SSR: A FRAMEWORK FOR A SECURE SOFTWARE REUSE

Ahmed Redha Mahlous

Prince Sultan University, Riyadh
e-mails: armahlous@psu.edu.sa
Saudi Arabia

Abstract: In this paper, we propose a threat model and a framework for the secure reuse of software. The proposed framework outlines the procedures to follow in order to avoid threats in a simple way. Addressing and resolving security issues from early on is relatively simpler and more cost effective than doing so later, and this is what the framework aims to achieve.

Key words: reuse, security, software, framework, risk.

1. INTRODUCTION

Software reuse is a software engineering discipline where developers reuse existing software in their own development process. This strategy was first proposed more than 45 years ago [1] but only became a norm for new business systems in the year 2000. Reasons for adopting a reuse-based development strategy include the need to cut maintenance and production costs, increase software quality, and offer faster delivery of systems. Another reason for the increase in software reuse is the large availability of free and open source projects on the Internet.

However, the use of third-party code does not come without a price. The Open Web Application Security Project (OWASP) [2] for example, has listed the use of vulnerable third-party code in new applications as being among the 10 most critical web application security risks. Research by Veracode, a company that specializes in code analysis, showed that third-party applications failed OWASP's Top 10 policy more frequently than internally developed applications (see Fig. 1). This reinforces the idea that the use of third-party components should adhere to a strict security policy before being used by developers.

In its 2017 State of the Software Supply Chain Report [3], Sonatype revealed that most large organizations have no control over which components are used in their software development projects, and that many of them admitted that developers do not focus on security.

In this paper, we propose a framework for the secure reuse of software along with a threat model, which if followed properly will minimize the risk of introducing security vulnerabilities through third-party software components.

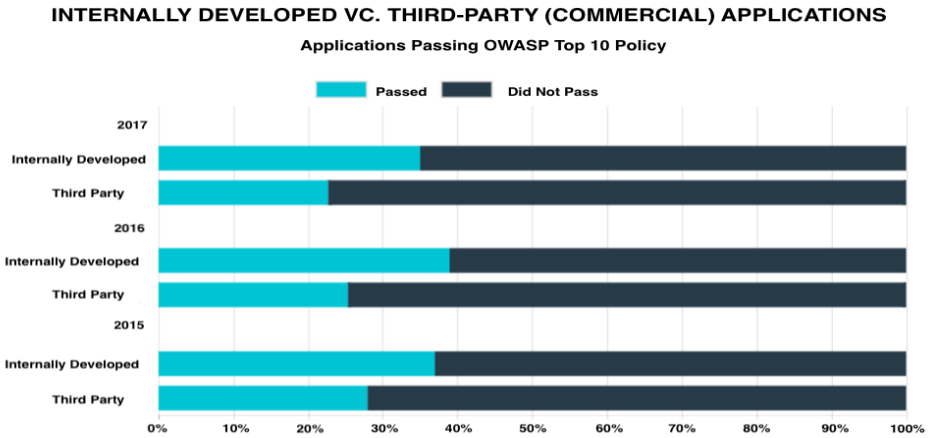

Fig. 1. Percentage of internally developed vs third party applications passing OWASP's top 10 policy

Section 2 presents a selection of works related to software reuse. Section 3 considers some of the risks associated with software reuse, while Section 4 lists some of its benefits. Section 5 introduces the proposed framework, which aims to improve the status quo. Section 6 introduces the proposed software reuse threat model, and Section 7 concludes the paper.

2. RELATED WORKS

Software reuse has become common practice in many software engineering projects. It has been considered as the main solution for the software crisis since the early days of the software engineering discipline, [4-6].

Software reuse is defined as the informal sharing of components among people working on the same or similar projects [7]. Several studies have examined the software reuse principle across different domains. In the architecture domain, for example, authors in [8] proposed an approach to detect anomalies within architectural projects that could affect the project quality attribute, especially component reuse. Authors in [9] presented a framework for software reuse engineering management consisting of four types of processes, multiple factor analysis, and key point management.

The search and organization of different components within a huge repository is considered a major problem in software reuse. Many researchers have suggested solutions to solve this problem [10-12].We also find that the reuse concept has been the study of many authors in different areas [13-17].

The reuse of software requirements has received a lot of attention by many researchers [18-20], as it provides a solid support structure for developing high quality software. Authors in [21] investigated how a systematic software reuse strategy can contribute to a successful reuse program. The problems associated with the maintenance of reusable software components was the focus of [22], and in [23], a framework for the certification of reusable software components was proposed.

3. RISKS IN SOFTWARE REUSE

As software is no longer being developed from scratch, 80 percent of code used in a typical software application relies on third-party components [24]. This is a double-edged sword because it saves development time but can also create a security gap in the system under development.

Some of the risks associated with the use of third-party components include:

3.1. Lack of careful evaluation

If a company were to buy proprietary (closed-source) software for its use, it would conduct a thorough analysis before adopting the product, which may include product evaluation, requirements analysis, defining acceptance criteria and comparing the product with other competitors. On the other hand, if the third-party product was open source, it would not undergo the same rigorous process as its closed-source counterpart. A company's reputation and credibility may be damaged if this inattentiveness results in a breach of private customer data through an attack vector introduced by a non-secure component.

3.2. Bogus third-party software

Malicious Internet users may embed viruses, backdoors, keyloggers, and other harmful instructions within their packaged software. They may even hijack the official distribution channel of a legitimate software supplier and infect it [25]. The absence of a clear security policy to follow when using external software leaves companies at a greater risk of introducing such threats into their own applications.

3.3. Lack of sponsorship

A lot of the popular open source software products are supported by a consortium, which is a group of people or organizations dedicated to further enhance and maintain the product out of interest and mutual benefit. However, if the open source software used by a company is not that popular, it usually lacks this support, which makes it difficult for the company to patch any discovered vulnerabilities. When this happens, companies are faced with two choices, both of which carry a significant resource cost. The first is to abandon the product and pick an alternative, which means that any internal code that calls into this third-party code will need to be re-written, and the greater the difference between the two APIs, the tougher this effort will be. The second choice is to attempt to "fork and patch" the original code if the original author(s) are unreachable or unable to issue a patch within a reasonable timeframe.

4. BENEFITS OF SECURE SOFTWARE REUSE

A secure software reuse approach can bring many benefits to an organization, some of which are:

4.1. Reusability prospect. In the software reuse process, the same component may be used across multiple systems. Scanning for security vulnerabilities within a

component before using it in one project is an opportunity to define common security measures associated with that component, ready for use in future projects.

4.2. Reduced cost. Scanning a component for vulnerabilities before using it once eliminates the need to scan it again when used in a future project, thus saving time and resources in the long run.

4.3. Improved quality. A secure, reusable component is one which has been inspected for security vulnerabilities and given thorough attention before first use, including making sure that it originated from a reputable source. Using "cleared" components reduces the risk of vulnerabilities being introduced and results in a more reliable product overall.

4.4. Customer satisfaction. Delivering secure applications will result in the satisfaction of customers with the product and the organization. This is particularly important as it only takes one security incident for customers to lose trust in a product.

4.5. Building a secure component catalogue. Ensuring that components are secure before using them will result in a catalogue of secure components that can be searched and consumed by developers across the organization.

4.6. Consistency. Reusing the same secure component will provide some level of consistency in terms of security throughout the organization. Developers searching for a particular functionality can browse the catalogue, and if they find something that matches their requirements then there generally won't be a need for bringing in yet another similar one.

4.7. Less staff with security skills required. Having a catalogue of secure components will lessen the need for security expertise in the software reuse and development process.

5. THE PROPOSED SECURE SOFTWARE REUSE FRAMEWORK

Most of the time, security is neglected during the software reuse process and left until the end of the software's deployment. This practice leads to unnecessary re-work, unsecure design, and a greater risk of vulnerabilities in the end. Software reuse as currently practiced suffers from a lack of proper analysis and a clear process to follow. From our own experience, developers (especially junior developers) are often eager to bring in free, third party software without hesitation. While this is not a problem and is in fact one of the great advantages of free and open source software, companies need to have a more formal process to handle the security aspects of introducing external code. To be effective in doing this, they need to have a proper framework aligned with the organization's goals. In this section we describe our proposed framework, which is to be followed whenever a third-party software product is chosen as a solution during any phase of the SDLC. It is divided into two parts: the first part governs the process in which third-party components are brought into the company, while the second part is concerned with the routine maintenance of the brought in components, some of which may be in use by more than one development project. The key elements of the proposed framework described in Part 1 and Part 2 are shown in Fig 2.

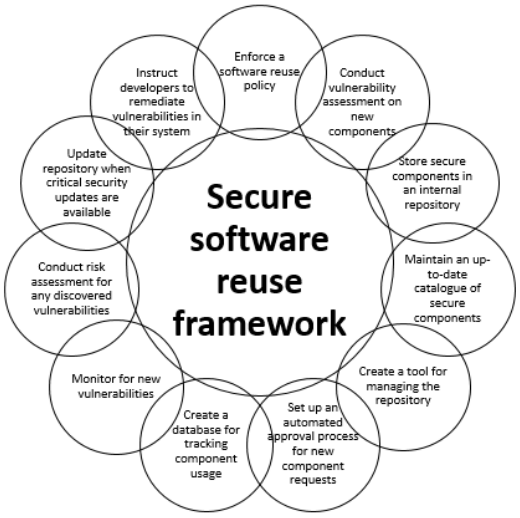

Fig. 2. Key elements of the secure software reuse framework

5.1. Part 1

The first step in our proposed framework is the **enforcement of a component reuse policy** by the company. Developers wishing to bring in third-party software would make an official request, which includes a business justification. Requiring a business justification with the request ensures that software is not brought into the company for unnecessary reasons, and the justification becomes even more important if the software comes at a cost. One of the proposed terms in this policy is that the catalogue (described a bit later) should first be searched for a component with the desired functionality, and only if nothing is found that matches the developers' requirements then a request can be made. This will promote software consistency across the company as mentioned earlier on, and will also contribute to the growth of a wider knowledge base around components, which in turn contributes to better developer productivity. The policy, however, should not completely prevent experimentation with similar components but should instead provide guidelines on how to "sandbox" running programs.

The request should then propagate to a team whose responsibility is **assessing third-party components** for internal use. This team should be trained in software security analysis, and should test run components in isolated or virtual environments. For all software components, whether closed or open source, the team should be aware of the type of license that accompanies the software and should utilize identity and integrity verification methods (checksums, PGP signatures, digital certificates, etc.) where possible to validate the downloaded copy. The assessment should also pay attention to the popularity of a component outside the company. Software that is more widely used is usually more mature, secure, and stable than unpopular software.

Once the team is satisfied that the component is vulnerability-free, it is **stored in a repository** on an internal server. A **tool to help manage the repository** such as a

package manager or an internally-developed alternative would be beneficial as the number of components in the repository grows and the task of managing them becomes more complex. Details of the added component such as name, version, date downloaded, source, license type etc. would then be **added to a catalogue** of "cleared" components where each component is assigned one or more meaningful labels, e.g. a Python package for data analysis would have the labels "python", "plot", "graph", "data analysis", and so on.

In technical terms, this catalogue can take any shape or form. In its simplest form it could just be part of the repository manager. This means that the repository manager should also be accessible to developers (with reduced privilege to prevent the direct addition of components). If the repository manager does not have this capability and cannot be customized, the next simplest form would be to store a metadata file in the repository along with each component. The catalogue in this case could be the output of a script that reads all of the metadata files in the repository and lists them in an easy-to-read format, with the option of filtering results based on the labels mentioned earlier. Another example could be a separate database that stores details of all the components in the repository, perhaps with a web interface to allow for a more pleasant interaction. The most important thing, regardless of the form that the catalogue takes, is that it should always be kept up-to-date with the components in the repository. Adding/removing/upgrading a component in the repository and updating the catalogue should ideally be performed as a single step or at most as two closely-linked steps in order to avoid having a mismatch between the information in the catalogue and the actual components in the repository.

This catalogue is then used by developers to search for components that fit their requirements. If they had previously submitted a request, an **automated notification process** would inform them that their component has been approved and added to the repository. They could then use the repository manager to download the component(s) from the repository to their computer for development.

Finally, the developers would **register the components** used in their project in a dedicated database. This "project dependency" database would make it possible for the security team (who manage the component repository) to gather statistics about component usage across the company, and to be able to easily draw up a list of people to notify if a component needs a critical update.

5.2 Part 2

This part of the framework covers the life cycle of components after they have been added to the repository. For users of the components, this would coincide with the development and maintenance phases of the SDLC. In other words, the projects using these components may either be in the middle of development or may be complete and signed off.

a) Monitoring: After a component is added to the repository, the security team is responsible for **monitoring for updates** from the original authors/vendors via mailing lists and other communication channels, in addition to being up-to-date on the latest security threats. Newer versions of a component should be brought into the repository

after an assessment of the changes. If the release notes for the newer version indicate a critical security patch then the older versions(s) in the repository need to be marked as unsafe for use in the catalogue, and developers who are listed in the dependency database as users of the component would need to be notified and instructed to begin the remediation process.

b) Remediation: Companies should have a clear approach for **remediating any discovered vulnerabilities**. This can be done by classifying them based on criteria such as the severity or exploitability of the vulnerability in conjunction with the sensitivity of the application(s) impacted. The outcome of this **risk assessment** would decide whether the update is critical. Components that need to be updated must be updated in a careful manner by running with the new version in an isolated environment first. This is where end-to-end or continuous integration tests play an important role. Having a ready set of test suites for a wide variety of scenarios that an application can encounter in a "production" (real life) environment makes the update process smoother and ensures that the update does not break any existing functionality.

c) Pre-policy systems: For pre-policy systems, or systems developed before the introduction of the component reuse framework, the main difference would be that the components used in the system may have not gone through the initial vulnerability assessment. After the introduction of the policy, developers in charge of the system would need to **conduct a thorough source code scan** in order to identify all of the third-party components in use and register them in the project dependency database. Then, components, which are not already present in the repository, would need to undergo a full assessment by the security team. If successful then they are added to the repository. Otherwise, the aforementioned risk assessment would need to be conducted in order to determine the risk associated with the continued use of the components.

While the framework described in this section is targeted at large companies, it can be scaled down to small/medium companies or independent development teams. For example, a large company may have the budget for a dedicated security team as described earlier, but a smaller company may not, in which case the responsibility for monitoring external news feeds and mailing lists for vulnerabilities in used components falls upon the members of the development team using that component.

6. SOFTWARE REUSE THREAT MODELING

In this section we propose a reuse threat model, which will help developers to identify, enumerate, and prioritize potential threats, and to define an appropriate mitigation method to eliminate any adverse effects on the system. The objective of such a model is to allow developers to build a list of probable attack vectors related to software reuse. The proposed threat model is inspired by the Microsoft Secure Development Lifecycle [26]. It consists of four key phases as shown in Fig. 3: Decomposing and understanding the reused component, Threat identification and enumeration, Threat rating, and Mitigation options.

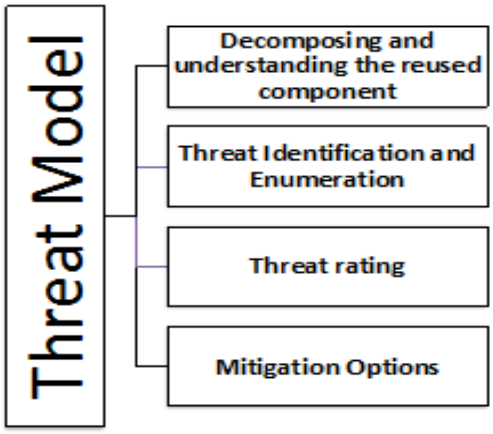

Fig.3. Threat Model

STEP 1: DECOMPOSING AND UNDERSTANDING THE REUSED COMPONENT

The first step in the proposed threat model is to try to understand how the component interacts with the rest of the application's components. "Application's components" here refers to the logical (code) or physical (associated hardware) parts of the application, e.g. the user interface, the logger, the database, etc.

With the help of use-case scenarios, developers should try to determine the entry points that an attacker can use to penetrate the system through the component. A data flow diagram can be created to help visualize the data flow between the reused component and the rest of the application's components. In this manner, a trust boundary will be created in order to aid the developers to better analyze the privileges and trust level, and to understand the threats. A sample data flow diagram is shown in Fig. 4.

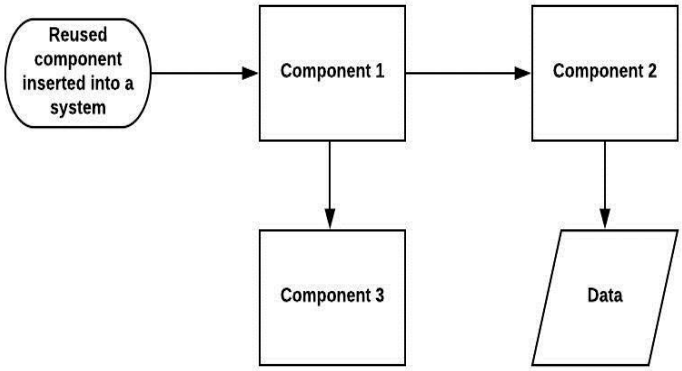

Fig. 4. Data flow of the interaction of the reused component with the rest of the system

STEP 2: THREAT IDENTIFICATION AND ENUMERATION

In this step, using the data flow diagram created in step 1, developers will try to determine what potential threats can be associated with the reused component. Threats are categorized based on the STRIDE [27] taxonomy:

- **Spoofing Identity:** Stealing or disguising one's identity with another by illegally using others' authentication information (e.g. username and password) to access a system. Another form of spoofing is server spoofing, e.g. phishing attacks.
- **Tampering with data:** Data tampering involves the malicious alteration and modification of data, such as unauthorized changes made to data held in a database. This type of attack compromises the integrity of the data, in addition to communication links and processes.
- **Repudiation:** A repudiation attack happens when an application or system does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation or forging the identification of new actions [28].
- **Information Disclosure:** This attack involves the exposure of information in an unauthorised way.
- **Denial of Service (DoS):** DoS attacks deny services to authorised users.
- **Elevation of Privilege:** In this type of attack, an unprivileged user obtains unauthorised, elevated access to services or resources.

STEP 3: THREAT RATING

In order to determine the most cost effective method for remediation and mitigation, it is essential to undergo a threat rating. It helps to allocate the necessary resources and effort to the most critical threats. The most effective threat rating is therefore a scaling based risk: the higher the risk of the component, the higher the priority of the threat.

STEP 4: MITIGATION OPTIONS

The aim of this phase is to reduce the risk associated with threats to an acceptable level. Using the results of step 3, it is possible to sort threats from those with the highest risk to the lowest. Once done, developers can prioritize the mitigation actions and choose the best ways to address particular threats.

7. CONCLUSION

In this paper we presented a framework for the secure reuse of software, which consists of a set of recommended steps to follow when reusing components at any stage of the software development life cycle. The framework is neither too specific to the point that prevents it from scaling flexibly according to the implementer's needs, nor too vague such that it becomes non-viable. The paper also presents a threat model that should be followed in order to minimize or eliminate any threats that may appear during software reuse. The methodology and processes included in the proposed framework aim at giving direction to developers and companies wishing to conduct

software reuse in a secure manner. We hope that the utilization of the proposed framework will help organizations adopt a secure software reuse process, and we believe that the steps proposed will allow them to gain confidence and assurance with regards to component reuse. The threat model along with the secure reuse framework proposed in this paper can be applied to many environments in order to develop applications to a high standard of security.

REFERENCES

[1] Mcllroy, M. D., 1968. Mass produced software components. *In Software Engineering; Report on a conference by the NATO Science Committee (Garmisch, Germany, Oct.)*. Naur, P., and Randell, B., Eds. NATO Scientific Affairs Division, Brussels, Belgium, pp. 138-150.

[2] The OWASP foundation, OWASP Top 10 – 2017, The Ten Most Critical Web Application Security Risks, 2017, *https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf*, last date accessed 4 Oct 2018.

[3] Fulton, MD By actively governing the flow of open source components organizations are improving application quality and developer productivity, *https://www.sonatype.com/sonatype-2017-state-of-the-software-supply-chain-report-reveals*, July 2017

[4] Mili, H., Mili, F., Mili, A., 1991. Reusing software: issues and research directions. *IEEE Trans. Softw. Eng*. 21 (6), 528–562

[5] Griss, M.L. Software reuse: from library to factory. *IBM Syst. J*., 32 (4). 1993, 548-566

[6] Krueger, C.W. Software reuse. *ACM Computing surveys*, 24 (2), 1992, 131-183

[7] Jacobson, I., Griss, M., Jonsson, P., 1997. Software reuse: architecture. Process and Organization for Business Success. *ACM Press/Addison-Wesley Publ. Co*., New York, NY, USA

[8] R. J. O. do Nascimento, C. A. G. Fonseca and F. D. M. Neto, Using Expert Systems for Investigating the Impact of Architectural Anomalies on Software Reuse, *IEEE Latin America Transactions, Volume*: 15, Issue: 2, Feb. 2017

[9] Tao Xin; Liu Yang,A Framework of Software Reusing Engineering Management, *IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*Year: 2017,Pages: 277 - 282

[10] N. Krishna Chythanya; Lakshmi Rajamani, Neural Network Approach for Reusable Component Handling, *IEEE 7th International Advance Computing Conference (IACC) 2017*, pp: 75 – 79

[11] T. Rajani Devi; B. Rama, Designing Software Reuse Repository Through Intelligent Classification for Effective Search and Retrieval Mechanism, *Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)* year: 2017, pp: 336 – 341

[12] B. Jalender; A. Govardhan; P. Premchand, Automation of Component Selection Process For Designing of reusable components using Drag and Drop Mechanism, *International Conference on Inventive Systems and Control (ICISC)*, 2017, pp: 1 – 5

[13] Sicong Ma; Hongji Yang; Meiyu Shi, Developing A Creative Travel Management System Based on Software Reuse and Abstraction Techniques, *IEEE 41st Annual, Computer Software and Applications Conference (COMPSAC) 2017*, Volume: 2, pp: 419 – 424

[14] Tihana Galinac Grbac; Per Runeson; Darko Huljenić, Unit verification effects on reused components in sequential project releases, *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp: 74 - 82

[15] Wang Jie; Tian Pei; Shi Wen-qing; Xiao Yan, The Generation of Software Reliability Test Cases Based on Software Reuse, *6th International Conference on Computer Science and Network Technology (ICCSNT)*, 2017, pp: 161 - 164

[16] Breno Mirandaa,b, Antonia Bertolino, Scope-aided Test Prioritization, Selection and Minimization for Software Reuse, *Journal of Systems and Software*, Volume 131, September 2017, pp: 528-549

[17] Doohwan Kim, Jae-Young Choi, Jang-Eui Hong, Evaluating energy efficiency of Internet of Things software architecture based on reusable software components, *International Journal of Distributed Sensor Networks*, Volume: 13 issue: 1, January 19, 2017, https://doi.org/10.1177/1550147716682738

[18] Mohsin Irshad, Kai Petersen, Simon Poulding, A systematic literature review of software requirements reuse approaches, *Information and Software Technology*, Volume 93, January 2018, pp: 223-245

[19] Pacheco, C., Garcia, I., Calvo-Manzano, J. A. & Arcilla, M, Reusing Functional Software Requirements in Small-sized Software Enterprises: *A Model oriented to the Catalog of Requirements, Requirements Engineering*, January 2016, 22(2), DOI: 10.1007/s00766-015-0243-1

[20] Cristina Palomares, Xavier Franch, Carme Quer,Requirements Reuse and Requirement Patterns: A State of the Practice Survey, *Empirical Software Engineering*, December 201622(6):1-44, DOI:10.1007/s10664-016-9485-x

[21] Marcus A. Rothenberger,Kevin Dooley, Uday R. Kulkarni,Nader Nada, Characteristics of Software Reuse Strategies: A Taxonomy of Implementations Patterns, This manuscript is currently under revision for IEEE Transactions on Software Engineering

[22] Anshul Kalia and Sumesh Sood, Concerns in Maintaining Reusable Software Components and the Possible Solutions, *Indian Journal of Science and Technology*, Volume 10, Issue 23, June 2017

[23] Anshul Kalia, Sumesh Sood, Framework For Certification Of Reusable Software Components, *International Journal of Advanced Research in Computer Science*, Volume 8, No. 8, September-October 2017, DOI: http://dx.doi.org/10.26483/ijarcs.v8i8.4667

[24] Sonatype, State of the software supply chain, *Sonatype's 3rd annual report on managing open source components to accelerate innovation*, 2017.

[25] Clem, Monthly News, https://blog.linuxmint.com/?p=2994, Sept 2018

[26] M. Howard and S. Lipner. *The Security Development Lifecycle*. *Microsoft Press*, 2006.

[27] F.Swiderski and W. Snyder. Threat Modeling. *Microsoft Press*, 2004

[28] The OWASP foundation, Repudaition attack, https://www.owasp.org/index.php/Repudiation_Attack, last revision 19/12/2013

Information about the author:

Dr Ahmed Redha Mahlous received his MSc in Computer systems and Internetworking from South Bank University (London, UK) and a PhD degree in Computer Networks from University of Bradford (UK). His area of research includes Network Security, Software Security and QoS. He is currently working as an assistant professor at Prince Sultan University (Riyadh, KSA).

Manuscript received on 22 September 2018