7-1-2023

How the Public Shaped the Internet: Open-Source Software Development and Implementation through the Years

Vineyard Wolfenbarger

James Smith

HOW THE PUBLIC SHAPED THE INTERNET: OPEN-SOURCE SOFTWARE DEVELOPMENT AND IMPLEMENTATION THROUGH THE YEARS

Ty G. Vineyard Wolfenbarger
Augusta University
Twolfenbarger@augusta.edu

James N. Smith
Augusta University
Jasmith8@augusta.edu

ABSTRACT

Open-source software (OSS) is ubiquitous, being implemented into nearly 90% of all websites and applications a user interacts with to some degree (Fitchner, 2022). This study focuses on how OSS has changed the technological landscape throughout its history, the various dangers and benefits inherent to its implementation, its relevance to the commercial and public entities, and future works that aim to regulate its use. The information presented provides a holistic and in-depth look at OSS as an independent subject. The topics covered provide characteristics necessary to determine whether OSS is beneficial to the growth of the technological industry, and more importantly highlight areas that can be improved or managed more effectively.

Keywords

Open-Source Software, History, Philosophy, Risk

INTRODUCTION

The world is accustomed to an information environment that integrates technology in a seamless manner; it is easy to forget the exponential rate of innovation that occurs in the cyberspace domain. The average technology user is quick to determine what a new device or software does for them, but often overlooks what goes into the development of that technology. Nearly 90% of the websites or applications a technology user interacts with has at some point implemented a piece of Open-Source Software (OSS) source code into its final product (Fichtner, 2022). OSS is the largest collaboration effort in human history with millions of contributors participating through various platforms. Developers are increasingly implementing OSS elements into projects to keep pace with innovation and the demand for new technology to streamline tasks; allowing them to significantly decrease the time and effort allocated to the software development life cycle (SDLC). The reliance and prevalence of OSS have the potential to introduce numerous security vulnerabilities if source code is not carefully selected and implemented. Because OSS is now ubiquitous throughout the computer environment, there is increased pressure for legislation and for industry leaders to identify a method for regulating how OSS is developed and implemented in an operational environment; there is a real possibility that a single foundational piece of OSS source code could drastically affect not only commercial giants, like Apple or Microsoft, but also governmental agencies, like the NSA or FBI. The discussion of OSS is complex with various ideologies, external influencing factors, and blended outcomes. This paper aims to examine the net consequence of implementing the current ideology of OSS and the potential landscape of OSS based on current events.

There is not a perfect definition for OSS because, depending on the context, the definition may vary. For now, a simple explanation of the critical characteristics of OSS gives a primitive understanding that will further develop as the history, utilization, and future of OSS is discussed in later sections. OSS can be described as computer software that is released under a license that allows users the rights to utilize, study, manipulate, and distribute the software and its source code to anyone for any reason (Updegrove, 2020). A focal point of this definition is the publicly available license that allows anyone that accesses OSS source code to become a contributor and provides litigation protection to the original developer of the source code.

OSS HISTORY AND IDEOLOGY

In the 1950s and 1960s, early computer hardware included a software OS and the necessary compilers without added costs; these elements provided a way for users to access and manipulate source code, the human readable form of software (Stallman, 1998). The critical aspect of allowing a user to access source code gave them the opportunity to customize functions and manually fix bugs as they were identified. The various methods to effectively fix bugs and optimize an OS's functions created a large community focused on sharing successful computer software conformations; this is especially true with universities that

managed more complex computing environments due to research. The rise in popularity and need of computer usage amplified these communities and created a market for software to become a standalone product.

Numerous Independent Software Vendors (ISVs) were founded to capitalize on the new customer pool as a result of the creation of the software market. Computer manufacturers were most of the rising ISV population and began removing software from the purchase of hardware to increase profits (Updegrove, 2020). This action significantly increased the power of companies that produced both hardware and software assets as proprietary products. A proprietary product is any item or idea that is protected by copyright and intellectual property law from any use or modification; these items are often not free, and the inner workings of the product may be held completely secret from the customer. The obscurity and rigidity of these products can be further swayed in a technological business's favor if they only develop hardware with limited compatibility. This forces a customer into a "lock in"; this term describes when a customer is forced into the use of a product due to functional limitations of a complimentary product or extreme financial cost associated with abandoning a product. The limitation of freedom by obscuring of source code, and the significant switching cost would lead to a barrier to change.

The first paradigm shift in software implementation ideology began at the Massachusetts Institute of Technology's (MIT) Artificial Intelligence Department in 1980. Richard Stallman, a software developer, and hacker-enthusiast requested the source code for a printer from the software providers but was ultimately denied due to nondisclosure agreements included the proprietary software in the printer (Stallman, 1998). The creation and popularity of "free software" spawned because the detrimental effects of proprietary technology to foster innovation in software and the ability to quickly address problems. The solution to this problem was first conceptualized in 1981 with the concept of "GNU's Not Unix" (GNU), a free software capable of upward-compatibility with UNIX (Stallman, 1998). Software compatibility with UNIX was a simple choice for Stallman and his team because it provided portability and the great track recorded associated with UNIX. Additionally, a benefit for being compatible with a popular OS was reaching many users that could easily transition over to the GNU software. This idea become reality in 1985 at the cost of Stallman's job at MIT; he was forced to resign to ensure that MIT could not claim ownership of GNU (Stallman, 1998).

The free software movement is often misrepresented and used synonymously with the current OSS concept. The term "free" in this ideology does not necessarily mean that a free software could not have a monetary price. The phrase alludes to the freedom to fully manage a source code when obtained. There are four freedoms that must be present for a software to be considered a free software; identified in the GNU Manifesto by Richard Stallman are: (1) the freedom to run the program as one's wishes, for any purpose; (2) the freedom to modify the program to suit one's needs; (3) the freedom to redistribute copies, either gratis or for a fee; (4) the freedom to distribute modified versions of the program, so that the community can benefit from one's improvements (Updegrove, 2020). Even if a software begins free of charge, an organization or individual may make modifications to it and redistribute it as a proprietary software for profit. The software can still be considered "free software" if it is released without restrictions and the customer has full access to do the described activities listed in the four freedoms. The free software philosophy is a foundational part of the current free and open-source software (FOSS) environment.

From its creation into the 1990s, free software gained popularity in the software development communities because it encouraged users to improve source code by making it available to them. A major flaw in GNU as an OS: it was not a completely "free software" and required the assistance of another source to provide a kernel. A kernel is a computer program at the core of a computer's OS. Richard Stallman's organization, the Free Software Foundation, began working on a free software kernel called Hurd; this was never be released. A hobbyist named Linus Torvalds beat them to the goal. He founded a public project called Linux in 1991 as an initiative to create a simple kernel for a Unix-like OS. Thousands of volunteer contributors participated and made the project a success. The free software kernel was paired with the GNU code to create what the world would consider the Linux, or sometimes labeled GNU/Linux (Balakrishnan, 2018). Linux became wildly successful as a commercial product is the reason the term "Open-Source Software" was developed.

Free software gives an initial impression of a financial cost; this became a problem for developers and organizations trying to get an employer or customer to implement an OSS project. The business and commercial community viewed the term "free" as cheap or unreliable. This became a ceiling for the rapidly growing software industry that was addressed multiple times; the solution came at a strategy conference in Palo Alto, CA in 1998. The conference was held to garner commercial success by marketing free software as a productive tool instead of a philosophical entity. The term "open-source software" was suggested and was generally supported by the conference (Opensource.org, 2018). The change in terminology led to commercial success but tore the information technology community into separate camps with differing opinions on foundational ideologies. They

share the same criteria framed by the four freedoms; some people criticize OSS criteria for being a bit looser in definition, but the true separation is only terminology context. Free software is seen as a social movement stressing the importance of the freedom to manipulate and distribute code as one sees fit; OSS, on the other hand, stresses the optimization of the software and is more palatable in the business world for its cost-effective potential. Both terms can and are often described by an all-encompassing phrase, Free and Open-Source Software (FOSS). The history of OSS illuminates the importance of crowdsource software and defines the characteristics of the internet that is utilized by millions today; a few main attributes include: the freedom to view and manipulate source code of software, the ability to collaborate and innovate common software code projects, and the innate protection and security concerns that come with the countless contributors participating in a project.

An OSS project at minimum implies any user can download the code with the freedom to distribute, utilize, and modify it as they want. There are a few limitations included in the licenses of OSS, and the most crucial is the protection from litigation actions levied against a creator or contributor of an OSS project version. Essentially, a user that decides to apply an OSS project is liable for the risk of flaws in the code, not the author, and must acknowledge the original code author within the new project. Popularity and public availability of an OSS project is crucial to its longevity and relevance. Increased contribution points out bugs and fix-actions, which increases the addition of improved code to a project and helps document and promote the ongoing work of others.

The primary control mechanism from a legal perspective is managed by copyright laws and license agreements (Fichtner, 2022). There are over 200 different license agreements with different terms. The difference in components often causes issues when trying to coordinate OSS products to form software stacks or utilize an OSS product with a specific hardware. There are three common versions of an OSS license that differ in components; these licenses are commonly referred to as permissive, semi-permissive, and restrictive licenses. With a permissive license, the only requirement is that the user keeps the copyright notice in place when a software is distributed. The advantage of a permissive license is the ability to use and change the OSS. A semi-permissive license requires any modifications to the source code of a software be made available under the specified terms of the given license. The license may define what is a modification. To comply with the license, the developer would have to release the source code in three forms: the original code, the modified code, and the newly added code. A user under this license has implicitly accept the terms of use; and is subject to the same liability as the original OSS developer. To coordinate with other software, the potential implemented OSS must follow all controls listed in the license; the license with the strictest guidelines becomes the limitation for usage. A restrictive license includes a "copyleft" clause; the clause ensures that OSS maintained by the license or any stack that includes a component under a restrictive license will never become a proprietary product and remain accessible by the public. Depending on how it is integrated with proprietary software, the user may be subject to significant legal risk. In the best-case scenario, proprietary software will only be required to fall under the same royalty-free license.

An OSS project can initially be launched by an individual, independent group, or even a commercial organization if they would gain benefits as a result. The format and features of a project are fluid and typically tailored to foster interest from a typical group or skill set; a project owner has many tools to do this, but the most common characteristics of choice include determining which coding languages it is compatible with (C++, JAVA, Python, etc.), what the primary function of the developing product is, and which platform the project is being maintained on (GitHub, Veracode, etc.). A project owner's goal is to align all the dependent variables involved with OSS project management with their target software developer audiences' preference (Coletta, 2022). The more aligned these aspects are, the more likely the project will obtain some form of interaction. Furthermore, an increased number of interactions enhances the chance of receiving code worth implementing into the project's source code. Prior to the introduction of a project, a list of criteria or a management board must be developed to review and integrate suggested code into the updated versions of a project. Multiple projects can collaborate to form software "stacks" to provide an essential service. The term "stack" references separate components working collectively to meet a specified function; a previously mentioned example is the Linux kernel and GNU combining to form the GNU/Linux OS. An OSS project is a direct representation of the four freedoms ideology that founded the FOSS movement, offering project owners a great deal of customization and software developers the freedom to choose which projects they contribute based on their preferences. The existence of choice and limitless participation creates the presence of inherent shortfalls and benefits that require discussion prior to considering how OSS is applied in the commercial and legal lenses.

OSS offers a cost-effective way for businesses to implement software to keep pace with competition and a method for software developers to build skills and experience for future employment endeavors. These are important qualities to consider when looking into OSS, but what are the costs? There are two key areas to introduce potential users or developers to the shortfalls of

OSS; (1) the longevity of an OSS project and (2) that the vulnerabilities of an OSS project are public knowledge (Fichtener, 2022). There are others, but the two listed are a bare minimum to understand and mitigate against.

Most individuals participating in OSS projects are volunteers. Usually, employers do not guarantee any monetary incentive to continue the development and maintenance of a project. The primary motivation for the developers working on OSS projects includes a sense of enjoyment, overcoming a challenge, gaining status within the project community, and gaining valuable job skills that enhance marketability and compensation potential. A developer may lose interest in a project or become involved in multiple activities and no longer sees value in participating. Software development is a compounded action, meaning certain software relies on previously written and managed code. A business could completely utilize only one OSS to complete a major security function; the ramifications of the OSS losing support is a major vulnerability that will never be fixed and may require a complete reconstruction of the source code or business model that was reliant of the lost OSS project.

Poor management and the loss of various OSS projects has occurred multiple times, but none more significant than the Heartbleed bug in 2014. This exploit targeted OpenSSL, the most popular open-source cryptographic library. The function of the Secure Socket Layer (SSL) and Transport Layer Security (TLS) implementation was to encrypt traffic on the internet. The attack was able to exploit a security flaw in products that incorporated OpenSSL world-wide; an estimated 20% of web servers on the internet were affected and led to the loss of sensitive data (Carvalho, 2014). After an investigation of the attack, the cause was deemed avoidable, and the result of poorly managed OSS source code maintained by two woefully overworked and underpaid programmers. The critical take-away for this section is that OSS as a source to provide functionality is based on the contributors; this may not be a stable source and should only be implemented with caution.

OSS code and its flaws are constantly under scrutiny. However, this is not necessarily a bad thing. Organizations like the Open Web Application Security Project (OWASP) and the National Vulnerability Database (NVD) have been founded to warn organizations and individuals of major OSS vulnerabilities, so problems are always well vocalized. In addition to OSS source code being open to the public, software developing communities are given early notice to correct problems. It has become common knowledge that OSS project freedom allows volunteer developers to fix problems faster than their proprietary counter parts because there are less administrative processes to overcome (Collins, 2021). Proprietary processes require a software development team to receive a problem, identify where in the code the problem lies, and then begin looking for solutions; this process could take weeks to months.

Although faster to reacting to vulnerabilities, there is still a major consideration and shortfall with vulnerabilities being public for OSS projects that is not a concern with a proprietary software. The identity of who contributes to an OSS project is relatively unknown; the motivations of these individuals cannot be weighed when considering the implementation of potential code. The use of a practice called "hypocrite commits" is a common way to take advantage of this fact; the term describes a user submitting a fix action to a relatively harmless, buggy section of code correctly, but introducing a much more dangerous avenue of exploitation with the code recommendation (Wu, 2021). This action was demonstrated when the University of Minnesota (UofM) conducted a case study targeting Linux with this form of exploitation in 2021. When a recommendation was accepted, the researchers then explained the study and provided a solution to the problem without the addition of a more dangerous flaw (Wu, 2021). Due to the violation of pen-testing, UofM was banned from contributing to the Linux kernel. The concern is that there were recommendations that did go through. The reason for the experiment's success was similar to the art of pre-texting. The university had a good standing by their educational and research-oriented background; they even had students or professors contribute unrelated source code corrections to further improve the implicit trust Linux gave to UofM. There is the need for increased scrutiny from the perspective of an OSS owner when accepting recommended code.

There are benefits to highlight regarding OSS which include user access to code, recycled code and bug fixes, cost-effective source of software, and another form of security. When compared to traditional software, the greatest flexibility is seen with the access to code; OSS customers have the ability and right to change code at any time to address any reason. The traditional method puts dependence entirely on the vendor to make changes at their discretion and any sort of customization comes at an additional cost. There is a risk with a vendor going out of business for the customer because they do not know what source code is running a specific function and would have to immediately hire a similar business to mitigate the risk. Customization and the associated cost should be emphasized because OSS requires a one-time fee to acquire and own a function, unless the business wants to hire a developer to update and maintain the source code internally (Collins, 2021). Conversely, a vendor-based method often requires a recurring payment to enlist the support and maintenance for a traditional software because they are the only ones allowed to legally update the code. This stark contrast makes OSS more attractive from a financial standpoint.

From an operational standpoint, the difference in time to implement favors OSS. Successful OSS projects are in constant motion, upgrading and fixing bugs in real time. Volunteer developers access and update work more frequently than a traditional product; a vendor typically waits to incur the cost related to making minor and major releases. That time to implement represents a period of vulnerability that may not be acceptable to an individual or organization. Source code being constantly visible seems counterintuitive to overall security, but popular OSS programs are acknowledged to be more secure (Collins, 2021). The reason for this phenomenon is because anyone can see the code, track down the source of a vulnerability, let project managers know of the cause of concern, and propose a fix themselves. Because OSS is cheaper to apply and fixes bugs quicker than most proprietary products are willing to address, OSS is now being selected by defense, financial, and other users who place the highest priority on security. The benefits for implementing OSS make a strong case for its widespread use, but the shortfalls that infer a proportional relationship with a program's popularity needs to be considered because obscure code with little participation is dangerous and riddled with potential vulnerabilities.

CURRENT EVENTS

The history of OSS only spans a period of about 40 years since its initial concept yet has grown to a global level. The ubiquity of the Android OS and the detrimental effects that resulted from the Heartbleed bug are merely two examples of the global footprint of OSS. The rapid success of OSS has captured the attention of the tech community, but it wasn't until 2021 that the United States (U.S.) Federal Government considered the power of OSS as something that required regulation (OpenSSF, 2022). The U.S. government and subordinate state governments have made it clear that regulation of the internet and related technologies is of importance to national security, but there was never a perceived need for legislation regulating OSS at the federal level. However, the OSS related vulnerability, known as LOG4J, provided a strong enough justification to draft-up the Securing Open-Source Software Act of 2022 in September of 2022(which, as of February 1, 2023, remains under review by the Committee on Homeland Security and Governmental Affairs).

LOG4J is a Java-based Apache utility that contained a zero-day vulnerability known commonly as Log4Shell. This vulnerability dated back to 2013 but was not discovered until 2021 by employees at the company Alibaba. The Log4Shell allows an attacker to remotely execute code that can grant full server control. The issue is critical and widespread because LOG4J is ubiquitous in most Java applications. The flaw is caused by poorly written source code that controls the function of remote code execution. This control allows a malicious actor to execute arbitrary code introduced by the Java Naming and Directory Interface (JNDI) API. The code, mimicking the API, allows access to the server (Lim, 2022). Although the total impact is unknown, experts assess that the flaw will affect security indefinitely because the flaw in the code is at such a foundational level.

The purpose of the Securing Open-Source Software Act of 2022 is to help protect federal agencies and critical infrastructure systems by strengthening the security of software from both open and close sources. The LOG4J incident is particularly worrisome because it highlighted a serious threat to federal systems and critical infrastructure. An assessment of the damage LOG4J caused by the Cyber Safety Review Board found that the effects of the flaw will be relevant for decades because of how many organizations implemented the code into their processes (OpenSSF, 2022). The bill identified the Cybersecurity and Infrastructure Security Agency (CISA) with the responsibility for strengthening the regulation of software for critical infrastructure. The legislation requires that the CISA develops criteria for identifying and hiring experts in OSS to create a Software Security Advisory Subcommittee; the purpose of this group is to review and approve software to be implemented in the federal government and organizations that handle U.S. citizens' sensitive data. A larger responsibility of the group is to produce an initial assessment framework for managing OSS code risk and distribute them to all relevant stakeholders. This document aims to provide best practice solutions to minimize damage caused by flawed code being implemented in lower-risk developer communities to address the problem. The Act will create a pilot program to affect rapid change at each federal agency. The program will initiate with a single entity and make improvements before a wider sample size is tested. The program will feature the creation of an Open-Source Program Office (OSPO) overseeing the coordination of open-source program activity within organizations. CISA will be managing the creation and oversight of the program but delegating the task of training and educating the respective Chief Information Officers (CIO) on risks and management of OSS to the Office of Management and Budget (OMB) (OpenSSF, 2022).

The proposed legislation is still in the nascent stages of development and is subject to change as it is scrutinized by more software-related professionals. The creation of the Bill by Congress represents an affirmative step in the right direction to address the lack of regulation in OSS as a major cybersecurity challenge. This increased focus on software-related issues

presents an opportunity for the community to be more involved in shaping the software environment through expertise of skilled OSS professionals. In closing, the interest of the government to increase the inspection of how an organization evaluates, implements, and manages OSS in key infrastructure can only benefit a significant risk associated with a large industry only being kept in check by the legal repercussions outlined in license agreements. There is also an implied understanding that the public is starting to understand that blindly trusting an OSS product has significant and lasting consequences.

CONCLUSION

OSS has emerged as the most utilized, outreaching, and diverse form of software in the world. There are different ideologies when it comes to FOSS products, but the underlying idea of freedom and collaboration developed security and innovation through transparency of source code. Although OSS offers a format to innovate and provide freedom to the individual user level, there are glaring obstacles in the form of unregulated and varying license agreements, irregular and unrefined management of source code suggestion, and the lack of an overarching framework to regulate the OSS industry. Currently, the most detrimental argument to the use of FOSS technology is the potential to introduce persistent vulnerabilities through poorly written code and the chance that an OSS project is ran by an inconsistent volunteer force that misses key vulnerability updates. The existence of major OSS flaws, such as Heartbleed and LOG4J, helped identify the need for strict regulation and distribution of OSS technology; furthermore, they forced the U.S. Federal Government actively draft the initial framework to positively shape future OSS project development and implementation. If the governmental pilot programs are successful, the potential to introduce a new major bug drops significantly; conversely, the value of utilizing OSS product sharply increase because it is more cost-effective and addresses flaws at a faster rate. Future works must address similar OSS implementation programs/processes and look to create a method for managing the various licensing agreements to allow for collaboration amongst FOSS products. In closing, OSS should be used for its ability to meet software functions securely and cost-effectively in the modern era, if there is additional consideration on research conducted when selecting an OSS, the existence of redundant technology to cover OSS, and a system in place to manage and update the OSS.

References

1. Balakrishnan, A. (2018, July 19). The History of Free and Open Source Software, for the ‘Third Generation’. Medium. Retrieved December 13, 2022, from https://medium.com/fossmec/the-history-of-free-and-open-source-software-for-the-third-generation-9997b5f6e73c
2. Carvalho, M., DeMott, J., Ford, R., & Wheeler, D. A. (2014). Heartbleed 101. IEEE Security & Privacy, 12(4), 63–67. https://doi.org/10.1109/msp.2014.66
3. CNBC, N. (2019, December 11). The rise of open-source software. YouTube. Retrieved December 12, 2022, from https://www.youtube.com/watch?v=SpeDK1TPbew
4. Coletta, J. (2022, December 10). The Risks Associated with OSS and How to Mitigate Them. Contrast Security. Retrieved December 12,2022, from https://www.contrastsecurity.com/security-influencers/oss-risks-and-mitigation#:~:text=At%20the%20same%20time%2C%20open,that%20leads%20to%20security%20gaps.
5. Collins, H. (2021, April 23). Is Open Source Software more Secure than Proprietary Products? GovTech. Retrieved December 13, 2022, from https://www.govtech.com/security/is-open-source-software-more-secure.html
6. Fichtner, E. (2022, February 14). 7 risks posed by open-source software and how to defend yourself. Datto. Retrieved December 12, 2022, from https://www.datto.com/blog/7-risks-posed-by-open-source-software-and-how-to-defend-yourself
7. Lim, W. C. (2022, August 30). Apache log4j vulnerability explained. Swarmnetics. Retrieved December 12, 2022, from https://www.swarmnetics.com/blog/apache-log4j-vulnerability-explained/
8. Opensource.org. (2018, October). History of the OSI. History of the OSI. Retrieved December 13, 2022, from https://opensource.org/history
9. OpenSSF (2022, September 29). The United States Securing Open Source Software Act: What you Need to Know . Open Source Security Foundation. Retrieved December 12, 2022, from https://openssf.org/blog/2022/09/27/the-united-states-securing-open-source-software-act-what-you-need-to-know/
10. S.4913 - 117th Congress (2021-2022): Securing Open-Source Software Act of 2022. (2022, September 28). http://www.congress.gov/
11. Stallman, R. (1998, December 25). About the GNU project - GNU Project - Free Software Foundation. GNU Operating System. Retrieved December 12, 2022, from https://www.gnu.org/gnu/thegnuproject.en.html

12. Updegrove, A. (2020, January 23). A Brief History of Open Source Software. ConsortiumInfo.org. Retrieved December 12, 2022, from https://www.consortiuminfo.org/open-source-open-standards/a-brief-history-of-open-source-software/