Security Concerns in Proprietary and OpenSource Software

Dorian Gregor Rabl
h12113228

Sommersemester 2024
4117 - Seminar aus BIS
LV-Leiter: ao.Univ.Prof. Dr. Rony G. Flatscher

Wien, am 05.06.2024

Structure

Structure **1**
1. Introduction **2**
1.1 Thematic Overview 2
1.2 Necessity of the Analysis 2
2. Fundamentals of Software Security **4**
2.1 Core Principles 4
2.2 Identifying Vulnerabilities and Threats 6
2.3 Strategies for Security Enhancement 7
3. Security Aspects of Proprietary Software **9**
3.1 Challenges 9
3.2 Risk Mitigation and Security Features 10
3.3. Comparative Advantages and Disadvantages of Proprietary Software 12
4. Security Aspects of Open-Source Software **14**
4.1 Community-Based Security Models 14
4.2 Balancing Transparency with Security 16
4.3 Comparative Advantages and Disadvantages of Open Source Software 17
5. Key Findings and Future Trends **19**
5.1 Key Findings 19
5.2 Future Trends in Software Security 20
5.3 Implications for the Future 21
6. Conclusion **24**
References **26**

1. Introduction

1.1 Thematic Overview

In the present day, software security has become a key concern for people and institutions in dealing with the digital world. Dependency on software is increasing manifold to manage many of our important operations and data. This requires the development of security measures that are very much rigorous and foolproof. This seminar paper points out the fact that there are security issues with both proprietary and open-source software. It identifies the fundamental principles, strategies, and challenges in either model.

Proprietary software is such software for which source code is closed and, in general, supplied for money. Generally, only developers or the people who are allowed have access and the rights to modify the software. Microsoft Office and Adobe Photoshop are examples of proprietary software (Kilamo, Hammouda, Mikkonen, & Aaltonen, 2012). Proprietary software's important property is its closed-source nature. This means that only the owning organization has the right to modify and redistribute the software. The model has been such to guarantee that a single entity develops and maintains the software, controlling the access and updating of the source code (Kilamo et al., 2012).

Open-Source-Software (OSS) is software that is accessible to anyone wishing to modify and distribute the software. Generally, this software has a license that allows for the study of the software and for modification. Examples of open-source software are Linux and Mozilla Firefox (Kilamo et al., 2012). It is the model on which OSS is developed that principles of transparency in the process of development of software and community development have been given priority. This implies that anyone can contribute to the development of the software and review the code for issues. According to Raymond (2005), the model for OSS development is that of a bazaar, in which a large number of developers are associated to develop and review the code in contrast to the centralized and controlled approach of proprietary software.

1.2 Necessity of the Analysis

The Analysis of the security concerns of proprietary and open-source software is essential for a number of reasons. One, discovering core security principles and how they are implemented across different software development models could enable an organization to make more informed decisions in the type of software they choose to go with. Second, the stakeholders can only enhance the security overall if they identify potential common vulnerabilities and threats and, in turn, effective mitigation strategies.

The comparative analysis of proprietary and open-source software will provide insight into the strengths and weaknesses of the two models that, in turn, help businesses to balance transparency, security, and cost-effectiveness in one way or another. Nowadays, many new technologies along with the framework of regulations that are evolving hand in hand, keeping a track record in advance of the software security trend is important from the point of view of compliance and also for the prevention of the new and revised, sophisticated cyber threats.

The core research questions discussed in this paper are as follows:

- What are the core security principles guiding both proprietary and open-source software development, and how are they implemented?
- How do security strategies and vulnerability management processes in proprietary and open-source software differ, and what does this imply?
- What are the emerging trends in software security, and how would they affect the future of developing software using both proprietary and open-source methodologies?

These, in turn, through the answering of these questions, will allow this paper's readers to understand the dynamics of security within and outside proprietary and open-source software—hence contributing towards a better understanding for developers, security professionals, and decision-makers in the technology domain.

2. Fundamentals of Software Security

In an increasingly digital world, software security is critical for protecting both digital devices and valuable information. Security is not only a technical challenge but also an organizational one. The following sections explore the fundamental principles of software security, beginning with the core principles, followed by the identification of vulnerabilities and threats, and strategies for enhancing security. These core principles can be identified regardless of whether the software is open source or proprietary, highlighting universal best practices in ensuring robust software security.

2.1 Core Principles

Software Security is very crucial for protecting digital devices as well as valuable information. According to Glasauer et al. (2024), the basic building blocks for making any software secure, we need to follow certain core principles. These principles help us to prevent weaknesses and protect the system from cyberattacks. It is partly dependent on the technical skills of engineers, but a large chunk is down to the organizational culture in the company developing the software.

One key principle that Glasauer et al. mention is security by design. Which means that, software developers should be thinking about security from the very beginning of the process of software development. They need to design systems that can handle problems like attacks, mistakes, and accidents. This includes for example reducing dependencies between components and also ensuring the software operates with the minimum level of access needed. By building security into the design, we can reduce vulnerabilities (Glasauer et al., 2024).

The principle of least privilege addresses the damage arising from a security breach. In other words, it dictates that the software should run with access at the minimum level required to function. We can manage the access limit, which prevents attackers from gaining widespread control over the system in case there is an exploit (Glasauer et al., 2024).

Conduct risk analysis and threat modeling regularly to detect and mitigate security threats at an earlier stage in the development process. Glasauer et al. underline that understanding the potential attack points and weaknesses of the software well enough can be described only by following these practices. Threat modeling predicts several possible paths of attacks and contributes to the preparation of strong defenses (Glasauer et al., 2024)..

During the coding phase, developers should follow the best practices for secure coding. This consists of using the latest compiler software, tools for analyzing the source code and making clean and simple code. This ensures early identification

and rectification of security errors to make the software robust to different attack scenarios (Glasauer et al., 2024).

Another core principle is the continuous testing and auditing of software throughout its development. Glasauer et al. highlight that testing at different stages helps ensure that security requirements are met and maintained. Regular audits and security checks are crucial for finding and fixing vulnerabilities before the software is released (Glasauer et al., 2024).

Not only the technical measures, but also the organizational culture plays a crucial role in ensuring software security. Glasauer and his colleagues further state that many of the cultural properties that determine the establishment of a secure software environment are characterized by the following: Security culture. Instilling a strong security culture in an organization creates awareness to the extent of making employees care about security. Also, spending shared responsibility for secure development practices under which organizations work towards optimizing workplace activities for proper arrangement of resources and ensuring an amiable environment where secure development becomes a norm (Glasauer et al., 2024).

The central core principles of secure software, from technical and organizational considerations, include security by design, least privilege, regular risk analyses, secure coding practices, continuous testing, and a supportive organizational culture. As underlined by Glasauer et al., only the complete establishment of such principles, both in the technical and cultural practices of the company, can guarantee high-level security performance in developing software processes and products in general (Glasauer et al., 2024).

These core principles are also shared by Brunetti et al. (2014), highlighting the importance of infusing organizational culture with security measures from the very beginning. According to Brunetti et al. (2014), recent trends in software development are radically focused on the enhancement of security measures, which simply means that the security issues for software development have to be treated quite holistically both at the technical and organizational levels.

2.2 Identifying Vulnerabilities and Threats

Therefore, to develop secure software and prevent frequent security breaches, there is a need to understand software vulnerabilities and threats. Bojanova et al. (2023) also said that a more definitive categorization of allowable terms, such as bug, fault, error, weakness, and vulnerability, should be done so that security discussions can be made more effective and security action plans can be made more precise.

Bojanova et al. (2023) pointed out that clear-cut definitions obviate confusion among security professionals. A software security vulnerability is, on the other hand, a chain of weaknesses that could lead to a security failure. A bug starts this chain; that is, a fault in the code or specification. The bug will be triggered into an error, which forms a fault that drives consequent weaknesses. These weaknesses propagate until a final mistake is reached, resulting in a security failure. A security bug, according to the definition by Bojanova et al., is a primary defect of both a software code and its specification. A fault is an error caused by data or an address of an incorrect type, data, or size. Errors originate from operations carried on by bugs or faulty operands and spread through the software to cause further weaknesses. This is the final error in a chain, which the attacker exploits to cause a security failure.

Bojanova et al. (2023) listed some types of vulnerabilities that might arise in the process of software development:

- Data Validation Weaknesses: These occur when inputs are not correctly checked and they may permit wrong or malicious data into the system.
- Memory Management Errors: These include issues like buffer overflows, where more data is written to a buffer than it can hold, leading to system crashes or exploits.
- Arithmetic Errors: These are errors in mathematical calculations that can result in incorrect data processing and security breaches.

Temizkan et al. (2017) give a different focus by looking at how software diversity might help enhance network security and manage shared vulnerabilities. They propose that distributing different software versions across systems can lower the risk of widespread security breaches. This method, called software diversity, reduces the potential impact of common vulnerabilities across many systems. They further argue that using a variety of software in different systems reduces the network's vulnerability. In such a case, if many systems run on various software and a particular vulnerability emerges within one system, it does not put the other systems at risk, thereby limiting the extent of a vulnerability.

2.3 Strategies for Security Enhancement

In the earlier chapters, an understanding has been provided regarding the core principles, vulnerabilities, and threats in software security. Effective security augmentation necessitates both organizational practices and technical measures. The present chapter explains some of the advanced strategies for software security enhancement: using secure design principles, software diversity, and fostering a culture of security within organizations.

In the context of proprietary software, the responsibility for security augmentation usually lies with the software vendor. It is in the interest of such vendors to apply advanced secure design principles and to deploy continuous updates and patches. They are under legal accountability for the security of their products, and many of them have specialized teams to manage security risks. For instance, companies like Microsoft and Adobe are known to release security patches and updates on a regular basis to handle the vulnerabilities in their software (Weir, Rashid, & Noble, 2020). Proprietary vendors often follow strategies like defense in depth, which uses multiple layers of security control in a way that provides failure redundancy in case one layer breaks, thereby protecting the organization from a wider range of attacks (Bojanova et al., 2023).

On the other hand, open-source software (OSS) depends on a community-driven approach for security augmentation. Responsibility is shared among a community of global developers and users. Such a model of working allows for intensive peer review and the swift identification and fixing of vulnerabilities. This also means, however, that the security of open-source software is extremely variable depending on the activity and expertise of the community involved. Projects such as Linux and Mozilla Firefox benefit from very active communities that provide frequent updates and patches for their software (Von Krogh & Von Hippel, 2006). A software diversity strategy is another key strategy used in OSS: it relates to using different operating systems, databases, and application servers to reduce the risk that a single vulnerability will compromise all systems (Temizkan et al., 2017).

In both models, fail-safe defaults are important. This ensures that systems transition to a secure state if there is a failure, usually to deny access by default and to grant it by exception. Although not a primary security mechanism, security through obscurity adds a layer of defense. In any case, that should not form the principal security strategy on which to depend; instead, it should support others (Bojanova et al., 2023).

It is, therefore, evident that creating a culture of security within organizations is paramount for both proprietary and open-source software. In the case of proprietary systems, security is often motivated through internal policies and driven by security teams. Security teams should exercise the security aspects, run awareness

initiatives, and even include security as a performance metric in staff appraisals to ensure that security is a consideration within day-to-day operations (Glasauer et al., 2024). In open source, a culture of open communication and collaboration is essential. Security champions within the development teams can mentor their colleagues on best practices and spread the principles of embedding security into every aspect of the software development lifecycle.

On the final note, although the basic tenets of software security - such as security by design, least privilege, regular risk analysis, secure coding practices, and continuous testing - are similar for both proprietary and open-source software, the responsibility for these principles is different. Traditionally, accountability for security in proprietary software is vested in the vendor, while open-source software depends on collaboration from the community. Both benefit from advanced strategies, such as defense in depth, software diversity, and a strong security culture, to add strength to software security.

3. Security Aspects of Proprietary Software

Proprietary software entails some unique security challenges by its very nature of being closed-source and dependent on vendors for updates. This chapter delves into these challenges, the strategies used in mitigating associated risks, and the comparative advantages and disadvantages of proprietary software in maintaining robust security measures.

3.1 Challenges

A key security challenge in proprietary software is the potential for hidden vulnerabilities. Open-source software, by virtue of the availability of source code, is scrutinized by a large number of developers and security experts worldwide. On the other hand, proprietary software restricts access to its source code, creating a barrier for external parties to report security flaws in them. According to Cadariu et al. (2015), the closed nature of proprietary systems can leave vulnerabilities unaddressed, especially in third-party components. Their study showed that many proprietary projects contained vulnerable third-party libraries, which brings to light the hidden risks in the software.

Proprietary software generally depends on vendors for updates and patches to security concerns. For instance, Microsoft Windows has an isolated security update team that ensures that security updates are issued regularly (Microsoft, 2024) This dependency can be a problem if the vendor is slow to respond to emerging threats or if the update process is cumbersome for its users. Meanwhile, delays in patching known vulnerabilities leave systems open to attacks for long periods. Weir et al. (2020) stressed that to have good security in proprietary software, you actually have to take a proactive stance, doing regular updates and holistic security reviews across the whole software development lifecycle. But centralizing control of those updates as a feature of the proprietary model can mean that response to security concerns gets dragged out.

Insider threats are again a major issue. Employees or contractors with access to the source code or critical systems might deliberately or unintentionally introduce security vulnerabilities. These threats are mitigated via robust access controls, regular audits, and monitoring systems to detect and respond to suspicious activities. Sometimes it is more difficult to have comprehensive monitoring in a closed proprietary solution than in an open-source environment where community control has a crucial role (Dhir and Dhir, 2017).

Transparency and the ability to properly perform external auditing are restricted due to the close access to source code by proprietary software. These prevent identification of best practices and hindrance in the identification of security flaws. Contrastingly, community-driven reviews and contributions mean that it receives

better community support in relation to the security of the open-source software. The essential external scrutiny that proprietary software lacks due to access restrictions is compensated by open software through rigid internal security practices and regular independent security assessments (Weir et al., 2020).

The addition of third-party components to proprietary software introduces significant additional security risks. The difficulty in managing the security of these components is mainly because the vendors have to ensure that they update frequently and are free from known vulnerabilities from the integrated libraries and frameworks. Many projects, most with proprietary code, as Cadariu et al. (2015) found, include highly vulnerable third-party libraries. This shows the importance of constant monitoring and updating of all components within the software.

3.2 Risk Mitigation and Security Features

To reduce security risks, proprietary software vendors deploy many strong security functionalities and approaches. These include encryption, access controls, intrusion detection systems, and frequent security patches. The success of these measures is dependent on the ability of the vendor in handling emergent threats and vulnerabilities.

A critical element of risk mitigation on proprietary software is the patch release behaviors by the vendors once a vulnerability has been detected. According to Temizkan et al. (2012), the severity of the vulnerability and legislative pressures affect the speed at which vendors release a patch. "Vulnerabilities with high confidentiality impact or high integrity impact are patched faster than vulnerabilities with high availability impact" (Temizkan et al., 2012). This means that the vendors value the aspect of data leakage or corruption more than data loss. The essence of this is to avoid persistent data loss.

A comparison between open and proprietary software vendors indicates that "open source vendors release patches faster than proprietary vendors" (Temizkan et al., 2012). The reason for this is that open-source development is a community effort, making it possible for scrutiny by the community and contribution towards making the software free of vulnerabilities, thus increasing the speed for reducing vulnerabilities

The legislative pressures are very influential. The study states that "appropriate regulation can be a powerful policy tool in exerting the effect in vendor behavior towards socially desirable security outcomes" (Temizkan et al., 2012). This shows how regulatory frameworks are useful tools in ensuring a timely and successful response to vulnerabilities.

Proprietary software vendors often incorporate strong encryption to secure sensitive data. Encryption ensures that data is unreadable to an unauthorized user, even if they obtain the data. This is particularly crucial in protecting personal and financial information. Magnanini et al. (2022) underscore that strong encryption is a crucial part of secure software updates. It works to guarantee confidentiality and integrity through the update process. Strong access controls are also implemented to ascertain that only duly authorized personnel access the systems and data assets that are deemed to be critical. Among such access controls are multi-factor authentications, role-based access controls, and biometric authentication.

Boulanger (2005) also validates the use of multi-factor authentications and role-based access controls in securing the proprietary software environment (Boulanger, 2005).

A proprietary software often also incorporates an intrusion detection system to bolster security. Such a system monitors both the network traffic and the activities in the systems to detect or identify any anomalous behavior that would further trigger a red flag or alarm regarding the existence of a security breach. IDS systems can be set up to trigger an alert to the administrators or even a pre-defined action upon detection of potential threats. This, therefore, becomes a proactive approach, which identifies and mitigates security threats before they impact the system. Weir et al. (2020) also validate the use of the IDS system in detecting and responding to potential threats in real-time, thereby fortifying the overall security posture of proprietary software. (Weir et al., 2020).

Regular security audits and penetration testing are some of the most important elements of a security strategy. Security audits are systematically inspecting the software and its environment to detect possible vulnerabilities and verify that security standards are being met. Penetration testing is testing the software by simulating some form of attack to discover and remediate all forms of security weak points. Weir et al. (2020) note that using a dialectic approach, meaning conflicting interactions between the developer and a security expert, will radically help this because the developers will gradually adopt secure programming methods, thereby getting continuous security improvement.

Equally important in security are the mechanisms through which updates and patches are delivered. Proprietary software vendors commonly maintain secure and scalable mechanisms for updating software to protect all users and systems in a timely manner. As opposed to this, Magnanini et al. (2022) have placed great importance on developing scalable, confidential, and survivable software update mechanisms whereby software updates are distributed efficiently and securely, even in adverse conditions (Magnanini et al., 2022). It helps in keeping the software's integrity and security proper all this while.

3.3. Comparative Advantages and Disadvantages of Proprietary Software

Proprietary software offers several advantages and disadvantages in the context of security, which can significantly impact its adoption and use in various environments.

Advantages
A central advantage of closed-source software is the proprietary support services that come with it. As explained by Boulanger (2005), such specialized support allows users to receive professional help when they encounter problems. Vendors usually have total customer service, such as troubleshooting services, regular updates to the software, and patches to security. This special support is instrumental in addressing any issues or vulnerabilities that could be identified in a quick and effective manner.

Closed-source software suites often have integrated features that are designed to work in smooth harmony. Temizkan et al. (2012) noted that such integrated features may increase the security of the software in question. Harmoniously working integrated components have the complexity and vulnerabilities that might arise from integrating separate tools reduced.

Legal accountability is another key advantage that vendors of closed-source software must adhere to with regard to the performance and security of their software. According to Weir et al. (2020), this accountability ensures that stringent security policies are adhered to by the vendors, who will also be quick in responding to identified vulnerabilities. Users get more confidence in using it when the vendor has the liability of ensuring they address any concerns that could be identified.

Disadvantages
A major disadvantage of closed-source software is that it lacks transparency due to its nature of being closed source. According to Cadariu et al. (2015), in closed-source software, the source code is hidden from the public, which means that they will not be able to conduct security audits nor independently verify the code. The existence of such latent vulnerabilities makes it possible for them to remain undiscovered until they are exploited by attackers, simply because the source code is only kept by the vendor.

The costs of closed-source software also prove to be a huge drawback. As has been observed by Dhir and Dhir (2017), there are huge licensing costs associated with such proprietary software, which are not only incurred during the initial acquisition but are also incurred in further maintenance and for support. In the long run, these costs accumulate, making the closed-source alternatives more costly than the open-source alternatives.

Moreover, proprietary software users are critically vendor dependent when it comes to the availability of updates and patches related to system security. As Temizkan et al. (2012) explained, such dependency can become cumbersome, specifically in the context of discovered critical security vulnerabilities that need instant rectification. In the event that a vendor is delayed in providing the fix, that leaves systems exposed to attacks until that time.

4. Security Aspects of Open-Source Software

Open-source software (OSS) uses the mechanisms of its transparent development model and community efforts to address security challenges. This chapter will explore the unique security dynamics of OSS, including its advantages and disadvantages.

4.1 Community-Based Security Models

Community-based security practices are crucial to open-source software (OSS) development and maintenance. Such models combine the efforts of many contributors with many different views on how software security could be improved. Thus, OSS exposes a wide range of security practices and strategies different from those found in proprietary development.

One of the most important benefits of the community-based security model is the exhaustive peer review. The open-source community comprises developers and security experts who go through the source code, find vulnerabilities, and make improvements. This collective, complex process of checking the source code helps in finding and combating security threats at an early stage. Linåker and Regnell (2020) claim that wide community contributions serve as a compensating mechanism of objectives and open-source development complexities, thereby ensuring strong security practices. Similarly, Von Krogh and Von Hippel (2006) stress that the open-source model helps to challenge a series of established assumptions about innovation—most importantly, that user communities can effectively steer the development and supplementation of software, including the development of security features.

The collaborative nature of OSS development leads to several unique security practices. Due to the volunteer nature of most open-source development, for example, community members are likely to conduct periodic reviews of code to identify potential security problems before they are exploited. According to Thompson (2017), practice of code review in modern times is useful for maintaining the security of OSS. Many OSS projects also have bug bounty programs as a method of encouraging both security researchers and developers to find and report vulnerabilities in a proactive manner. According to Malladi and Subramanian (2020), bug bounty programs have become a part of the security arsenal for many companies, allowing them to harness a global pool of security researchers to find and fix vulnerabilities.

Indeed, there are many examples of successful community-based security models in OSS projects. The Linux kernel is a great example, as its wide peer review and the active involvement of individual contributors and corporate entities make this operating system very secure and reliable (Thompson, 2017).A notable instance of

its robustness was seen during the Heartbleed vulnerability in 2014, where the OpenSSL community quickly identified and patched the issue, showcasing the efficiency of community-driven security (Durumeric et al., 2014). Another example is Mozilla's Firefox browser, which derives its security from a very active community that reviews and perpetually updates new features that make it almost impenetrable, with respect to other browsers (Baker et al., 2021). The collaboration and decentralization in OSS projects can add up to a significant enhancement of security, as the liability of software maintaining is distributed worldwide among the contributors.

Although they are essentially beneficial, community-based security models of OSS have their problems. It can be a daunting task to ensure that the many and diverse contributors are effectively coordinated. The governance structures have to strike a balance between being inclusive and making decisions effectively. There are times when this is easier said than done — a walkover is not always the case (Thompson, 2017).

To begin with, the resources that are available play a big role. Even though many contributors volunteer their resources, resource availability is variable, and this could affect the quality and speed with which security updates are applied (Thompson, 2017). Such variability sometimes results in critical delays in fixing security vulnerabilities.

Contributors also have different agendas and priorities. This diversity sometimes interferes with the prompt and efficient remediation of security issues. The processes through which these perspectives could be harmonized would, therefore, necessitate effective leadership and clear communication (Thompson, 2017).

With regard to more specific challenges in bug bounty program (BBP) implementation, Malladi and Subramanian (2020) mention the presence of clear guidelines and effective communication, as well as appropriate incentives, as requirements for maintaining the quality of submissions at a high level while motivating researchers. Should these components be missing in the implementation of BBPs, their effectiveness will be reduced, with lower-quality submissions and decreased motivation among security researchers being the most likely of outcomes.

Community-based security models and wide peer reviews are essential to the security of open-source software. In conclusion, these collaborative efforts enhance the identification and mitigation of security vulnerabilities.

4.2 Balancing Transparency with Security

Balancing transparency with security in open-source software (OSS) development is a critical yet challenging task. Transparency is one of the fundamental philosophies behind OSS; it gives the possibility for any person to scrutinize, modify, and improve the source code. However, this also means that even potential security holes become visible to malicious actors. Therefore, a trade-off should be made, maximizing the benefits of transparency while minimizing security risks.

One of the greatest challenges of the openness inherent in OSS pertains to vulnerability exposure. With open source and open access to the source code, the door is open for developers and attackers. Such widespread access can lead to security flaws discovered by attackers, which may then be exploited before they are ever fixed. As Cadariu et al. (2015) assert, tracking known security vulnerabilities in proprietary systems is challenging enough; for open-source systems, this task becomes even harder due to the availability of the source code.

Another challenge is posed by potentially insufficiently reviewed contributions. While the openness of OSS enables contributions from a broad group of participants, the other side of the coin is the fact that a broad range of contributions might not be equally well scrutinized. This can be the reason for the genesis of security vulnerabilities if not well scrutinized by experienced developers. Linåker and Regnell (2020) note the necessity of balancing the contributions and, therefore, seeing to it that all the changes within the code pass through the most solid review process, thereby preserving security.

However, transparency bears significant security-related benefits. Most important among the benefits are those that leverage the use of a large and diverse community of developers to pinpoint and then fix the vulnerabilities. Von Krogh and Von Hippel (2006) note that the collaborative nature of OSS means that security problems are identified and then fixed fast due to transparency since the code faces very many eyes. Besides, transparency results in trust and accountability. The users and developers can verify the security measures realized in the software – no backdoors, no malicious code. In the long run, such openness can create a sound security culture within the OSS community – the culture that developers will treat security as their priority in contribution to development. Boulanger (2005) offers an explanation that transparency can enhance software reliability and security through continuous peer review and feedback.

To balance transparency and security without compromising either principle, OSS projects can adopt various strategies. One such strategy is Bug Bounty Programs (BBPs). As defined by Malladi and Subramanian (2020), BBPs are the security programs that offer external security researchers incentives for finding out vulnerabilities and reporting them. they have proven to be very useful, though quite

costly, methods of leveraging the broader security community to find and fix vulnerabilities; hence, employing the use heightens the levels of security of the software.

Another approach is the enforcement of strict code review practices. According to Thompson (2017), the current practices of code review are, in fact, a must for the security of OSS projects. Adequate code review by experienced developers will help OSS projects minimize the likelihood of a security hole being introduced by a code commit. Another measure is the use of automated security tools that can balance the transparency of the code against security. It will continuously scan the codebase for security holes and ensure that any new contribution meets the necessary requirements for security. Such an approach provides the ability to identify and prevent risks in real time. Dialectical methods are proposed to further strengthen the security postures of OSS projects by Weir et al. (2020), where the developers are challenged on their security assumptions.

4.3 Comparative Advantages and Disadvantages of Open Source Software

Advantages
Transparency and Trust: Perhaps the greatest advantage of OSS is its high level of transparency. The source code is available to all for inspection, modification, and improvement. In this way, confidence in and trust among users is created, especially due to the ability of users to check the implemented safety precautions independently. According to Boulanger (2005), the transparency of OSS opens it up for peer review and continuation feedback, making it possible to enhance reliability and security (Boulanger, 2005).

Community Support and Collaboration: A large, varied community of contributors takes part in the development of OSS and its maintenance. Thus, security issues are detected and resolved quite rapidly. Von Krogh and Von Hippel (2006) point out that the collaborative nature of OSS leads to a broad accumulation of knowledge and innovation, which can significantly enhance software quality and security (Von Krogh & Von Hippel, 2006). As Bettenburg et al. (2015) have highlighted, working with a vibrant community of users or developers is an important success factor to innovate technology based on market need (Bettenburg et al., 2015).

Cost-Effectiveness: OSS is either free or less costly than proprietary software. This cost competitiveness is an appealing choice for those with small budgets, whether they are individuals, startups, or enterprises. The small outlay cost is usually transferred into substantial savings over time. Dhir and Dhir (2017) refer to the fact that adoption of OSS enables cost savings in the total cost of ownership because there are no direct costs for purchasing licenses, and updates and upgrades are usually available free of charge (Dhir & Dhir, 2017).

Flexibility and Adaptability: OSS users can modify source code in a way suitable for their needs. It provides an opportunity for organizations to receive highly specific software results fitting their requirements in the best way. Also, this leads to more efficient and effective solutions. Linåker and Regnell (2020) write that "The flexibility to adapt and change OSS can bring competitive advantages in terms of adaptability and innovativeness" (Linåker & Regnell, 2020).

Disadvantages

Security Risks Due to Transparency: Transparency is good, but on the other hand, it can be a source of security problems. OSS is open to everyone, which means that anyone can see potential weaknesses, including those who might use these weaknesses for malicious purposes. As discussed by Cadariu et al. (2015), "The openness of OSS is also associated with a greater risk of security vulnerabilities being uncovered and exploited" (Cadariu et al., 2015).

Lack of Dedicated Support: Unlike proprietary software, OSS does not include dedicated support from the vendor. It relies upon community support. Although the community is often very responsive, there are cases when critical issues should be resolved promptly, and the absence of formal support is a disadvantage. Organizations that rely on OSS to develop their mission-critical applications face this challenge. Dhir and Dhir (2017) argue that "Though community support may be helpful, the lack of ensured support may be risky for organizations" (Dhir & Dhir, 2017).

Quality and Consistency: The quality of OSS varies significantly. This depends on the contributors and the governance model that the project uses. Sometimes, OSS projects do not have proper control processes for quality, and their codebase is often inconsistent. According to Thompson (2017), "Although many OSS projects have great code review practices, others may not, and code review may or may not be strictly enforced, which affects the quality and security of the software" (Thompson, 2017).

Dependency Management: A number of dependencies on other open source components might get added, causing additional vulnerabilities. Updating and managing these dependencies is really a very hectic and complicated task. Malladi and Subramanian (2020) also stress the necessity of effective dependency management for minimizing security risks in OSS projects (Malladi & Subramanian, 2020).

5. Key Findings and Future Trends

5.1 Key Findings

The analysis shows that security by design, least privilege, regular risk analysis, secure coding practices, and continuous testing are crucial principles for strong software security, regardless of the software development model. These principles help to prevent weaknesses and protect systems from cyberattacks in the development of software (Glasauer et al., 2024).

Proprietary software usually requires strong internal security practices and proactive risk management, and vendors need to control their code tightly and provide updates in time to mitigate security risks. Proprietary software has the advantages of dedicated vendor support and legal accountability, which may lead to enhanced security but also to dependencies on vendor responsiveness. In contrast, OSS benefits from transparency and extensive peer review, which form the basis of its security model. Community-based practices and collaboration are supposed to help highlight and fix security issues at the earliest opportunity. However, openness can also place OSS at security risks if not effectively managed. Balancing transparency and security is very critical, and this requires effective governance to coordinate contributions and set the high standards for security (Linåker and Regnell, 2020).

Li et al. (2011) state that the importance of user motivation in OSS adoption is never overlooked. It is found that intrinsic motivations such as the desire for knowledge have a positive impact and drive the usage and participation in OSS projects. Extrinsic factors, like the perceived importance and value of the features of OSS, have a significant effect on the adoption and contribution of OSS. This shows the importance of fostering a motivated community to maintain robust security in OSS.

This is confirmed by Piva et al. (2012), which states that cooperation with the OSS community has a positive effect on the innovation performance of entrepreneurial ventures. Based on their analysis, they indicated that ventures collaborating with the OSS community had a better innovation performance than their non-collaborating peers. Such finding is important, suggesting that active participation in OSS adds to security enhancement via community efforts, while fostering innovation, which may mean a competitive advantage.

Both proprietary and open-source software have their unique security strengths and weaknesses. The proprietary software usually contains a comprehensive vendor support mechanism, ensuring timely responses to security incidents (Boulanger, 2005). Normally, it will have more well-integrated security features that operate in a more integrated manner, with less complexity and potentially fewer vulnerabilities (Temizkan et al., 2012).

However, due to its closed source nature, it is difficult to carry out independent security audits (Cadariu et al., 2015). Also, end-users are at the mercy of the vendor for updates and patches, which may be a problem if the vendor is slow to release them due to the commoditized nature of the software (Weir, Rashid, and Noble, 2020).

In contrast, open-source software benefits from transparency; therefore, it makes large-scale peer review possible, with many security problems being identified and solved quickly (Boulanger, 2005). This high level of openness can enhance OSS security and reliability, in that its use in practice benefits from constant feedback and collaboration from the OSS community at large (Von Krogh and Von Hippel, 2006).
However, the same transparency could expose it to malicious actors who will exploit any visible weaknesses (Cadariu et al., 2015). Furthermore, though community support can be strong, it is not always assured, and severe issues will not always get immediate attention (Dhir and Dhir, 2017).

The need for consistent security standards rests on the dynamic environment of security that the open-source model depends on, through community contribution and peer review for vulnerabilities. But this is possible only with effective coordination and governance. In contrast, proprietary software can be streamlined regarding security processes because of the use of dedicated internal teams and vendor accountability. Broad community input, however, can be lacking due to its nature of multiple vendors and open source (Thompson, 2017).

In a nutshell, the analysis highlights variances in security practices and provides for differences in the security challenges among proprietary and open-source software. While this gives good results for vendor support and built-in security features, a serious shortcoming arises: challenges in the area of a lack of transparency and dependence on the vendor. For OSS, the features of community collaboration and transparency pay off, and risk has to be taken care of delicately.

5.2 Future Trends in Software Security

Moving forward, a few trends will likely define the future of software security. Among them is the integration of the cutting-edge technology of artificial intelligence (AI) and machine learning (ML) to elevate security measures. It is evident that early vulnerability detection and the automation of responses to potential security threats will enhance the general security posture of both proprietary and open-source software (Smith, 2020).

Second, and in all likelihood, the trend will see more security requirements put into policy. The General Data Protection Regulation (GDPR) in Europe is just the start of policies that are likely to emerge in other regions with higher data protection and software security standards. Security regulations will become dynamic and will lead

organizations to accept newer and more rigid security frameworks to ensure continuous improvement in security practices (Gordon et al., 2019).

Regarding OSS, the motivation and commitment of the developer community will remain pivotal. From the results of Li et al. (2011), it is implied that having a motivated and active community of developers will be crucial in maintaining and advancing security in OSS. This can be achieved by organizing hackathons, running bug bounty programs, and providing recognition to contributors. It would be vital that contributors feel valued and motivated to maintain the collaborative security work upon which OSS relies.

Third, according to Piva et al. (2012), OSS collaboration will not only enhance security but is also likely to drive innovation. When more entrepreneurial ventures realize the dual benefit of both improved security and increased innovation performance, the trend of integrating OSS collaboration into business strategy will grow.

In conclusion, proprietary and open-source software will need to align these trends and challenges to result in solid security. Continuous learning, community involvement, and technological innovation will be crucial strategies in responding to future security threats.

5.3 Implications for the Future

Security is an enormous factor that dictates the choice between proprietary and open-source software. Proprietary software provides some level of vendor support and responsibility, which is an important case for those organizations where the security of a system is a top priority. Brunetti et al. (2014) also indicate that proprietary software is structured in a way to support users with structured support and proper updates, which are useful in supporting their software security. Usually, the vendor is supposed to make available security patches and updates in a timely manner, and as a result, the software remains secure from new threats. On the other hand, hidden vulnerabilities may exist in proprietary software, and it relies on vendor-dictated updates. Organizations, therefore, have to consider these issues and balance them with the organization's specific security requirements before making the decision to adopt the software. Also, the cost of licensing proprietary software increases its total cost of ownership (Brunetti et al., 2014).

OSS also has the advantage of source code transparency and a community-based security model. As stated by Brunetti et al. (2014), since the source code is open to everyone, issues related to security could be easily identified and fixed with the help of wide peer reviewing. But, on the negative side, this transparency may actually expose vulnerabilities if not handled with the proper security measures. Transparency and security in OSS projects demand effective governance and

coordination. Inherently, OSS communities quickly identify and patch detected vulnerabilities, but it is dependent on contributors acting in accordance with best practice regarding secure coding and vulnerability management. Notably, there has been a growing acceptance of OSS among businesses and governments. Cost savings, flexibility, and the ability to collaboratively innovate are among these benefits. Besides, large tech firms, such as Google, Microsoft, and IBM, are increasingly not only adopting OSS solutions but also contributing to their development, which further enhances their security by working closely with the community. This trend indicates a move towards a more collaborative approach in the development of software, where the responsibility for security is shared by all stakeholders.

The future will likely hold a mix of proprietary and open-source models, with hybrid solutions becoming more common. These hybrid models can bring the best of both approaches to the table, providing the flexibility and innovation that the OSS can offer with the critical security functions found in proprietary solutions. According to Brunetti et al. (2014), organizations will likely take up hybrid solutions that provide both a safe and balanced software environment, enabling them to achieve the best of both worlds to meet their organizational security and operational requirements. For instance, organizations may use OS components in areas of their systems where failures are not critical as a means to access fast innovation and reduced cost. On the other hand, they could use proprietary software for other, more critical functions that require tight security and continuous support. The flexibility accorded to the organization to choose each of these components based on the impact of their failure will ultimately make these organizations have the best of both worlds, in terms of security, as they will have secure and optimized software environments.
Artificial intelligence and ML are currently trending and will continue to revolutionize software security. Kaur et al. (2023) explained that AI, particularly machine learning and deep learning, can help analyze vast amounts of data for patterns and deviations that point to vulnerabilities or security compromises. This, in turn, results in timely detection and automatic response to threats, significantly enhancing the security of proprietary as well as OSS. AI systems can take the desired action—execution of response protocols, isolation of the affected component, patch application, and restoration of the system to a secure state—completely autonomously. This automatic response mechanism is imperative to reduce damage from cyber incidents and ensure organizational continuity. Apart from that, AI can be used in building nuanced threat models that can predict and prevent cyber attacks by understanding the TTPs of cyber adversaries.

AI can also greatly improve risk management practices. By analyzing historical data and predicting potential future threats, AI supports proactive risk management strategies. Kaur et al. (2023) suggest that AI can provide valuable insights into vulnerabilities, helping organizations prioritize security investments based on the likelihood and impact of potential threats. This proactive approach can significantly

reduce the chances of successful attacks by addressing vulnerabilities before they are exploited. However, adopting AI in cybersecurity comes with challenges. Kaur et al. (2023) note that the quality and availability of training data, robustness, and interpretability of AI models are significant concerns. Addressing these challenges requires developing methods for generating high-quality training data, improving model robustness, and enhancing interpretability to ensure AI systems are effective and reliable in real-world applications. Ongoing research into ethical considerations and potential biases that AI systems may introduce into cybersecurity practices is also necessary.

Regulation and policy play crucial roles in shaping software security practices and the broader industry landscape. Compliance with regulations like the General Data Protection Regulation (GDPR) demands robust security measures, requiring organizations to meet strict data protection standards. Gordon et al. (2019) emphasize that as regulations grow in scope and complexity, they will drive best practices in software security across the industry. Organizations must stay informed and adaptable to ensure their security strategies comply with evolving regulatory requirements, maintaining robust security in software development and deployment. The regulatory landscape is continuously evolving, with new laws and guidelines being introduced to address emerging threats and technologies. This requires organizations to adopt a dynamic approach to compliance, integrating regulatory requirements into their overall security strategies and ensuring they can quickly adapt to new regulations as they arise.

In conclusion, the future of software security is dependent on an intricate web of proprietary and open-source models, technological strides in AI and ML, and maturation of a regulatory landscape. It is imperative that organizations remain informed and flexible, picking best practices from both proprietary and open-source software, leveraging AI to strengthen security measures in place, and ensuring regulatory compliance to reap the benefits of a robust security strategy while safeguarding software development and deployment processes from emerging threats. The really big game changer in terms of AI in security, and in particular with respect to cyber, is bringing the detection and mitigation of threats to a completely new level, far beyond where it has ever been. But all of this will require a collaborative effort of all involved parties, from the developers and security practitioners to regulators and the general community, to ensure that the technologies are developed and used responsibly and in a beneficial way.

6. Conclusion

This seminar paper has examined the security issues of both proprietary and open-source software. These have been done through the presentation of a rather extensive literature review and analysis, which will bring out the fundamental principles, strategies, and challenges of the given area under consideration. The results show that proprietary and open-source software both have their advantages and disadvantages with respect to security.

Proprietary software enjoys support from the vendor, who is accountable and can bring more stringent security controls to address unique needs. This architecture introduces a single source of updating and security patching, hence instills some sense of reliability and trust to users. Some of the challenges of proprietary software include a large number of hidden vulnerabilities due to the fact that the source code is not open for public scrutiny, which may therefore mean that there are many more vulnerabilities that may not have been identified yet. Users are also dependent on the vendor's timing for updates and patches, and this might sometimes take longer, hence leaving systems open for longer periods.

On the other hand, open-source software does well in matters of transparency and community-based security models. The open nature of the source code allows for broad peer review and collaboration, which can quickly identify and patch security vulnerabilities. As such, this kind of collaborative approach has the potential to pool in the expertise of a lot of people in the global community, bringing about more innovative and effective security solutions. However, the transparency attributed to it in most of its operations may also make it a liability to potential threats if not effectively managed. Governing and coordinating such is paramount in achieving the required balance for transparency while ensuring security provisions in vetting and management to prevent malicious code entries.

The analysis concludes that both models will coexist in the software ecosystem, with the increase in hybrid approaches that present the best of each model. The adoption increase in open-source software by organizations and government is proof of acceptance growth and the benefits of saving costs, flexibility, and collaborative innovation. Companies such as Google, Microsoft, IBM contribute effectively to open-source projects, improving its security and resilience via community interaction (Li et al., 2011). Proprietary software is still relevant today due to its reliability; it also possesses integrated security features and structured support, especially for organizations that have guaranteed support and accountability.

In the upcoming years the use of emerging technologies like artificial intelligence (AI) and machine learning (ML) will significantly enhance the security of software. AI and ML will supplement the early detection of vulnerability and automate the response to potential threats. For open-source software, this could, in an extended sense, also

generally improve security. Such technologies can analyze data in enormous volumes to identify patterns and anomalies, which may lead to suspicion of security problems.Clearly, the integration of AI and ML in the way security is undertaken will provide a monumental transformation in the maintenance of security in the software industry.

Future work should work on how AI and ML can enhance security measures to a further extent and the dynamics of moving in between proprietary and open source models. The role that AI can take on in this field is particularly intriguing and may offer major advances in the security of software. Furthermore, it will be pertinent to observe how organizations are managing shifts between software models and what this means for their security practice. This area of research will provide valuable insights into the benefits and challenges of hybrid models, where one is able to choose the best from both proprietary and open-source software to create a secure and efficient system.

Overall, both proprietary and open-source software will play important roles in the future of software security. Each model presents distinct strengths and faces unique challenges, fostering a diverse and innovative software landscape. Stay informed and adaptable, and you will get a resilient and stout security posture for any organization with its software development and deployment best practices, harnessing the best from both worlds to meet the needs in the best possible manner and the security requirements.

References

Baker, M., Davidson, A., Munyua, A., & Kak, A. (2021). Digital ID: A White Paper from Mozilla. Retrieved from https://www.mozilla.org/en-US/foundation/reimagine-open/

Bettenburg, N., Hassan, A. E., Adams, B., & German, D. M. (2015). Management of community contributions: A case study on the Android and Linux software ecosystems. *Empirical Software Engineering, 20(5)*, 252-289. https://doi.org/10.1007/s10664-013-9284-6

Bojanova, I., Galhardo, C.E.C., & Bojanova, I. (2023). Bug, Fault, Error, or Weakness: Demystifying Software Security Vulnerabilities. *IT Professional, 25(1)*, 7–12. https://doi.org/10.1109/MITP.2023.3238631

Boulanger, A. (2005). Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal*, 44(2), 239-248. https://doi.org/10.1147/sj.442.0239

Brunetti, G., Feld, T., Heuser, L., Schnitter, J., & Webel, C. (2014). *Future Business Software: Current Trends in Business Software Development*. Cham: Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-04144-5

Cadariu, M., Bouwers, E., Visser, J., & van Deursen, A. (2015). Tracking Known Security Vulnerabilities in Proprietary Software Systems. In P*roceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering* (pp. 516-520).

Dhir, S., & Dhir, S. (2017). Adoption of Open-Source Software versus Proprietary Software: An Exploratory Study. *Strategic Change, 26(3)*, 287-299. https://doi.org/10.1002/jsc.2141

Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., & Foster, S. (2014). The Matter of Heartbleed. *Proceedings of the 2014 Conference on Internet Measurement Conference*, 475-488. https://doi.org/10.1145/2663716.2663755

Glasauer, C., Maurer, L., Spreitzer, C., & Alexandrowicz, R.W. (2024). Development & psychometrics of the SOLID-S – An inventory assessing software security culture in software development companies. *Computers & Security, 140,* pp. x-x. https://doi.org/10.1016/j.cose.2024.103753

Kaur, R., Gabrijelčič, D., & Klobučar, T. (2023). Artificial intelligence for cybersecurity: Literature review and future research directions. *Information Fusion, 97*, 101804. https://doi.org/10.1016/j.inffus.2023.101804

Kilamo, T., Hammouda, I., Mikkonen, T., & Aaltonen, T. (2012). From proprietary to open source—Growing an open source ecosystem. *Journal of Systems and Software, 85*(7), 1467-1478. https://doi.org/10.1016/j.jss.2011.06.071

Linåker, J., & Regnell, B. (2020). What to share, when, and where: balancing the objectives and complexities of open source software contributions. *Empirical Software Engineering: an International Journal, 25(5)*, 3799–3840. https://doi.org/10.1007/s10664-020-09855-2

Magnanini, F., Ferretti, L., & Colajanni, M. (2022). Scalable, Confidential and Survivable Software Updates. *IEEE Transactions on Parallel and Distributed Systems, 33(1)*, 176-191. https://doi.org/10.1109/TPDS.2021.3090330

Malladi, S. S., & Subramanian, H. C. (2020). Bug Bounty Programs for Cybersecurity: Practices, Issues, and Recommendations. *IEEE Software, 37(1)*, 31–39. https://doi.org/10.1109/MS.2018.2880508

Microsoft. (2024). Security Update Guide. Retrieved from https://msrc.microsoft.com/update-guide/

Pinto, G., Steinmacher, I., Dias, L. F., & Gerosa, M. (2018). On the challenges of open-sourcing proprietary software projects. *Empirical Software Engineering, 23*(6), 3221–3247. https://doi.org/10.1007/s10664-018-9609-6

Raymond, E. (2005). The Cathedral and the Bazaar. *First Monday*. https://doi.org/10.5210/fm.v0i0.1472

Temizkan, O., Kumar, R. L., Park, S., & Subramaniam, C. (2012). Patch Release Behaviors of Software Vendors in Response to Vulnerabilities: An Empirical Analysis. *Journal of Management Information Systems, 28(4)*, 305-338. https://doi.org/10.2753/MIS0742-1222280411

Temizkan, O., Park, S., & Saydam, C. (2017). Software Diversity for Improved Network Security: Optimal Distribution of Software-Based Shared Vulnerabilities. *Information Systems Research, 28(4)*, 828–849. https://doi.org/10.1287/isre.2017.0722

Thompson, C. (2017). Large-Scale Analysis of Modern Code Review Practices and Software Security in Open Source Software. *eScholarship, University of Californ*ia. Retrieved from https://escholarship.org/uc/item/0mj0k0zp

Von Krogh, G., & Von Hippel, E. (2006). The Promise of Research on Open Source Software. *Management Science*, 52(7), 975–983. https://doi.org/10.1287/mnsc.1060.0560

Weir, C., Rashid, A., & Noble, J. (2020). Challenging Software Developers: Dialectic as a Foundation for Security Assurance Techniques. *Journal of Cybersecurity, 6(1)*, tyaa007. https://doi.org/10.1093/cybsec/tyaa007