Examensarbete

1977
HÖGSKOLAN
I SKÖVDE

UTMANINGAR I SÄKERHETSREVISIONER I SYSTEM MED ÖPPEN KÄLLKOD

CHALLENGES IN SECURITY AUDITS IN OPEN SOURCE SYSTEMS

Examensarbete inom huvudområdet
Informationsteknologi IT610G
Grundnivå 22.5 Högskolepoäng
Vårttermin 2019

Pontus Nordberg
a15ponno@student.his.se

Examinator: Jianguo Ding
Handledare: Thomas Fisher

2019-08-31

Abstract

Today there is a heavy integration of information technology in almost every aspect of our lives and there is an increase in computer security that goes with it. To ensure this security, and that policies and procedures within an organisations related to this security are enforced; security audits are conducted.

At the same time, use of open source software is also becoming increasingly common, becoming more a fact of life rather than an option. With these two trends in mind, this study analyses a selection of scientific literature on the topic and identifies the unique challenges a security audit in an open source environment faces, and aims to contribute on how to help alleviate the challenges.

The study was performed in the form of a literature review, where the comparison and analysis revealed interesting information regarding the open source specific challenges, including both technical issues as well as challenges stemming from people's perception and handling of open source software today.

The answer to the question "*What are the challenges when conducting security audits for open source systems and how can they be alleviated?"* shows the main challenges to be too much trust is put in unverified binaries. The report offers suggestions and ideas on how to implement solutions in order to help diminish this challenge through the use and integration of *Reproducible Builds*, answering the second part of the question.

Keywords: Security Audit, Open Source Software, Reproducible Builds

Table of Contents

1. Introduction

With heavy integration of information technology in almost every aspect of our lives today, data in information systems within organisations is a valuable resource which needs to be kept safe just as much as any other (perhaps even more so). The three main aspects of the *CIA* security model, *confidentiality*, *availability* and *integrity*, are the cornerstones which should be followed to ensure safekeeping of this data, which can become a big risk if one of these aspects are neglected (Pereira & Santos, 2010).

To ensure that this model is being followed, with regards to both laws, as well as policies and procedures within the organisation, check-ups need to be made to ensure that the reality matches such policies and that both physical and logical security is ensured. This is done by security audits, which are systematic reviews of the information security in an organisation, such as a company or government agency. Audits may be performed with different levels of scope, width, and objectives. They can also be done both during normal operation of a system or in other situations, such as during development or implementation of software (Perl et al, 2015).

The software which is then audited can be either *open* or *closed* source, having different software licenses which determine the rights of a software user. As the name "open source" license implies, anyone can inspect or even modify the source code of the software. The difference in the level of transparency of the source code in these types of software, i.e. how the source code is fully viewable and modifiable by anyone, can affect and change how a security audit might be conducted. This literature study will examine the challenges and unique aspects of security audits in open source software environments.

The motivation to undertake such a study is based on two parts. Firstly because security audits, as mentioned, are a critical part of IT security and secondly because open source software is today becoming more and more commonplace. For example, rather than as an option to proprietary software, many systems, even commercial ones, today utilize a large amount of open source components (Deshpande & Riehle, 2008) (Synopsis, 2017).

Thus, to help improve these essential security audits of open source software in the future, this literature study will examine the peculiarities and challenges that arise when auditing open source, be it during development, the implementation or integration of software, or more generally later on. What these challenges may be and how to help mitigate them will be examined through a comparison of scientific reports and publications relating to the topic.

2. Background

In this chapter, the concepts of a security audit and open source will be looked at in more detail.

2.1 Security audit

To ensure that the security policies and procedures of an organisation are being correctly enforced in practice, so that the physical security matches the "logical security" in relation to these documents, security audits are conducted (Onwubiko, 2009).

The term "security" in this context can be defined with certain models which are designed to guide policies for information security within an organisation (Rouse, 2014). The three main aspects of the *CIA* security model, *confidentiality*, *availability* and *integrity*, are the cornerstones which should be followed to ensure safekeeping of data in regards to information security. It can pose a big risk if one of these aspects is neglected (Pereira & Santos, 2010). *Confidentiality* refers to the measures undertaken to ensure sensitive information doesn't reach the wrong people and that the right people get it. Restrictions to ensure only authorized people can access the data. *Integrity* involves maintaining consistency and accuracy of data, it should not be altered by unauthorized people and measures to ensure the integrity of data should be in place. Finally, *Availability*, which is ensured through maintenance, keeping a system up-to-date and providing redundancy, disaster recovery etc. (Rouse, 2014).

More specifically, audits are necessary for organisations to perform in order to ensure that the internal policies and goals set by an organisation on how their security should be handled are followed. Aspects such as limiting external access into the IT environment, required password complexity for employees, separation of subnets and VLANs to slow down attacks or malware are a handful of examples of security measures which could be implemented (Lincke, 2015).

When all such security measures are in place and fulfil the specified policies as given by the organisation, audits should be conducted to assess if these security measures actually are in place and implemented correctly, as well as how well they work. The purpose of the audit is to make sure that both regulatory and security-compliance policies are in actuality maintained, that assets are protected, that security processes to protect them are up-to-date and working, and that the security processes are continuously improved to maintain the pillars of the CIA-model (Lincke, 2015) (Onwubiko, 2009) (Pereira & Santos, 2010).

Audits can also be performed on more focused areas, but with the same idea of ensuring they fulfil security and legal requirements as defined by policies. What is meant by this is when audits are conducted, for example, during development of software or during implementation or integration of new software, such as through a merger of systems, departments or companies. In these cases it needs to be ensured that after or during these events or the developed software still follows all the “rules” set out by the organisation (Duan, 2017).

A security audit should identify the main assets, such as sensitive or valuable data, and their vulnerabilities as well as possible threats and attacks towards them (Pereira & Santos, 2010). During the aforementioned evaluation and examination of the procedures and practices of an organisation, if needs to be determined if these assets are adequately protected. Do the security measures actually contribute to a better security for the organisations if they fulfil what the policies wants them to do? Perhaps something important has been missed? Perhaps they do not give adequate security under closer scrutiny, after all.

The increased computerized storage and processing of sensitive data over the past decades have led to increased demands on IT security to keep computer systems and data secure and confidential. As the consequence of the increased need for security in recent times, the *General Data Protection Regulation* (GDPR) introduced the European Union in 2018. After this introduction, the personal data of citizens within the EU became more sensitive than it had previously been, and the need for security around such information increased (Synopsys Inc, 2018). While not related to security audits directly, it can provide an example to show how security becomes more important, which in turn leads to security audits also increasing in importance with along with it.

Security audits of various types are usually preformed according to a kind of schedule. Routine audits may be largely automated to preform “normal” auditing activities as part of regular maintenance and can be done on a monthly, quarterly or bi-annually basis depending on the size and type of organisation. Lincke (2015) names minimum requirements for maintenance, monitoring and analysis of security audit logs. Server logs are recommended to be archived for months and anomaly logs are reviewed at least biweekly.

Other than that, there may also occur “special audits”, such as after a data breach, during integration of new software or other situations. In such cases, a special, perhaps extra in-depth, audit will be conducted to analyse and evaluate that situation in order to make sure that things are following the rules (Lincke, 2015).

A security audit is usually conducted by either an internal auditor from within the organization or a hired consultant or external auditor and can be performed in different ways depending on the context. These audits are not always done manually by human experts, but utilize special auditing tools which can either support the auditor or conduct largely autonomous audits.

2.2 Example Auditing Framework

While this report will focus more specifically on the challenges of open source security audits, it may be necessary to first explain the basic framework of a how a security audit is conducted in general. The basic overall steps presented here should not differ if the audit is performed on an open or closed source system and the general steps and points can provide an overview to be kept in mind when later results are presented.

Presented here is a framework proposed by Onwubiko (2009), consisting of five components. This is only a framework by one author, but it was chosen to be presented here due to it showing a good example of a “general" audit framework, not that something such as a “universal framework” exists, but which follows the same basic steps usually undertaken which are also presented in other frameworks or guidelines, such as by Pereira & Santos (2010), Szeżyńska et al. (2009) and Lincke (2015). The five components are as follows:

Security Policy, being a high-level formal document of an organisations core policies, such as use policies, incident response, email policy, hiring policy. It may also cover regulatory or legal compliance. These polices are assessed during a security audit to ensure that the organisation security policy covers every aspect of the business and that the policies are relevant and appropriate.

Audit policy is defined as the main component of the framework that points to specific aspects of the organisation that must be examined and assessed during an audit. Some examples are what that the organisation’s plans for protection are? If these plans follow the recommended standards and guidelines? If they are correctly implemented and follow the written processes for handling information security? An audit policy must define which activities are to be audited for systems, operating systems, users, networks and processes.

Processes, the processes around security that the organisations follows must be applicable and auditable. Each element of a process must work in consistency with what the organisations do. These processes must be audited to ensure that organisations follow the security standards, guidelines and recommended best security practices. Examples are infrastructure protection, disaster recovery or risk management.

Procedures, which the organisations follow to administer information security must adhere to recommended best-practice guides and standards. These procedures must be auditable and practical to the overall security policy and the processes in the organisation. Do the processes match the actual practices of the organisation? Are the processes consistent with the procedures? Also, does the organisation have appropriate procedures in place?

Procedures in line with the management standards of ISO/IEC 27000 are a common security best practice (Onwubiko, 2009). Families of ISO/IEC standards exists to comprise and define information security standards. These are published by the *International Organization for Standardization* (ISO) and the *International Electrotechnical Commission* (IEC) and the ISO/IEC 27000-series provides standards which can provide recommendations and guidelines to have best practice information security. Many organisations try to follow these standards to achieve the ISO certification which can be received by following them. A security audit can be done from this perspective, following the ISO/IEC 27000 standard to receive guidance and increased understanding in managing an information security management system.

Regulatory compliance, as an ongoing reactive process, security audit should be scheduled periodically to test policies, check controls, processes and procedures and see to it that everything is properly implemented and maintained. Information logs should be analysed after audits to help ensure this.

The framework proposed by Onwubiko (2009) presented here is as mentioned just an example of a framework for security audits, but it is easy to see many similarities in other proposed guidelines and frameworks. Szeżyńska et al. (2009) give several examples of near "universal" information security audit activities, such as pre-established purpose and planning, including the rights and obligations of the auditor, as well as the purpose and scope of the audit. Other aspects brought up include the identification and collection of "evidence", such as event logs and collected information from penetration tests.

Lincke (2015) also agrees that during an audit, whether internal or external, the auditor should confirm that policies and procedures are thoughtfully documented and implemented, including that things match reality. Employees should be mindful of security awareness and their security responsibilities and system administrators should be ensuring security for the information security program, data and equipment and monitoring logs.

2.3 Open Source Software

Now that what a security audit means and entails has been covered, a closer look around open source will be presented.

The main aspect of the concept of *open source* is the availability of the source code, which is released with a license that grants the right to anyone to examine, change, use and distribute it. This is in opposition to *closed* or *proprietary* source software. In closed source solutions, the creator, or in most cases, the company or organisation which the creator belongs to, are the only ones with control over the source code, including not only modification, but even being able to get access and inspect it. It should however be noted that whether a software is *open* or *closed* does not impact if it is free (as in, "costs nothing") or not. Proprietary software can be handed out for free or open source software can be part of a commercial business model (Raymond, 1999).

The *Open Source Initiative* (OSI) which since its establishment in the late 1990s, is the group whose board of directors *de-facto* decides what is open source or not. This is via *Open Source Definition*, published by the OSI, which determines whether a software license can be labelled open source or not. Software that is distributed under such approved licenses are *OSI-certified open source software* (Rosen, 2004).

There are different types of licenses; some don't allow you modify the source code, only inspect it. This can raise an interesting question, what if an overall system needs or uses several “parts” using different licenses? Reddy (2015) say that many for-profit companies, especially ones in their early stage, don’t fully realize that open source software can’t be freely used however one pleases, and that there can be severe consequences if open source licenses are not followed. It could therefore be possible that an unregulated mixture of licenses could present security and legal problems which would certainly violate policies set by an organisation.

There has been much discussion on the subject of whether open source software is more or less secure than its closed source counterpart. Open source software is, as mentioned, *transparent* and its source code can be readily examined. The code can be peer-reviewed and issues can be identified. This is argued to be a good security benefit, since it means many vulnerabilities are identified and fixed before they have a chance to become exploited (Raymond, 1999).

The security as measured by detected defects in the source code has been analysed and reported by *Coverity*, a provider of a static source code analyser. In a comparison of open source code and proprietary, closed source code, it was shown that the C/C++ code quality in open source projects surpassed the quality of proprietary projects at all code base sizes, measured in the identified defects and defects fixed in 2014 (Coverity Inc., 2014).

On the other hand, the inability to view the source code of a closed source system is also presented as a security benefit contrary to the easily viewed code of the open source software. In closed source software, the hiding source code is in itself a means of security. For example, it means that a potential attacker or similar would not be able to examine the source code in order to find weak spots, exploitable issues or other vulnerabilities. This concept is the so-called “*Security through obscurity*” (Schryen & Kadura, 2009).

Much debate has taken place regarding which of the two types of software is “more secure”, but it is something which is hard to measure. Advocates for both sides have arguments for the superior security of their favoured, respective type of software.

Putting the information about open source in the perspective of a security audit, it can be of importance since it would allow an auditor to be notified about and/or fix vulnerabilities that arise from discovered security issues in the system, whereas in proprietary systems, this is not possible. Proprietary software can therefore be “locked” in its current state with vulnerabilities present in the software, while waiting for patches from the company owning the software to be realised with fixes and solutions. This forces the use of workaround solutions until such an update patches the vulnerability (Baird, 2008). Furthermore, it also presents an interesting notion regarding how auditors make sure that code which is so *open* is fully secure and follows the policies of the organisation. Especially when new open source code is introduced to existing systems, since this can change things around in regards to following the set organisational policies.

3. Research Question

This chapter will cover the research question which aims to find out what potential challenges or peculiarities that the increasing number of open source auditors in different situations would have to face, as well has how to help mitigate them. Motivated by the described trend becoming more and more relevant for each day. Thus, the research question is the following:

What are the challenges when conducting security audits for open source systems and how can they be alleviated?

The question is asked with the intention to examine what these challenges could be and what to be mindful of during different types of security auditing relating to open source software systems. As well as to bring forth some type of conclusion of suggestions which could help mitigate these challenges.

As has been mentioned, the combination of the growing usage of open source making it more a matter of fact than an option, as well as the importance of security by upholding the pillars of the CIA model. Therefore, the motivation to undertake such a study is based on two parts. Firstly because security audits, as mentioned in the background section (2.1), are a critical part of IT security and secondly because open source software (2.2) is today becoming more and more commonplace. For example, rather than as an option to proprietary software, many systems, even commercial ones, today utilize a large amount of open source components, making it a reality that will be hard to avoid (Deshpande & Riehle, 2008) (Synopsis, 2017).

To help improve these important audits of open source software, be it during the development or implementation of it, or more generally when it is a part of the larger organisational system later on, this literature study aims to examine the challenges that may arise during said audits. What these challenges may be and also how to help mitigate them will be examined through a comparison, followed by further analysis, of scientific reports and publications relating to the topic. More detail on this process will be presented in the chapter Methodology (4).

Previously mentioned frameworks and other information, such as the CIA model, can be kept in mind as their general guidelines will usually “hang over” the discussion of auditing during the comparison and analysis of the literature. This is pointed at as these steps and the exact process of the audit they detail are not what this literature analysis aims to concentrate on. Rather, the scope is narrowed to focus on the specific open source auditing issues and how to help solve them.

4. Methodology

This chapter will cover and discuss the chosen method that will be used in order to answer the research question. It will be discussed why the method was chosen and how a study using that method is conducted, as well as why other methods weren't chosen.

4.1 Chosen Method

The method chosen for this work has been a *literature review*. This entails preforming a comprehensive literature analysis on written work and existing literature and scientific publications related to the topic, such as making categorizations and detecting patterns. When utilizing this method, every source needs a careful interpretation and analysis to ensure validity of the study.

To ensure that appropriate sources are used, they need to be made sure to actually be relevant. This can be done by searching in relevant bibliographic database containing journal and conference proceedings. Such databases are a good source of relevant literature, especially from databases of high reputation (Berndtsson et al. 2008). The systematic search should eventually amount to a relatively complete census of relevant scientific publications on the topic (Webster & Watson, 2002).

Berndtsson et al (2008) elaborates further that an issue that is fundamental in importance in terms of validity is the amount of collected literature. The reader's understanding and appreciation of the strategies and trust in the literature study increases when he or she is aware of the process of selecting sufficient materials. The validity in the literature study will be enhanced when the systematic process of selecting sources is conveyed to the reader and the reader can understand why specific sources has been chosen or not. Both the process and the analysis need to be properly addressed for the results to be trustworthy.

Wee and Banister (2016) mentions that a literature review paper should *add value*, as opposed to a literature *overview*. Value can be added to the review in different ways, considering what angle is chosen in writing the review. This literature review will aim to highlight various challenges which exists today and provide a suggestion on how some could be softened.

Wee and Banister (2016) also say that the methods section of a literature review is often much shorter compared to other types of scientific publications as the literature in the review is drawn from the extensive publications available. The literature that is to be used in this literature study are reports, conference proceedings and books from well-established and academic databases such as *ACM Digital Library, SpringerLink* and *IEEE Xplore Digital Library* (Brereton et al. 2007).

Webster and Watson (2002) recommend that, in the search for relevant articles, one may examine the table of contents to make sure the article contains useful information and is not a “false positive” just because it contains a relevant keyword. Many scientific publications can be “manually filtered” in this way, as it can sometimes be obvious that they appear in the results as a kind of “false positive.”

Criteria for inclusion for the sources to create an unbiased and relevant selection are that the literature should have to do with security audits and/or open source (general guidelines, implementation, development, licensing, how open source affects it, security, pros and cons). Furthermore, the sources should be from respected authors, or at least from authors which are not considered untrustworthy in the field, which literature from respected publishers should help ensure (Keele, 2007).

In order to find useful and relevant scientific publications related to the research question from these databases, relevant keywords and search strings related to the question were used to find useful and appropriate literature in these databases. These include *Security, Audit/Auditing, Open Source, Comparison, Organization/Company, Environment, Windows, Linux, UNIX, License/Licensing, Challenges* and *IS* in a variety of combinations. Wee and Banister (2016) recommends databases such as *Google Scholar* when the literature review aims to review more or less all of the main literature in the area. As a result *Google Scholar* was also used in addition to the aforementioned databases in order to search for information. Here, filters such as reports older than 2015 and no patents were selected in addition to removing results including the keyword “cloud”, by adding “-cloud” to the search.

After the sufficient amount of literature was found in the previously described way, it was subsequently filtered based on relevance to the topic, some of the literature was for example on seemingly related based on the title. Eventually only scientific reports with useful information relating to the stated research question left, which were used in the comparison.

This literature is then compared with each other to identify the challenge-related aspects of open source security auditing that appeared in several of the reports. These common aspects are then further analysed to identify what, in these collected literature, appears to be the main challenges an auditor might face. A chapter on possible solutions to these challenges is then presented.

4.2 Alternative Method

Aside from the chosen method of literature study, there exist other possible types of strategies that could have been used to conduct the study. One of these which was originally considered, was an interview study with relevant “people of interest” within organizations who might deal with audits of open source software. This type of study was disregarded, however, with one of the main reasons being the prospect of the difficulty in finding several people who would have had such specific experience of both security auditing and open source, and also be willing to conduct an interview.

5. Comparison of Collected Material

In this chapter a comparison of the various findings and information from the reports and literature which were found in the aforementioned databases through the previously described search-method are analysed and discussed in different categories.

5.1 Auditing when Implementing Open Source Software

Dashevskyi et al (2016) says that the question of whether open source software is more or less secure than its proprietary counterpart is not the right question to ask, at least from the perspective of the software vendor using the software. There may be situation when there is no alternative other than to use open source components in the larger "software chain", or that open source software might be more economical when offering functionalities which are otherwise expensive to implement. The yearly Coverity Scan Report from Synopsis (2017) say that the distinction between proprietary and open source is irrelevant today. Commercial software being shipped to customers can contain up to 90% open source code. Open source software is the norm, and some companies are even founded entirely on open source software. But with this new status quo, they want to improve the quality and security maturity of the software.

Dashevskyi et al (2016) says that, security-wise, open source software should be treated as one's own code. When wishing to integrate open source software into applications or products, one must consider which open source software products to use and how to deal with its maintenance. Code auditing tools are used to verify the combined code base of the open source component in conjunction with proprietary applications when preforming said security assessment in the aforementioned selection process of open source software.

A potential problem regarding maintenance is when a security issue or vulnerability is discovered and known to the public. A business software vendor then has to verify if that particular issue affects customers which are using solutions with that open source software in the manner in which they are using them. This can occur many years after the initial deployment (Dashevskyi et al, 2016).

With the notion from Dashevskyi et al (2016) that open source software should be treated as one's own code in mind, it can be noted that other author's note that the majority of users download and run compiled software, this goes not only for proprietary, but also open source software. Most users do not compile the available source, meaning, for example, that they implicitly trust that there are no backdoors inserted (Carnavalet & Mannan, 2014). In a scientific publication published by Unruh et al (2017), it was discovered that many vulnerabilities in open source projects had come to be after the developers had simply copied code or parts of code from (flawed) online tutorials and Q&A sites, and thus introduced the same flaws into the software which they are developing on.

Over 100 vulnerabilities very similar to code patterns in popular online tutorials were discovered, including 8 instances of SQL injection vulnerabilities which were all copied from a single tutorial. The publication is limited by examining only PHP application code, but is used to provide evidence for the concept of how insufficiently reviews tutorials are introducing vulnerabilities and compromising the security of open source projects, as well as how to discover them (Unruh et al, 2017).

The report by Unruh et al (2017) also shows how security audits can use these tutorials code as a base to find the related flaws introduced or "based" by them. This shows how non-proprietary software can introduce vulnerabilities when adjusting the code using presumed secure code from even highly rated, popular "tutorials." " "

Carnavalet and Mannan (2014) suggest that security critical open source software could be compiled and compared with the developer given binary to help prevent backdoors or other vulnerabilities from having been inserted into the compiled software by malicious or coerced authors. This would prevent targeted attacks and can be achieved via the use of verifiable builds that enables reproducing the official build, through either a comparison between the official and recompiled files (such as done by *Fedora*), or through a deterministic build process, which should always give the same output when repeated and thus exactly match the official build.

5.2 License Auditing

It is noted that careless use of open source software can introduce substantial legal and security risks, which can have serious consequences for both organization providing the software and possible clients or customers. This can be due to conflicting licenses and error-prone interdependencies in various open source software components. Keeping all these "parts" in check can be tedious and error-prone, especially if the software was imported from somewhere else (Duan et al. 2017).

Developers must manage all open source software components in an application by tracking and updating them, as they may contain exploitable vulnerabilities. In addition, various licenses for open source software needs to be followed. Even large companies such as Cisco, VMWare and Facebook have had problems regarding the security and licensing issues leading to legal disputes in recent years (Duan et al. 2017). In addition, the yearly *Open Source Security and Risks Analysis Report* by Synopsys (2018) said that a lack of licenses compliance is a risk related to open source and business run a risk when failing to ensure compliance with them.

German et al (2010) agrees with this and say that the licensing requirements are an aspect of using open source that requires understanding from the developers and integrators of the open source software. Determining the license of a package and assessing whether it depends on other software with incompatible licenses is not easy. Different licenses impose certain constrains in regards to how the software can be used or changed. Many packages even contain source code under different licenses, which can further complicate the determining the license of the entire package. Duan et al (2017) shares the same notion and says that managing these aspects usually require legal and technical expertise and be time-consuming and expensive, which often leads to it being overlooked.

5.3 Client Computers

In recent years, new malware penetrating into Linux systems are increasing at a high rate, using obfuscation techniques to better hide from traditional countermeasures. As a result, new types of *automated dynamic malware analysis systems* are being developed and used by security researcher to protect against aforementioned threats. The goal is to develop an efficient and generic technique which can detect malicious files (Damri & Vidyarthi, 2016). These measures could lead to or become common tools or techniques for Security auditors in ensuring security in more exposed Linux system, should the trend continue.

Windows security audits can be accomplished by using features such as *Global Object Access Auditing*, which can log successful or failed attempts of chosen types of events, as to not clutter up logs with information overload. Zeng et al. (2015) created a Linux auditing framework utilizing an adaptive audit logging mechanism in order to reduce performance overhead from logging after experiments indicated that the logging overhead is quite significant. The adaptive logging introduced in their study was shown to dramatically reduce system overhead, something which can be useful in the same ways Windows *Global Object Access Auditing* features reduced "information overload" in that operating system.

It can be seen from the information presented in these publications that certain features in Windows operating systems has the equivalents created for Linux grant similar functionality for the auditing process of those open source clients as well. As can be seen from Zeng et al. (2015); while the support from Microsoft grants their operating system Windows specific auditing functions, similar functions can be (and has been) developed and can be use in Linux machines. Due to Linux being open source and also very widely used, new software can rather quickly be created to suit new needs. The example here being when automated *dynamic malware analysis systems* is being created the "community" after an increase in new Linux malware (Damri & Vidyarthi, 2016). The benefits of open source is utilized well here to mitigate what could be a challenge for Linux in order to improve auditing capabilities in Linux to match that of Windows.

5.4 Implementation of Auditing and Security Tactics

When conducting a security audit for the first time in software development, developers of the software can realize that security requires a lot of attention. But Poller el al (2016) say that although, in their study, they noted that developers found security audits and workshops informative and that they raised awareness of security risks, it did not lead to much change in the development process afterwards.

Poller et al (2016) attributes this to preliminary explanations such as that development teams were globally distributed and were specialized in different areas, with management attributing security as an intrinsic requirement to the teams. Another possible reason was that product management and development management both, with different priorities, where cooperated in security had not been addressed so far. Security was perceived as “invisible” and not marketable to customers and was not highlighted as something that could serve as a product feature.

Perl et al (2015) say that many serious vulnerabilities are discovered in open source software despite security audits that should have found and removed them. This is due to the sheer volume of code being produced, as well as an overall lack of sufficient auditing. There is a lack of manpower and expertise. Only a small team of core developers usually review the code, even though anyone “in theory” could.

Furthermore, according to a study by Ryoo et al (2016), the majority of open source software projects have a rather limited scope regarding the adoption of security tactics, with data encryption being the most commonly used tactic. Other tactics were in comparison quite rarely used. It is suggested that this “presents an opportunity for the OSS community to explore other less heavily used security tactics and consider them for adoption”, and points that the overrepresentation probably stems from a lack of knowledge, rather than avoidance.

6. Analysis

As can be observed from the compared reports by various authors, there exists several challenges and unique quirks which an auditors would have to account for during an audit of open source systems. These challenges has been observed to take place at different stages in an open source systems lifetime, such as during the integration, creation and development, falling in line with what was said by Lincke (2015) regarding special audits during special occasions. How to manage security audits during regular operation in open source systems, in this case client machines, was also covered with how adaptive auditing mechanisms are created for Linux as detailed by Zeng et al. (2015).

The collected and compared writings of these scientific reports will be analysed here for the collective image and bigger picture of what the different author’s papers are saying to appear.

The most important points of the comparison will be identified through this, to find what area the biggest emphasis is put on among the author’s reports.

These clarified challenges which will emerge will then receive suggestions on how to help mitigate and alleviate them.

From reports such as from Dashevskyi et al (2016), backed up by the report from Synopsis (2017), it can be noted that the open source “movement” as a whole is growing and that open source software is becoming more and more a fact of life, rather than as an alternative to proprietary software. Sometimes commercial software event contained upwards of 90% of open source components. The distinction between proprietary and open source is getting blurred, and it’s time to focus on improving the maturity regarding security of the software.

With open source becoming (or having become) so widespread, it is important that people are able to handle open source software in a secure way and with appropriate auditing.

Dashevskyi et al (2016) puts forward the notion that open source software should be treated as one’s own code security wise. That careful consideration should be made regarding the selection of open source software products in regards to future maintenance and integration.

With the current trend in mind, people need to adopt a new mind-set regarding open source software and realize its differences in comparison with closed source software. For example, it was shown how people tended to implicitly trust, perhaps too much, code from online tutorials or available compiled source code in the way they might have with closed source software from a more “accountable” company (Carnavalet & Mannan, 2014) (Unruh et al, 2017). Little auditing was done on these occasions, clearly going against the recommendation by Dashevskyi et al (2016) that the code should be treated as one’s own. Precompiled binaries were used without inspection and flawed code from tutorials was introduced without auditing (Carnavalet & Mannan, 2014) (Unruh et al, 2017).

These are all things which can threaten the *integrity* aspect of the CIA model, since the integrity of the code can become compromised by using such flawed code found on online. It can also affect the *confidentiality*, since other people may use “backdoors” or similar vulnerabilities inserted through this flawed code. Carnavalet & Mannan (2014) and Unruh et al (2017) made suggestions on how to think and act when auditing the security of new open source software before it is integrated into an organisation or product. This included analysing code before its introduction (Carnavalet & Mannan, 2014) and/or the mind-set of “treating the code as one’s own code” (Dashevskyi et al, 2016). They cited a lack of these qualities as a reason for why this challenge exists today.

As can be noted from the publications of both German et al (2010) and Duan et al (2017)., licenses are clearly an aspect which should be payed extra attention to during the investigation and examination during a security audit when using open source components to ensure full compliance. Incompatible licenses can lead to the *integrity* part of the CIA model being neglected, since the consistency and trustworthiness of the data becomes jeopardized. Referring to the example framework of a security audit (chapter 2.1.1), one can see that legal compliance, such as mentioned in the *security,* relates heavily to the compatibility of the licenses, since they would otherwise not fulfil the legal compliance of that policy. As the framework suggests, these policies should be examined during security audit to ensure that every aspect is covered.

The authors of both publications seem to agree that larger systems can increase the chance for license incompatibilities. This can be especially important to keep in mind after, for example, combining previous systems or after a merger with another organization, which can have combined, or brought in new, components and the licenses that go with them. Auditors in an environment with open source software should therefore, in the event of a merger of organisations, departments or systems, keep a good track of the open source licenses which is part of software. Ensuring that licenses are being followed and that components with "contradictory" licenses aren't present, which could even result in legal risks. This falls in line with was what said by Reddy (2015): that many for-profit companies, especially ones in their early stage, don't fully realize that open source software can't always be freely used however one pleases, with both legal and security related consequences if open source licenses are not followed. Meaning that it could violate an organisations policies.

To audit this aspect, one should extract the licenses and dependencies declared in the package, extract and classify licenses from the source code files themselves as confirmation, and use this information to detect licensing incompatibility issues (German et al, 2010). Tools to help automate the procedure of detecting license incompatibility issues or violations are being developed (German et al, 2010) (Duan et al, 2017).

Similarly, certain functions to audit open source operating systems, such as Linux, can be seen to have been created and developed by the "community" to grant similar functionality to the proprietary auditing capabilities in the Windows operating system (Zeng et al, 2015) (Damri & Vidyarthi, 2016). A typical benefit of open source which shows how the openness of it lets "anyone" give it the same benefits as its proprietary counterpart, in this case an adaptive audit logging mechanism which could reduce performance overhead from logging, useful in the same ways Windows *Global Object Access Auditing* features reduced "information overload" in that operating system.

This benefit should be utilized more. As both the reports by Perl et al (2015) and Poller et al (2016) notes, there seems to be a need for better knowledge, skill, and integration of security auditing and implementation of security tactics particularly in open source software development, as well as a subsequent follow-up of implementation of security tactics to fix the issues which are discovered, particularly in open source software development.

Both the report from Poller et al (2016) and Perl et al (2015) point towards an overall lack of auditing, through mismanagement or a lack of manpower and expertise. Poller et al (2016) suggest that the results of first time security audits should be incorporated into the overall setting of the development team using activities and tools to make the security status more visible to managers and foster an understanding of security as a feature. It is stressed that this is not just a simple developer awareness problem.

The chance to implement and explore security tactics as suggested by Ryoo et al (2016) could benefit such a development. Ryoo et al (2016) say that the opportunity to adopt less used security tactics can provide a chance to research and find ways to implement and incorporate these kinds of security tactics into an existing hierarchy, and to audit and evaluate the effectiveness of them, once incorporated. This will make security more visible to managers and foster an understanding of security as a feature (Poller et al, 2016). These are points which could have a positive societal impact if they are applied more thoroughly.

7. Contribution

This chapter will discuss how to solve the identified main challenges, with particular focus on the concept of reproducible builds and how it can be used to alleviate what has been considered by this report to be the biggest open source specific auditing challenge.

7.1 Solutions to Challenges

From this analysis of the information gathered from the reports, a clearer image appears in regards to the questions of what the main challenges in open source security auditing are. This chapter will examine the challenges identified as most crucial and closer examine how they affect the security of an organisation, as well as the main contribution of the report, how to mitigate and alleviate the challenges facing the auditors.

The main challenges which were identified can be summarised in this report as "*licensing issues*" and "*too much trust in unverified binaries.*"

Firstly, regarding *licensing issues*, one of the main issues which appears as an important issue in the analysed reports is the matter of licenses and the difficulties of determining the compatibility of different licenses, especially in cases were a package may contain source code of several licenses (Duan et al. 2017) (German et al. 2010).

Since managing the compatibility and compliance of licenses usually requires legal and technical expertise, this becomes a big auditing challenge for open source auditors. Referring to the example security audit framework in chapter 2.1.1 (Onwubiko, 2009) we can see the point of *regulatory compliance*, which entails periodically testing processes, procedures and policies in order to ensure that everything is properly implemented and maintained. This would naturally include legal policies, something which a neglected auditing of licenses can jeopardize. It can also impact the CIA *integrity* aspect, as discussed in the analysis.

This is clearly something which can be harmful to an organisation (Synopsis, 2018)(Reddy, 2015), as even large companies like as Cisco, VMWare and Facebook have had problems regarding the security and licensing issues leading to legal disputes in recent years (Duan et al. 2017).

However, there has been steps taken to help improve this situation, automated tools to detected incompatibility or violations in licenses are being developed (German et al, 2010) (Duan et al, 2017). Thusly, this chapter will recognize that this is real issue, but that successful steps are already being made to alieve it. Because of this, too much focus on helpful contributions in this area will not be made here, but instead on other, more unsolved problems. Therefore, the main contribution of this report to help mitigate the challenges in open source will focus on the area regarding the overall lack of auditing in open source and unchecked use of unsecure and unaudited code, and how to fix this.

This brings us to the second main challenge, *too much trust in unverified binaries*. As has seen in the analysis, from what could be gathered from the publications from Dashevskyi et al (2016), Carnavalet & Mannan (2014) and Unruh et al (2017), the CIA pillars of *integrity* and *confidentially* can easily become threatened when there is insufficient security auditing when dealing with open source code, or when easily availed code taken from online sources are carelessly handled. And there is indeed insufficient open source auditing: the way people view open source software or code does not mesh with the considerations and auditing needed. This problem can be seen as somewhat less technical compared to more other more challenges, but from the collected literature it becomes apparent that it is in fact a real issue. The challenge seems to stem from “outdated” ways of thinking about and seeing open source from people used to handling mostly closed source software.

The *Integrity*, i.e. maintaining consistency and accuracy of data, is jeopardized if unverified open source code is used in such a widespread manner. Carnavalet & Mannan (2014) raises the point in regards to the use of binaries which are not being compiled by the users themselves, but who instead implicitly trust the source to not have any backdoors, which can easily be drawn to the potential neglect of the *Confidentiality* pillar. The measures undertaken to ensure that (potentially) sensitive information doesn’t reach the wrong people can naturally not be ensured to fulfil their purpose if such backdoors have been introduced to due insufficiently audited open source software.

The security audit framework presented in chapter 2.1.1 by Onwubiko (2009), but which also falls in line with most other recommendations and guidelines by other authors, brings up how *Processes* must be auditable (and audited) in order to ensure that they follow the security guidelines, standards and recommendations of the organisation. The aforementioned neglect of auditing for the open source code can be seen to go against these guidelines, and as a result goes against fulfilling the necessary components of a comprehensive security audit.

For the auditor of open source systems, a necessary improvement in regards to this trend is that the *Audit Policy,* as described in the aforementioned framework, should include the necessary steps of compiling and examining open source code in order to audit them. This would help with the issues of lack of treating the code as “one’s own” and too much trust being put into available binaries.

So to help work against this challenge, the aforementioned steps to the *Audit Policy* should become part of the general order of business in the event of an audit. However, inspections of such nature in this particular context should perhaps only be necessary in the event of the open source softwares or components introduction, as opposed to a general scheduled audit, as this is the only time new code is introduced in such ways. The way to compile and examine the source code in the way described during a security audit can be done through the method of *Reproducible Builds*.

7.2 Reproducible Builds

The contribution and proposed method in this report in order to help counter the problem of *too much trust in unverified binaries* and to mitigate this challenge, is through the use *Reproducible Builds* and to fully integrate them in a workable way into an open source auditing framework. These methods should be used instead of relying on the precompiled binaries, which was touched upon in the report by Carnavalet and Mannan (2014).

The official website of the Reproducible Builds project defines that a build is reproducible if anyone can recreate an exact bit-by-bit copy of a binary if given the same source code and instructions. The reproduced build should after that be verified, usually using cryptographically secure hash functions, to ensure the integrity of the build. (Reproducible Builds, 2019). While reproducibility may seems like an obvious property that builds should have by default, modern compilers can make it easy to unknowingly introduce problems (Devietti, 2019).

A programmer or user who compiles software from the source code can use a variety of compilers, with different versions and updates like all other applications. But these different compilers can produce slightly different output, even from the same source code. These inconsistencies can lead to unreliable software and lead to one compiled version working while another doesn't (Zimmer, 2018). Which version is secure or not can become very hard to determine if it is due to one of these compiler-related differences. A malicious compiler could be created to purposefully create code with purposefully placed vulnerabilities or exploitable faults (Zimmer, 2018). This is something which has been an issue discussed since the 1980's (wiki.c2.com, 2014).

Several open source projects, such as *Debian* and *Fedora,* try to utilize reproducible builds to allow anyone to audit the source code to verify the code created through the identical recreation and ensure that no flaws have been introduced during the build process (Debian, 2019). They are working on taking the entire Linux OS and making it so that the compiler and environment used to create an application are recorded, making it possible to compare and verify the compiled code, something which will lead to greater reliability and safer software. (Zimmer, 2018).

To further develop this technique of flaw finding for open source security related purposes and making it a "standard" as an integrated part of open source security audits, to be used in the appropriate contexts, would greatly play to the strengths of open source since the method is only fully possible if the process does not rely on close source software.

The difficulties with reproducible builds in closed source is due to a perfectly deterministic build only being achievable if all variables are controlled, something which is not possible to ensure if the build process relies on closed-source software where the exact documentation is not all available (Carnavalet & Mannan, 2014). This makes it a good open source related solution to an open source challenge which auditors evidently faces.

Even if the software proves to be unable to be perfectly recreated, it would then provide a chance for the auditor to see why it differs. This could be due to minor, irrelevant things, such as differing embedded timestamps, or it could lead to the discovery of different threats to security (Carnavalet & Mannan, 2014). Either way, it would all be beneficial to the overall auditing process and help alleviate the documented troubles around this particular issue regarding open source code.

As mentioned, there is a huge amount of trust but in distributed binaries, and even if an attacker would not have introduced a backdoor in the source code, the toolchain (programming tools used to create software) could be compromised. This would not be discovered during an audit, as the source code would not contain the error itself (Devietti, 2019). However, with reproducible builds, even this could be avoided since the binary package would differ from the official version and could be discovered during this new step in the auditing procedure (Devietti, 2019).

Should reproducible uploads become mandatory, attackers would also not have the same enticement to compromise the system of a developer with upload rights as that developer would not be able to upload a binary which does not match the uploaded sources.

The attacker would also have to compromise all machines which can audit the reproducibility of the uploaded source (Debian, 2019). It can also help in situations were a reported bug software from long ago can be recreated so that a suitable fix can be developed (Fowler, 2010).

The CIA pillars which the usage of unverified open source code could jeopardize, namely the previously mentioned *Integrity* and *Confidentiality*, would naturally have the risks towards them lessened as a result of more mandatory use of reproducible builds.

For these reasons, Reproducible Builds are an excellent way to solve the challenge of poor auditing of open source code to due to lax handling and too much trust of available sources. It should be more widely introduced, even striven to be made mandatory, and through the security audits help in improving the security of open source.

8. Discussion

This report has aimed to answer the research question of what the challenges within open source auditing are and how to alleviate them. Through comparisons and analysis of the collected material of the literature review, challenges have been identified and light has been shed on the main challenges that open source auditors might face in the work. While focus has been given on important issues identified as the biggest, such as the issues that can occur related to license auditing, the main challenge which has received the most focus in this report has been the challenge regarding an overall insufficient lack of auditing of open source code in general. This is including a too large of a trust in the validity and too much trust in open source code found at different places online, a “blind” usage of pre-compiled code.

This challenge has also received the most focus in regards to how to solve it, with the concept of reproducible builds. The report has shown how they would greatly benefit open source auditing and play to its strengths, as well as how to introduce them in the auditing process. This process has not only been shown to have a lot of support of existing open source projects, but the report has also shown how reproducible builds would help mitigate the specific issues which has been identified In the collected material (e.g. “Open source code should be treated as one’s own”, “open source code should be compiled yourself”). Its current trends mentioned earlier in the report continue and open source becomes more and more commonplace or “non-optional”, this would be greatly beneficial and is therefore presented as the main contribution of this report.

Overall, the process of writing this report has been fairly smooth, the systematic search for the relevant literature worked well and interesting points were found in several of them which made it identify the main challenges which present themselves across several of the collected papers. There was not much to consider in the way of ethical dilemmas or considerations, since this report is just based on other published peer-review scientific publications. It is not an experiment which could have affected people, for example.

9. Conclusions

In conclusion, after the review of the collected literature followed by comparing and analysing it, it can be seen that the biggest challenges relating to security audits in open source which became apparent through this report were the ones relating to licensing issues as well as, although not as much of a “technical issue”, how people think about, view and handle open source code and its security and its impact on auditing.

Looking back at the research question it can be divided into two parts:

1. *What are the challenges when conducting security audits for open source systems*
2. *How can they be alleviated?*

We can see that they have now been identified and suggestions on how to alleviate them has been brought forward, answering both questions.

1. Challenges identified were narrowed down to the ones identified as most relevant, i.e. the main challenges. These were *licensing issues* and *too much trust in unverified binaries*.
2. While licensing issues already had progress in the development of automated tools to detected incompatibility or violations in licenses, the main contribution of the report on solving the second main challenge was with heavier use and further integration of *reproducible builds* into an open source auditing framework.

This has provided an answer to both parts of the research question.

As mentioned, the way to help solve one of the identified main auditing challenges is fortunately already in development: There are ongoing efforts being made to develop automated tools and programs which are designed to assist this part of an audit by automating the process of determining the licenses of source code and in finding incompatibilities between several simultaneously used licenses.

However, such a developed integration is not present for the other the main challenge of ***too much trust in unverified binaries***, which relates to how people simply put too much trust in open source code and binaries without proper verification, and as a result of that handle it in very unsecure ways. This report has shown this to be counterproductive to several auditing and security components which are a part of open source security audits.

The answer to this challenge which the report has presented and suggested is the ***promotion and integration of reproducible builds into open software auditing frameworks*** to the point of trying to make it (at least de facto) mandatory. Some open source projects are already trying to utilize reproducible builds, but its continued development and integration into open source security auditing, to counter the current difficulties of unverified introduction of binaries and code (as discussed in Chapter 7.1 and 7.2) has not previously been brought up in such a context.

As the report has shown in these previous chapters, this would be greatly beneficial in alleviating the associated challenge by making the source code more secure. This would not just help solve the issue technically, but would also make people "view" of the code in a more appropriate way. This can be seen to echoing Dashevskyi's (2016) plea that open source code should "be treated as one's own code" as opposed of the large amount of trust otherwise put in it without verification, which has led to the aforementioned challenge. Auditors could with this method have a standardised way to verify that introduced code is not compromised and can also study it to identify why the recreated code might differ. This would encourage people to get more familiar with the source code while developing a deeper understanding of it. This familiarisation, as an auditor would have to properly analyse and understand the code, and not just blindly insert it, would also have the benefit of awareness and consideration for maintenance, which the lack of was another problem raised regarding the implementation of open source software.

How this open source specific issue would be part of an integration into a "regular" auditing framework for the increased open source utilisation (and corresponding audits) is shown in chapters 7.1 and 7.2, which is what constitutes the main contribution of this report.

As mentioned, there does exists some projects which wish for further integration of reproducible builds, but it is something which needs more attention and integration in order to maximise the effectiveness of the tactic in regards to security and verifiability. Especially in this unique area of specific lay open source auditing and corresponding frameworks.

In summation, to create a standardised way for open source code to be compiled and examined in such a way in order to audit them, and to introduce this as a common general step of the *audit policy* of an *open source auditing framework*, would be greatly beneficial for the future conduction of open source security audits and would impact society in a beneficial way. It serves as an effective solution to the challenge of *too much trust in unverified binaries*.

10. Future Work

For future work, it would be interesting to focus more on the now identified largest challenges. This report also found several minor challenges through the comparison and analysis of the literature review, only filtering out and identifying the main ones after this process. A future work could perhaps lay more focus on these challenges as well as to continue to develop a more practical integration of the suggested "mitigations" (such as reproducible builds) into a practical new auditing framework for open source auditing. To create some future work investigating this in more detail could make for an appealing future report.

References

Baird, S. A. (2008). “*The heterogeneous world of proprietary and open-source software*”. In *Proceedings of the 2nd international conference on Theory and practice of electronic governance* (pp. 232-238). ACM.

Berndtsson, M., Hansson, J., Olsson, B., & Lundell, B. (2007). ”*Thesis projects: a guide for students in computer science and information systems”*. Springer Science & Business Media.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., & Khalil, M. (2007). “*Lessons from applying the systematic literature review process within the software engineering domain*”. *Journal of systems and software*, *80*(4), 571-583.

de Carné de Carnavalet, X., & Mannan, M. (2014). ”*Challenges and implications of verifiable builds for security-critical open-source software*”. In *Proceedings of the 30th Annual Computer Security Applications Conference* (pp. 16-25). ACM.

Coverity, Inc. (2014) “*Coverity Scan: 2013 Open Source Report*”. Synopsis.

Damri, G., & Vidyarthi, D. (2016). “*Automatic dynamic malware analysis techniques for Linux environment*”. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 825-830). IEEE.

Dashevskyi, S., Brucker, A. D., & Massacci, F. (2016). ”*On the security cost of using a free and open source component in a proprietary product*”. In *International Symposium on Engineering Secure Software and Systems* (pp. 190-206). Springer, Cham.

Debian (2019) “*ReproducibleBuilds About*”. https://wiki.debian.org/ReproducibleBuilds/About (Viewed 2019-07-26)

Deshpande, A., & Riehle, D. (2008). “*The total growth of open source*”. In *IFIP International Conference on Open Source Systems* (pp. 197-209). Springer, Boston, MA.

Devietti, J. (2019) “*Introduction to Reproducible Builds*”. https://www.cloudseal.io/blog/2019-05-15-introduction-to-reproducible-builds (Viewed 2019-07-28)

Duan, R., Bijlani, A., Xu, M., Kim, T., & Lee, W. (2017). “*Identifying open-source license violation and 1-day security risk at large scale*”. In *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security* (pp. 2169-2185). ACM.

Fowler, M. (2010). “*ReproducibleBuild*” https://martinfowler.com/bliki/ReproducibleBuild.html (Viewed 2019-07-28)

German, D. M., Di Penta, M., & Davies, J. (2010). “*Understanding and auditing the licensing of open source software distributions*”. In *2010 IEEE 18th International Conference on Program Comprehension* (pp. 84-93). IEEE.

Keele, S. (2007). “*Guidelines for performing systematic literature reviews in software engineering”* (Vol. 5). Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Lincke, S. (2015). "*Security planning: an applied approach"*. Springer.

Onwubiko, C. (2009). "*A security audit framework for security management in the enterprise*". In *International Conference on Global Security, Safety, and Sustainability* (pp. 9-17). Springer, Berlin, Heidelberg.

Pereira, T., & Santos, H. (2010). "*A security audit framework to manage Information system security*". In *International Conference on Global Security, Safety, and Sustainability* (pp. 9-18). Springer, Berlin, Heidelberg.

Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., ... & Acar, Y. (2015). "*VCCfinder: Finding potential vulnerabilities in open-source projects to assist code audits*". In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 426-437). ACM.

Poller, A., Kocksch, L., Kinder-Kurlanda, K., & Epp, F. A. (2016). "*First-time Security Audits As a Turning Point?: Challenges for Security Practices in an Industry Software Development Team*". In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1288-1294). ACM.

Raymond, E. (1999). "*The cathedral and the bazaar*". *Knowledge, Technology & Policy*, *12*(3), 23-49.

Reddy, J. (2015) "*The Consequences of Violating Open Source Licenses*". Berkeley Tech. L.J. Blog. *Berkeley Technology Law Journal University of California, Berkeley School of Law*

Reproducible Builds (2019) "*Reproducible Builds*", https://reproducible-builds.org/docs/definition/ (Viewed 2019-06-05)

Rosen, L. (2004) "*Open Source Licensing: Software Freedom and Intellectual Property Law*". 1st Edition. Prentice hall Professional Technical Reference, Upper Saddle River, New Jersey, USA.

Ryoo, J., Malone, B., Laplante, P. A., & Anand, P. (2016). "*The use of security tactics in open source software projects*". *IEEE Transactions on Reliability*, *65*(3), 1195-1204.

Rouse, M. (2014) "*Confidentiality, Integrity, and Availability (CIA triad)*". *WhatIs.* https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA (Viewed 2019-06-05)

Schryen, G., & Kadura, R. (2009). "*Open source vs. closed source software: towards measuring security*". In *Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 2016-2023). ACM.

Synopsys, Inc. (2017) "*2017 Coverity Scan Report Open Source Software—The Road Ahead*"

Synopsys, Inc. (2018) "*Report: 2018 Open Source Security and Risk Analysis*"

Szeżyńska, M., Huebner, E., Bem, D., & Ruan, C. (2009). ”*Methodology and Tools of IS Audit and Computer Forensics–The Common Denominator*”. In *International Conference on Information Security and Assurance* (pp. 110-121). Springer, Berlin, Heidelberg.

Unruh, T., Shastry, B., Skoruppa, M., Maggi, F., Rieck, K., Seifert, J. P., & Yamaguchi, F. (2017). “*Leveraging Flawed Tutorials for Seeding Large-Scale Web Vulnerability Discovery*”. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*.

Webster, J., & Watson, R. T. (2002). “*Analyzing the past to prepare for the future: Writing a literature review*”. *MIS quarterly*, xiii-xxiii.

Wee, B. V., & Banister, D. (2016). “*How to write a literature review paper?*”. *Transport Reviews*, *36*(2), 278-288.

Wiki.c2.com (2014) “*The Ken Thompson Hack*” http://wiki.c2.com/?TheKenThompsonHack (Viewed 2019-07-26)

Zeng, L., Xiao, Y., & Chen, H. (2015). “*Auditing overhead, auditing adaptation, and benchmark evaluation in Linux*”. *Security and Communication Networks*, *8*(18), 3523-3534.

Zimmer, D. (2018) “*Reproducible Builds – Solving an Old Open Source Problem to Improve Security*”. https://www.privateinternetaccess.com/blog/2018/07/reproducible-builds-solving-an-old-open-source-problem-to-improve-security/ (Viewed 2019-07-26)