An open source software: understanding the benefits and concepts

Science, Computer Science

Abstract

The greatest challenge for network and system administrators is information security. Information security depends mainly on the software used and the administrative procedures employed for operating systems, intrusion detection, network monitoring, company mail, and antivirus protection just to mention a few. This paper highlights open source software security in an effort to contribute to the current situation in respect to comprehension of security systems. This paper also seeks to explore whether open source software will have serious repercussions for corporates or companies utilizing open source software and the kind of trust that exists between users and the software. Besides, it will provide helpful insights for the developers as well. This paper will advance the state of comprehension in regards to source philosophies and examine vulnerabilities resulting from source visibility differences. It is in the intent of this essay to propose an answer to a modern and relevant question: Does open source software enable development of a more secure code.

Initial Systems Analysis

Overview of Open Source Software

This part presents two opposing arguments on the security of open source software. The primary intention is to summarize the argument of each side of the opposing views, regardless of whether they can be substantiated. Other sections will discuss claims with solid facts and data.

Is Open Source More Secure

With open source, individuals who wish to have a glimpse of the source code for any section of this project can do so. Both experts and novices, with their many eyes, may spot bugs and vulnerabilities in the source code. Open source software is subject to security analysis by any software developer and “ in-house” review by programmers tied to it. It is also indispensable to note that it is subject to spontaneous review by any other person around the world. Some will argue that more code examinations are likely to take place on open source projects compared to others considering that, developers responsible will subject it a certain degree of test and the outside sources will subject it to a certain degree of tests. Normally, this happens when big companies intend to sell products on open source projects. In such a case, they have interests in the security of the project as if it were proprietary. Finally, many cite security concerns as a primary reason for not opening the source code to the public. However, there seems to be little conclusive proof to support the argument.

Self-assurance within Anonymity

It is indispensable to note that if I hide my code from my attackers, it will be hard to establish the presence of susceptibilities in the software. Negative results come along at the source level only by offering visibility. It is also necessary to understand that writing good software is difficult and complex, and writing non-trivial, perfect software is unfeasible. Therefore, if writing software is difficult and complex, and reverse engineering is somewhat difficult, then why sacrifice protection by anonymity? Certainly, the practice is not reciprocally beneficial; it simply gives an attacker the blueprints of your development or construction. As security flaws identification tools improve, it benefits the attackers as well. They can identify susceptibilities before deployment of fixes to the clients.

Being slow or being able to prevent propagation of a source will enable an organization to capitalize on the gains that we are fully responsible for, and for that matter, disclosing your plans to your attackers and competitors is not an ideal security strategy.

Habitually Deceiving Data

Deciding on this argument would necessitate critical examination of attacks against or reported susceptibilities in " comparable open source and proprietary software products" and select a winner. However, it is important to note that several variables exist besides development philosophy, and these variables influence such data. Among the variables that influence the metrics include, deployed base size, provided features, code size, business decisions, the appropriateness of client maintenance and patching, and attackers' preferences. It is in the interest of all attackers to exploit susceptibilities in systems as much as they can. They may decide to exploit susceptibilities in popular operating systems. Likewise, it is evident that security organizations and " white hat" puts little effort towards the identification of vulnerabilities. Therefore, from this information, it is hard to conclude that software with more vulnerabilities or attacks is more or less secure than any other software.

Support

At one point, susceptibility in the Linux operating system was found, reported, and fixed. Upon releasing the security update, the adversaries learned the exact nature of the weakness in the system and conjured a comparable gap might exist on other information resource managers and applications. The group was successful in attacking and exploiting a similar susceptibility that existed in a closed source program, Windows NT. Therefore, it is evident that the existence of weakness in an open source program can divulge the flaws of both itself and any isometric application. This can be a clear indication that the closed source system and open source system are intertwined in a manner that inhibits investigations. The intertwining makes it different to separate the two systems for analysis considering that they do not operate independently and separately. The construction and engineering standards have many similarities. Additionally, evidence might suggest that it is possible to leverage the similarities of the two systems by simply making observations on the various classes of susceptibilities in a platform-independent manner. Further, it is indispensable to consider the implications of revealing patches, which end up divulging high-level operational weaknesses shared by similar platforms providing adversaries with cross-platform day zero attack (Wheeler, 2017).

It is necessary to comprehend the extensive overlap in the development of primary operating systems discussed here and expect timely identification, reporting, and fixing of the weaknesses that ought to be overlap in the susceptibility space. This will create a platform where, major operating systems can share findings thus, be able to provide mutually beneficial security improvements. For this study, it is indispensable to observe that applications and operating systems, in general, overlap a great deal in the used design patterns (Open-Source Army, 2015).

Vulnerabilities

Wheeler (2017) argues that not all attacks targeting resource managers rely on source code weaknesses. He adds that not all attacks relate to the security of open source software. Many of the attacks are highly dependent on humankind to offer assistance. Therefore, it is essential to note other types of cyber security attacks that if can breach even perfectly secure systems. Attacks can either source-dependent or source–independent attacks.

Source-Dependent Attacks

Before, according to research, open source software was not a primary target for attackers. This was mainly due to the predominance of Windows platform. However, it is evident now that the number of attacks targeting the Linux operating system is rising rapidly. This could be because Linux operating system is becoming more popular by the day. Currently, more than 100 viruses are targeting Linux, and the number is continuously rising. Besides viruses, other attacks are targeting Linux include Trojan, worms, rootkits, and denial of service (DOS) attacks just to mention a few (Broersma, 2002).

The attacker’s target is primarily on Linux operating system servers. Most of these attacks occur due to the popularity of Linux. Poor administration of

Linux servers seems to be the challenge contributing to its vulnerability. Researchers attribute the attacks to poor system administration and not to its inherent weaknesses, and especially those that exist in the software. Additionally, the researchers cite complexities associated with administering the Linux servers as another reason for using Windows server and not Linux (Chen et al., 2011).

System weaknesses, form another source of security concerns. In this case is complexity associated with administering Linux servers. It is evident that poor administration results in unimaginable security concerns. Different distribution of Linux leaves many open ports, which in turn increases the susceptibility of the application to attacks. Just like Linux, a default Windows installation might leave multiple open ports. However, due to good and easy administration, it becomes simpler to close unnecessary services compared to Linux. Therefore, systems that easier to administer becomes the preferred option, but is important to remember that it does not become intrinsic to source distribution viewpoints, and so is beyond this work’s degree of coverage. Notable as well, is the need to understand that ease of administration does not imply that one operating system (open or closed) is more or less secure in relation to the other. A knowledgeable system and network administrator should have knowledge of how to reduce susceptibilities within a network to avoid attacks (McLean, 2012).

McLean (2012) further state that, administration of Windows and Linux operating systems may vary in many ways. For instance, administration of Linux is in a way whereby, it runs as a typical account using “ sudo” or “

setuid" to execute instructions as root, but it does not run as " root." On the other hand, Windows operating system runs using administrator mode. In as much as this might not be an open v. closed source software issue, it helps in explaining why attacks on Windows are more prevalent compared to Linux. Typically, viruses targeting Linux affect EFL (Executable and Linkable Format) files, associated with Linux operating systems. Other attacks make use of shell scripts, which happen to be compatible across systems and distributions, allowing them to spread extensively compared to binary files. Additionally, the complexity of viruses targeting Linux has also increased. For instance, the W32/Etap. d, which is a polymorphic virus, is one of Linux viruses that are hard to detect. This polymorphic virus is a variant virus that can infect " Windows portable executable files" besides Linux files (Ven, Verelst, & Mannaert, 2008).

The sharing of open source code frequently happens between various open source software, for instance, different Linux applications. Some weaknesses exist on several open source software like Apache. Linux worm, also referred to as Slapper worm, exploits vulnerabilities in an Open SSL library to infect " Apache Web servers." On top of that, Linux/Adore worm is another type of worm that infects Linux servers. It exploits root access vulnerabilities on Linux servers through a random port scan. A single vulnerability can result in infection of multiple operating systems (Ven & Mannaert, 2008).

Because open source software come in different forms, for example, various distributions of Linux, it becomes hard to synchronize a simultaneous patch for flavors. Most attacks occur once vulnerabilities and patch are publicized.

However, not all flavors have solutions available. Besides, some distributions remain susceptible to attackers in the process. This forms one of the disadvantages associated with the distributed nature of open source software. However, it is correct to argue that the simultaneous distribution of open software has made it difficult for attackers to carry attacks. Thus, there are both negative and positive aspects associated with distribution (Chen et al., 2011).

Buffer Overflow

Automated tools that perform static code analysis can reveal buffer overflow. The attack focuses on symbols and structure of a program. It is easy to detect the attack from any visible body of source code. Any open source software with buffer overflows is susceptible to adversaries. In this case, the attackers can make use of automated tools to carry out analysis. However, the automated tools used by enemies are available during software's development life cycle. Modern development settings automatically warn programmers in instances where they used legacy code constructs that will possibly experience buffer overflows (Ven, Verelst, & Mannaert, 2008). Besides, " newer programming languages" prevent " memory structure declarations" that permit developers to upend memory spaces. The villainous Blaster worm that came into being in 2003 is a perfect illustration of buffer overflow exploits. The Blaster worm exploited a weakness in the Windows DCOM system. It does not require user participation for it to spread (Chen et al., 2011).

SQL Injection

SQL Injection looks for Web servers that promote database services. SQL Injection will then look for an interface, for example, username and password, which will accept client's side input. This will contribute greatly to compromise the SQL database it is running on it. It operates in the same manner as buffer overflow. SQL injection exploits vulnerabilities that arise due to failure to authenticate and correctly input login details. It is a form of source-dependent attack (Wheeler, 2017).

Patching

The infamous zero attack is a product of reverse engineering, recently launched software. The patch, in this case, is a blueprint that outlines real vulnerabilities that computers are prone to. The " day zero" moniker refers to the finest time to exploit the greatest number of computers, especially after the revelation of a patch. Remember, exploits become less significant with continuous application of patches to many systems. Attackers prepare for this kind of attack by developing tools that will enable them to generate working exploit automatically using a patch as their input. Day zero attacks are code level attack, and a shortcoming to open source software. Interestingly, closed source software often suffer the same attack besides closed source attack. Reverse engineering the code between the patched and unpatched files gives the attacker an opportunity to view the original security flaw. This class of attack affects both open and closed sources (Schweik & English, 2012).

Source-Independent Attacks

According to Open-Source Army (2015), source-Independent attacks involve large sets of attacks, which do not rely on security flaws at all. None is safe between open source and proprietary software. In this section of the study discusses some source-independent attacks. Additionally, it seeks to demonstrate that despite programs construction philosophies, operating systems are equally vulnerable.

Users

Spyware, data theft, viruses, and other forms of attack require the input of the users. Most of these attacks depend heavily on information provided by system users who execute programs and provide data to unauthorized parties. Unauthorized parties, in this case, are parties that should have no access to certain forms of data. The attackers, through their attacks mostly take advantage of the ignorance of the users. For instances, some users will click " Yes" dialog on pop-ups to make them disappear. Users also transfer viruses running infected applications and by using floppy disks. Such viruses do not rely on vulnerabilities in the systems (Schweik & English, 2012).

If a user with root or administrator privileges logs into a system, applications executed during this session will also run with similar privileges. This can include, for example, installation of drivers and patch the kernel among others. Other attacks depend on the input of the user to exploit weaknesses, like the ones that allow elevation of privileges. At some point, attackers take advantage of unprivileged users to exploit security flaws. The attacks use unprivileged users to launch applications that execute the exploit. This happens mostly in instances where security flaws are inaccessible via the network. Limited and unprivileged users will assist in execution malicious programs enabling the attackers to elevate the privileges of the accounts (McLean, 2012).

McLean further states common examples of such attacks as attachments sent via email. Adversaries create scripts and “ socially engineered” emails with intentions of drawing the attention of email users. This includes emails that entice email users to open the attachments. “ Love Letter” and “ MyDoom” worms are common examples of attacks. They will only exploit a system only if the user decides to click on a malicious link contained in by these email attachments.

Visceral Incidents

According to Chen et al. (2011), visceral incidents refer to situations where adversaries try as much as they can to log into a system. Enemies carry out the activity mostly as a root or administrator repeatedly until they are successful. Its execution does not require any form of vulnerability in the source code. The attacker uses passwords attacks on an operating system. Therefore, open source and closed source software are susceptible to this kind of attack.

Protocol Vulnerabilities

In Schweik & English (2012) argument, some protocols have weaknesses that are intrinsic in the protocol and its dependencies. Software that supports a protocol implements similar protocols. Under such a case, software using similar protocol will be equally susceptible. A good example of this is the Kerberos, in other words, “ Man-in-the-Middle,” vulnerability. Any execution of “ man-in-the-Middle” could exploit no matter the availability of source code. Such protocol susceptibilities tend to be more worrisome considering that, the target set is great since it makes use of various platforms.

Internal Situations and Incursion

McLean points out that this as a type of attack executed by an individual with access. An employee or persons with access can offer access to attackers who will access the system from a remote location. This will offer some form of security and anonymity to adversaries. Additionally, when the attacker has access to the system, the system implementation philosophy does not matter. In internal incursions, sometimes referred to as inside jobs, the main adversaries are the employees.

Attack Regularity

The number of attacks targeting operating systems has grown steadily in the past few years. Computer Emergency Response Team (CERT) has reported over three-hundred thousand attacks since 1990. CERT attributes the rapid increase in cyber attacks to the growing use of automated tools. The use of automated attack tools has resulted in frequent attacks. With CERT’s metrics, it is evident that the frequency of attacks has been increasing steadily over the years. Access to computers and increase in the use of automated attack tools could be the major contributing factors. The factors also support the fact that there has been a steady growth in the number of attacks targeting operating systems. Attackers have more access to computers and automated analysis tools that facilitate attacks (Shin et al., 2011).

Many studies suggest that there have been more attacks targeting Windows compared to Linux. The main reason behind this can be due to the popularity of Windows operating system. In 20014, globally, Linux market share was approximately 3% whereas Mac OS was almost similar to Linux. Windows controlled 94% of the global market making it a popular operating system at the time. Given the data, replacing closed source software with open source software would not have offered a solution. Adversaries would have shifted to the next chief target. Besides, Linux viruses are fewer compared to Windows. Some weaknesses in Debian Linux remain unpatched because of the feeling that the attack was not noteworthy to push them to fix the vulnerabilities in the operating system (Broersma, 2002).

Security Analysis

Vulnerability Discovery

The open source community owns the responsibility of identifying and fixing security flaws that exist in open source software. With the growing support and development of open source software like Linux, by companies like Novell and IBM, many people are getting a job. The responsibility of professionals is to identify and fix security flaws. Therefore, the security of open source software is likely to improve in the new few years. The improvement is the security of open source software will reduce disparities that exist between open source and closed source software (R, 2017).

For closed source software like Microsoft, security processes have become routine. The main goal is to reduce the number of susceptibilities in closed source systems. Companies like Microsoft have security teams. The company’s security processes enable the security team to fix problems reported by Microsoft’s clients or hackers (Chen et al., 2011).

Security Tools

Threat Modeling and Exploit Classification

Threat modeling refers to ways used in identifying and addressing security threats in software. The process’ summary is as follows: Creation of lists of inputs of a program. Second is the creation of a list of all vulnerabilities. These may include external dependencies, like APIs that a programmer has no control and other vulnerabilities that are difficult to eliminate. Classification of susceptibilities is according to STRIDE classification system. STRIDE stands for “ Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elimination of privileges.” Spoofing identity refers to the process of obtaining access to use of other people’s logins. Tampering with data refers to the modification of data without owners consent. Repudiation involves keeping track of activities and persons who performed the activities. Information disclosure is a situation whereby one obtains information without permission to access. Denial of service, on the other hand, refers to attempts that intend to prevent authorized users from accessing or using the system. Elimination of privileges is that situation where underprivileged limited users can obtain higher-level privileges. Microsoft developed the STRIDE model to enable programmers to identify security flaws in the source code (Shin et al., 2011).

Shin et al. further assert that after categorization of vulnerabilities, ways in which inputs can threaten susceptibilities follows. The final stage is the listing of mitigations. If the system is too big, it is advisable first to list all inputs, and then a breakdown of the system into feature and sub-features before performing threat models for each one of them.

Source Code Scanners

Source code scanners exist in both open source and closed source software. They assist in the creation of more secure code by finding common security flaws in source codes and developing new, secure alternative system. Flawfinder, RATS (Rough Auditing Tool for Security), and ITS4 (It's The Software Stupid Source Scanner) are tools that aids in finding problems. Besides, Microsoft Visual Studio 2005, FXCop, and PREfast are other forms of tools that scan C++ code. However, automated source code scanners have limitations. Among the constraints, include the fact that they will never be thorough, like a manual code review. However, it is advisable to use the scanners before carrying out manual audits and reviews. In as much as they are helpful, they might become less relevant in the coming years (Ven, Verelst, & Mannaert, 2011).

Security Reviews

Compared to an open source, the organization of closed source better and involves a lot of work. Companies use threat models and security scan tools on code. Microsoft carries out security tests at least once at every stage of the development lifecycle. During security tests, it is indispensable to review the passcode, fix bugs, and perform tests on security flaws. Open source software being typical and less distributed, it does not require lengthy security processes. Codes scanners, RATS, and ITS4 can work well for open source software (Shin et al., 2011).

Socioeconomic Effects

The presence of source code for open source software has several implications on the ecosystem of such software. The effects include the speed of security and the customization of security.

Accountability

This arises in instances where large corporations are making plans to adopt open source software. Open source software needs somebody to hold accountable. In this case, it needs someone who has the essential resources to address possible problems that might occur. This is indispensable for security concerns. Attacks by worms and viruses are dangerous and expensive to organizations. For instance, when an organization purchases software from Microsoft, it is in their exactions that Microsoft will be accountable. The End User License Agreement does not hold the users of software accountable. Instead, the level of reputation of software gives the developer accountability. If it does not work, it will not be the responsibility of the end user. It will be the responsibility of the developer to fix security flaws. Additionally, when a company purchases Linux from IBM, IBM does not provide accountability because it does not own the software. However, it will support but not fixing bugs and flaws. Additionally, it will not lose reputation because Linux is available through GNU public License (Schweik & English, 2012).

Fix

Microsoft releases updates regularly. Apple, on the other hand, releases security updates on a monthly basis for its product, Mac OS. They notify users as soon as the updates are available. The users will then download then install manually. The biggest problem is that users will have to reboot their computers after installation of a security patch. Rebooting may delay installation. At this stage the machine is vulnerable. Thus, it is necessary to reboot. However, in the case of Linux, rebooting is not necessary. Just like Microsoft, Linux also has automated security updates as well (Ven, & Mannaert, 2008).

Availability

Software vendors ought to validate patches when fixing security issues. The process could take several months or even a year in extreme cases. Besides, there are instances where companies refuse to fix flaws. In open source software, the end users can fix flaws as soon as developers release the patch. In open source software users, consumers can apply security fixes as soon as they need them. Additionally, in Linux, changing the OS is simple and hassle free (Wheeler, 2017).

Custom Software

Open source software allows consumers to make changes to suit their needs. Thus, users can improve their security. Any knowledgeable individual can develop derivative or customized versions of open source software. However, the custom-made software could be having numerous bugs and vulnerabilities due to limited reviews. Additionally, the larger community, which is necessary to determine the existence of flaws has not tested most custom made as well, making it vulnerable to adversaries. However, attackers are likely to go for widely available software (McLean, 2012).

Security Fixes

Once software reaches end-of-life, it will not receive any further support from the vendor. Lack of support from the vendor will expose the business to numerous challenges associated with adversaries. Despite the fact that business needs might force them to use the software, it might be impossible. That is why; many companies prefer updating their decades-old software than migrating to new software. Open source software can be a perfect choice for alleviating such problems (Chen et al., 2011).

Conclusion

Open source allows visibility of its code. From the study, it is evident that visibility does not help in protecting a project from potential threats. Security experts argue that at some point, source level visibility helps in project convergence allowing for stability and security. Additionally, it is also evident that more attacks will focus on companies that offer superior economies of scale. However, this does not imply that close source software is secure considering the number of assaults directed towards the systems. Open source system developers note their software to be more secure to be bearing in mind the frequency of attacks. My study has not been able to demonstrate greater security with anonymity. Thus, is hard to conclude that closed source offers tangible security protection features we can leverage in hunt of a more secure and stable system. Lastly, code leaks are is an issue that is hard to prevent.

Summary of Main Points

Many security attacks do not rely on the source code, so none of open and closed source systems is more or less secure than the other. Open source is a little bit secure bearing in mind the frequency of attacks and assaults. Additionally, open source reinforces its security by involving many people to identify bugs and security flaws. On top of that, it is evident from the research that code visibility does not aid in achieving secure and stable states.

Revisit introduction or tie all ideas together

The interest of this study was neither to dissect corporates ecosystems nor to draw comparisons between different products. Additionally, the study does not seek to argue the most superior system, identify the most secure, or the most economical. The study does not point out the most secure operating system. The main goal is to seek data that will quantitatively and qualitatively confirm or deny the security level of open source software bearing in mind visibility of its code. Subjects peripheral to this study include close source software. The study also recognizes tools that will enhance the security of systems. The study concludes that visibility of codes does not strengthen the security of a system. In addition, none of the open and closed source software is superior that the other.