities, that may not be the case.

What this could implement is a quasi unpaid internship position for developers. One could argue that this hypothetical is inherently different from a regular unpaid internship because it does not provide direct benefit to the proprietary software vendor. However, I have shown that OSS development eventually benefits proprietary vendors, who extensively use OSS in their software or use the standards created by OSS developers in their systems and software. Such a system would drive developers into OSS by monetary instead of moral or philosophical pressure, thereby degrading the very notion of the Open Source Community. Thus the motivation of proprietary software vendors is detrimental to the workers of both proprietary software and OSS.

While something as extreme as described above is not likely, as the balance between proprietary and open software vendors seems quite balanced and mutual at the moment, the inherent power imbalance in the interactions between these communities potentially makes this status quo quite unstable.

C. Power Imbalance

Even under the current status quo, though, some of these problems persist, and they all stem from this aforementioned power imbalance. The differences in motivation force the two communities to act in different ways. Because of the lack of strong communal guidelines, proprietary software companies are more free to act in their own interests. More crucially, proprietary software companies are usually composed of much larger teams, giving them a much greater ability to pursue those interests.

The broadest reaching of the problems created by this imbalance is the stagnation or limitation proprietary software places on education and the improvement of software more generally. OSS, being built on sharing and teaching one another, is much more conducive to learning and improving itself, and doing so aligns with its goals. Proprietary software development has no such goals. By siphoning OSS developers, proprietary software is also siphoning innovation from the public and thereby privatizing a common good.

One of the most egregious acts of exploitation in the same general vein is the blatant disregard for Open Source licenses shown by some proprietary software companies. *OSSRA* found that 53% of the codebases they audited contained license conflits. Since much of OSS is developed by small teams or even by individuals, who do not have the means to defend themselves against theft – or, indeed, even to be aware that the theft took place – OSS is an easy target for theft of intellectual property. Though OSS is generally free to use and modify, using it without a proper license is intellectual property theft. This does not mean that the practice is widespread amongst proprietary software but simply that the motivation exists to perpetuate these actions, which degrade the values of the open source community. In Garrett Hardin's famous thought experiment of the "tragedy of the commons," the open source community is not a grazer upon the commons but the commons themselves (Hardin, 1968). It is a crucial resource for all software, but without protection, the motivations of those who use it lead to an inevitable outcome.

D. Software Users

That is to say nothing of the actual users of the software, by far the largest section of shareholders in this discussion. I analyze two crucial effects of this interaction between OSS and

proprietary software. The first is that the interaction between proprietary and open software communities stifles innovations – as already discussed – which decreases the quality of the products the users have access to. Not only this, but the prices of the products that the users do have access to, though they are worse in quality, are more expensive due to the motivation of the vendors to make as much money as possible. As Stallman discussed, the motivations between vendors and users are not aligned, for this exact reason, just as they aren't aligned with developers (Stallman, 1990).

The second effect is on the security of the software users have access to. As already discussed, proprietary software vendors often do not update their open source software as frequently as would be ideal. With a solid framework now established, I can now examine why this is the case: there is little motivation to do so. Having potential security vulnerabilities does not decrease profits until those vulnerabilities are discovered or exploited, and by that time it is too late to undo the damage. The profit motive demands that developers always produce something new in order to gain more customers or to fix apparent issues that pose the risk of losing existing customers. *OSSRA* acknowledges this, saying, "with many teams already stretched to the limit building and testing new code, updates to existing software can become a lower priority, aside from the most critical issues. But it's highly possible that a large percentage of that 88% is due to the DevSecOps team being unaware that a newer version of the open source component is available—if they are aware of the component at all" (*OSSRA*, 2022, Page 20).

Another effect of the differing motives behind the two systems of software development can be seen here. The culture of OSS maintains that the user is also in a way a developer, a paradigm that the proprietary software community does not share. Thus, in OSS "the user is expected to be aware of a component's security and stability status and apply new versions as

they become available," (*OSSRA*, 2022, Page 20) while in proprietary software, the producer has that responsibility. OSS components may not be as easily implementable as proprietary components, since they are meant to be edited to suit the user's needs. The difference in culture between the two communities may cause this fact to be lost on some developers, which could lead to a poorly-maintained system.

4. Discussion and Conclusion

The core problem with the current status quo in OSS is the imbalance of power between OSS and proprietary software both monetarily and ethically. This section of the paper discusses possible solutions to this problem, though it does not claim that any one of these solutions would be completely effective in mitigating the issues posed.

A. Internal vs External Use of OSS

When OSS is taken from the community with little given in return, exploitation can occur. One potential solution to this problem is a type of license that includes a clause for "copyleft." What this means is that any work derivative of the licensed work must also be released under the same license or a compliant license. This is an intriguing solution because it forces proprietary software into the realm of OSS if they wish to use the component. The way this solution acts on all parties equally yet only affects proprietary software is quite elegant. OSS can freely use the code since it would likely be released under a compliant license either way, while proprietary software cannot without a large step against their motivations. If these were implemented on a large scale, they could preserve the open source community and minimize intrusions into it.

However, there are a few issues with this approach. Firstly, as I have discussed before, forcing developers to be part of the open source community could degrade the values of the community, thereby decreasing its strength. Such licenses could force actors that disagree with the philosophy to enter and unwittingly disrupt the community. Secondly, these licenses could be ignored on a large scale. Since the OSS community does not, in large part, possess the means to hinder large companies from breaking these licenses, there is little motivation to do otherwise if the component is useful enough. As such, there should be stronger enforcement of these licenses if this is to be a viable solution.

B. OSS Experience in the Hiring Process

The benefits in the hiring process given by work in OSS is a double-edged sword. On one hand, it allows the individual to better their position and prove their worth. On the other hand, it plays into and strengthens the motivations of proprietary software company recruiters; motivations which, when carried to their logical conclusion based on the system they reside within, would severely damage the OSS community. They could render OSS as being mostly a proving ground for those wishing to leave OSS, thereby driving many into OSS that have none of the motivations that thus far have animated the community, subsequently degrading it.

There is also the ethical consideration of unpaid labor to discuss. It is unethical to require unpaid labor before being hired. Promise of reward in the distant future does not change this fact. Requiring OSS experience in order to be hired is therefore unethical.

C. Minimum Wage for Creative Work

A proposed solution to this problem is a minimum wage for OSS developers, thereby making the development of OSS a subsidized activity. While being a drastic solution to the problem, one that most people likely would dismiss out of hand, it has some promise. What has not been discussed up to this point is that development of OSS is a privileged pastime. One must have enough time at hand to spend many hours without any promise that that time will have any monetary return. Having a minimum wage for such work would ease this temporal burden and open the doors to many more developers and their ideas, thereby strengthening the community as a whole. However, the opposite could also be true. Having such a minimum wage could alter the motivations of the community to be more driven by profit than by a desire to do good by and for the community itself. This would make the OSS community much more akin to the proprietary software providers, which is counter to its entire conception.

5. References

2022 Open Source Security and Risk Analysis Report. (2022). Synopsis. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2022.pdf

Aberdour, M. (2007). Achieving Quality in Open-Source Software. *IEEE Software*, *24*(1), 58–64. https://doi.org/10.1109/ms.2007.2

Bijker, W. E., Hughes, T. P., & Pinch, T. (2012). *The Social Construction of Technological Systems, anniversary edition: New Directions in the Sociology and History of Technology (The MIT Press)* (Anniversary). MIT Press.

Collins, K. (2022, July 21). How one programmer broke the internet by deleting a tiny piece of code. *Quartz*.

https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code

Dahlander, L., & Magnusson, M. (2005). Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*, *34*(4), 481–493. https://doi.org/10.1016/j.respol.2005.02.003

Debian Social Contract. (n.d.). Debian. Retrieved February 4, 2023, from https://www.debian.org/social_contract

Giblin, R., & Doctorow, C. (2022). *Chokepoint Capitalism: How Big Tech and Big Content Captured Creative Labor Markets and How We'll Win Them Back*. Beacon Press.

Hardin, G. (1968). The Tragedy of the Commons. Science, 162(3859), 1243–1248. http://www.jstor.org/stable/1724745

Hauge, Y., Ayala, C., & Conradi, R. (2010). Adoption of open source software in software-intensive organizations – A systematic literature review. *Information and Software Technology*, *52*(11), 1133–1154. https://doi.org/10.1016/j.infsof.2010.05.008

Khan, B., Rafiq Mufti, M., Habib, A., Afzal, H., Abdul Moiz Zia, M., Almas, A., Hussain, S., & Ahmad, B. (2021). Evolution of Influential Developer's Communities in OSS and its Impact on Quality. *Intelligent Automation &Amp; Soft Computing*, *28*(2), 337–352. https://doi.org/10.32604/iasc.2021.015034

Merilinna, J., & Matinlassi, M. (2006). State of the Art and Practice of OpenSource Component Integration. *Software Engineering and Advanced Applications*. https://doi.org/10.1109/euromicro.2006.61

Porter, T. M. (1995). Trust in numbers: the pursuit of objectivity in science and public life. *Choice Reviews Online*, *33*(03), 33–1499. https://doi.org/10.5860/choice.33-1499

Stallman, R. (1990). The GNU manifesto. *Oxford University Press, Inc. EBooks*, 308–317. https://web.cs.ucdavis.edu/~rogaway/classes/188/materials/gnu-manifesto.pdf

Van Elderen, M. (2020, May 12). Synopsys Study Shows That Ninety-One Percent of Commercial Applications Contain Outdated or Abandoned Open Source Components. *PR Newswire*. https://www.prnewswire.com/news-releases/synopsys-study-shows-that-ninety-one-percent-of-commercial-applications-contain-outdated-or-abandoned-open-source-components-301057386.html

Von Krogh, Haefliger, Spaeth, & Wallin. (2012). Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly*, *36*(2), 649. https://doi.org/10.2307/41703471

What is open source software? | IBM. (n.d.). IBM. Retrieved February 4, 2023, from https://www.ibm.com/topics/open-source

Winner, L. (2017). Do Artifacts Have Politics? *Computer Ethics*, 121–136. https://doi.org/10.4324/9781315259697-21