ASPECT-ORIENTED SOFTWARE SECURITY DEVELOPMENT LIFE CYCLE (AOSSDLC)

Aws A. Magableh and Anas M. R. AlSobeh

Department of Computer Information Systems, Faculty of Computer Science and Information Technology, Yarmouk University, Irbid, Jordan

ABSTRACT

Recently, the need to improve the security of software has become a key issue for developers. The security function needs to be incorporated into the software development process at the requirement, analysis, design, and implementation stages as doing so may help to smooth integration and to protect systems from attack. Security affects all aspects of a software program, which makes the incorporation of security features a crosscutting concern. Therefore this paper looks at the feasibility and potential advantages of employing an aspect orientation approach in the software development lifecycle to ensure efficient integration of security. It also proposes a model called the Aspect-Oriented Software Security Development Life Cycle (AOSSDLC), which covers arrange of security activities and deliverables for each development stage. It is concluded that aspect orientation is one of the best options available for installing security features not least because of the benefit that no changes need to be made to the existing software structure.

KEYWORDS

Aspect Orientation, AO, Aspect-Oriented Programming, AOP, SSDL, Software Security Development Life Cycle, Security.

1. INTRODUCTION

The software development life cycle (SDLC) of an information system (IS) consists of four main stages: planning, creating, testing, and deployment. It has also been described as involving a requirement, design, coding, and documentation phase. The SDLC is applicable to a variety of configurations because an IS can comprise just hardware, just software, or both [3]. Given the current global situation and the heightened need for security in both industry and government as well as in personal life, are search area that is growing in importance is the enhancement of the SDLC to include the implementation of the security software development life cycle (SSDLC).

Some of the recent security threats and attack reports can be found in [19] and [20]. A more comprehensive analysis of the exploits, vulnerabilities, and malware based on data from Internet service providers and over 600 million computers worldwide can be found in [1]. Figure 1 illustrates the attacks that focused on applications during the period of 2016. According to [1], "Disclosures of vulnerabilities in applications other than web browsers and operating system applications decreased slightly in first half of 2016, but remained the most common type of vulnerability during the period, accounting for 45.8 per cent of all disclosures for the period."

Dhinaharan Nagamalai et al. (Eds) : CSEN, NCWC - 2018
pp. 33–47, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.81204

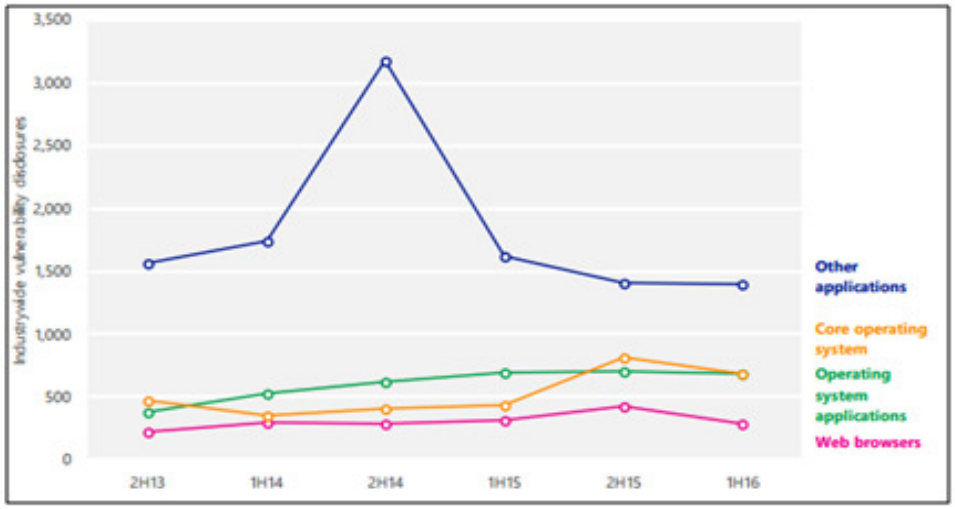

Figure 1: Example of a Microsoft Security Intelligence Report [1]

Thus, it can be said that security is the main requirement of all users, especially those in charge of critical infrastructure. Therefore it is crucial that software vendors address the issue of security threats head on. However, creating software that is ever more secure is a huge challenge [2]. Nevertheless, software vendors must endeavour to do so in order to maintain society's trust in computers in this digital era One of the key steps that software vendors and their collaborators need to take is to shift to a substantially more secure SDLC process that places a greater emphasison security in order to reduce the amount of vulnerabilities in all stages of the process– from requirement to documentation–and that attempts to reduce such vulnerabilities as early in the SDLC as practicable.

The SSDLC helps developers build more secure software and address security compliance requirements. It is created by adding security-related activities to any stage of the software development process by incorporating the concept of aspect orientation (AO) into the SDLC, as shown in Figure 2.

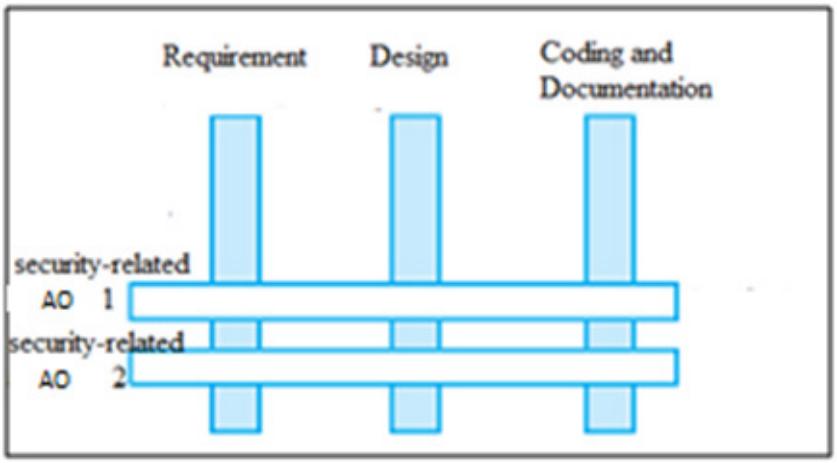

Figure 2: Inclusion of security activities in the SDLC

The two main goals of this study are to identify the techniques currently being used to enhance the security of the SDLC through an in-depth review of the literature and to propose a model to enhance the security of software development. The main objective of this study is to utilize the strength of AO and its concepts to enhance software development security. This work aims also to eject security activity into SDLC with less amount of impact on the standard process of development. The study was guided by two research questions: "What is the practical applicability of existing models for a secure software development life cycle?"and "How can aspect orientation enhance SSDLC?"

The remainder of this paper organized as follows: section 2 provides an overview of the key concepts addressed in this paper. Section 3 explains the methodology used in this research. Section 4reviewsrelated works. Section 5 explains the proposed aspect-oriented software security development life cycle (AOSSDLC) model, and finally, section 6 contains a conclusion and suggestions for future work.

2. OVERVIEW

Before discussing the research methodology, an overview of the key concepts of AO and SSDLC is provided in order to clarify their contribution to the main aim of this study.

2.1 ASPECT ORIENTATION

A research team headed by Gregor Kiczales at Palo Alto Research Center coined the term 'aspect-oriented' when they were developing aspect-oriented programming (AOP) as well as AOP language (AspectJ),a language that is now very popular among developers working in Java [21]. Just as object-oriented (OO) programming [22] before it resulted in a wide range of OO development methodologies [23], AOP has engendered a growing number of software engineering technologies such as AO development methods, modelling techniques that are usually based on the principles of unified modelling language (UML) [24], and assessment technologies to test the effectiveness of AO approaches. Nowadays, the term 'aspect-oriented software development (AOSD)' is used to refer to an array of software development techniques that support the modularization of aspects (also known as crosscutting concerns) throughout an entire software system [25]. This modularization covers requirement engineering, business process management, analysis and design, and programming. Aspect orientation offers a systematic means by which to modularize crosscutting concerns which, as the name implies, has an effect on the other concerns. More often than not, crosscutting concerns cannot be smoothly or completely decomposed from the rest of the system in the design or in the implementation phase, which means that, at implementation, scattered code (code duplication) and/or tangled code (significant dependencies between concerns) can appear. Figure 3 illustrates both of these problems: Figure 3(a) shows how the logging aspect code can get scattered and duplicated in other concerns while Figure 3(b) shows how that same code can become tangled up in one concern.

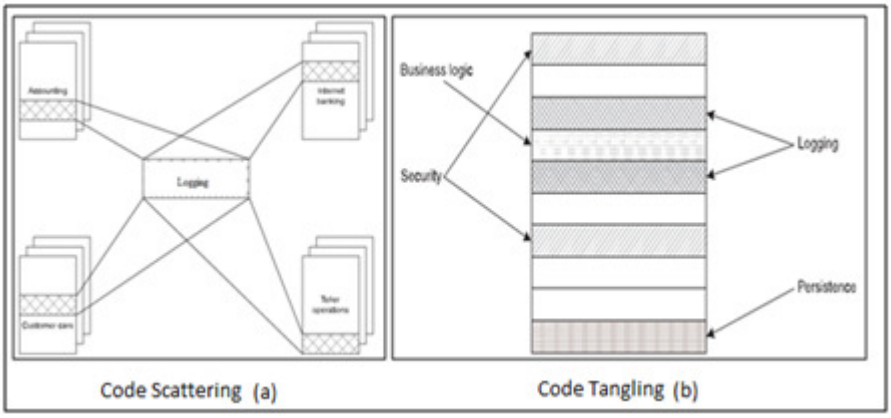

Figure 3: Code scattering and code tangling

2.2 SOFTWARE SECURITY DEVELOPMENT LIFE CYCLE

During the SSDLC the processes of software development are modified by embedding activities that result in enhanced software security. This section summarizes these activities and the ways in which they are incorporated into a SDLC in general terms. Here, the SDLC consists of requirement, analysis, design, and implementation. Also, it should be noted that the modifications are not designed to completely change the developmental process, but to add clear deliverables for software security. Moreover, the software architecture should be designed in such a way that the software is able protect not only itself but the data it processes [2]. Hence it is important that designers assume that security faults will exist in a system and that software should therefore run with the least privileges. Services that are not regularly needed should be disabled by default or made accessible to just a few users or distinct groups of users [2]. Also, tools and guidance should be provided with software at deployment to help users and administrators use it securely, and updates should be easy to deploy. The implementation of security measures in the SDLC is not limited to the above; it needs to be considered in the requirement, implementation, verification and release stages as well.

3. METHODOLOGY

As mentioned in the introduction, this study attempts to answer the following questions:

RQ1. What is the practical applicability of existing models for a secure software development life cycle?

RQ2. How can aspect orientation enhance the SSDLC?

RQ3: What would be the impact of employing AO and its concepts to enhance the security of the SDLC?

In order to find answers to the above questions, we analysed a large number of research papers that were published during the period January 2003to November 2017. The year 2003was chosen as the start date because it was during that year that publications on the security of the software development life cycle began to appear. The papers were collected by using three complementary

search methods in order to achieve the maximum coverage of the domain, as illustrated in Figure4. First, we performed a manual search of the proceedings of several conferences which are particularly relevant to the SDLC. Second, we searched through a number of digital libraries. Third, we performed a snowballing search [34] on the papers collated by the first two search methods. Snowballing involves retrieving papers that are cited by the considered papers (forward snowballing) and papers that cite the considered paper as a reference (backward snowballing). For forward snowballing, we used the bibliographies of the identified papers. Google Scholar was used to perform backward snowballing; Figure 4 illustrates the search process.

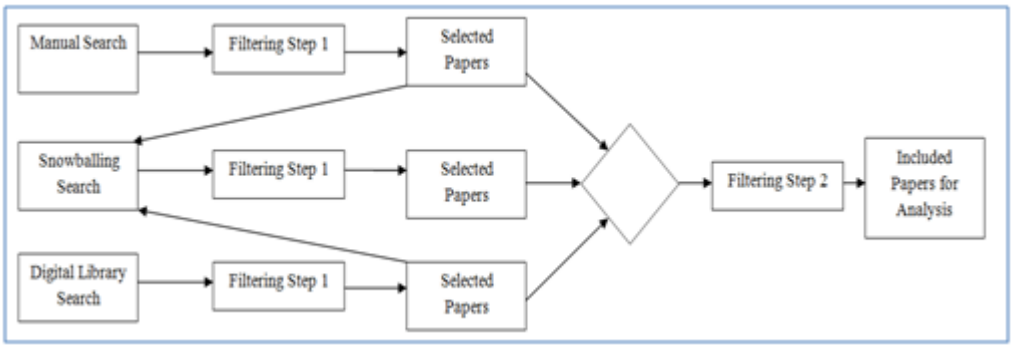

Figure 4: Search strategy

4. LITERATURE REVIEW

Before presenting the proposed model, this section reports the results of our review of literature that was undertaken to discover whether any of the existing models use AO in the SSDLC. Literature review consists of reviewing, extracting and evaluating and then analyzing and interpreting the studies that are relevant to this research. Most research starts with a literature review. However, unless a literature review is fair, it is of little scientific value [4]. With the extended article of this work, we will be adopting a systematic literature review approach [5] we were able to find the best and most-cited works that are relevant to the research question posed by this study, i.e. What would be the impact of employing AO and its concepts to enhance the security of the SDLC? It is worth noting that research on the use of AO to improve the security of the SDLC is quite scarce. This section is exploring major works and categories of works that have been done to employ security in SDL.

4.1 MICROSOFT SECURITY DEVELOPMENT LIFE CYCLE (MSSDLC)

One of the first initiatives in relation to the SSDLC was the MSSDLC proposed by Microsoft, which works in line with the phases of a classic SDLC. Microsoft proposed some of the best security practices to fit with each stage of its classical SDLC, which is shown in Figure 5 [6].

Figure 5: Classical software development life cycle of Microsoft [6]

For the training stage, which is the initial stage of the classic SDLC at the company, Microsoft proposed some core security training to secure the other upcoming stages and focused particularly on the issue of privacy. At the stage of Requirements, Microsoft proposed quality gates and privacy risk assessment to determine on the privacy impact rating. The same thing goes to the other stages, for the design phase, Microsoft proposed quite a few security practices such as threat modelling and attach surface analysis. As for implementation, they suggested using approved tools, deprecating unsafe functions and performing static analysis. For all stages, best practices were applied to ensure the highest possible levels of security. Figure 6provides a summary of the main security activities Microsoft deployed for each SDLC stage [10].

Figure 6: Secure software development process model of Microsoft [10]

4.2 MODEL-DRIVEN ARCHITECTURE FOR THE SECURE SOFTWARE DEVELOPMENT LIFE CYCLE

A model is a crucial component of the design process in many engineering disciplines, including software engineering, as it represents a real system or entity and enables developers to test a proposal or prototype before expending a significant sum on the real thing[10]. To facilitate the software engineering process, the Object Management Group (OMG) developed model-driven architecture (MDA) (The OMG describes itself as an international, open membership, non-profit computer industry consortium and it came into being in 1989.)It is essentially a methodology that helps to specify software system specifications regardless of the hardware and platforms being used for the implementation. In MDA, there are three default models (CIM, PIM, and PSM), as shown in Figure 7 [7]. The concept has been involved in model-driven security (MDS), which can model security requirements at a high level of abstraction.

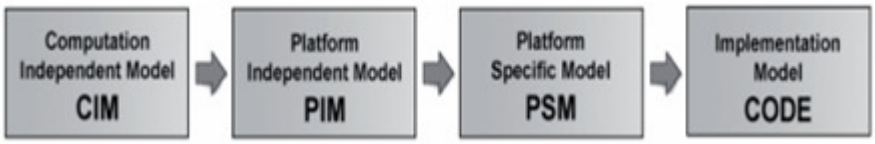

Figure 7: Structure of model-driven architecture [11]

Following the MDA for SDLC proposed in [9], an architecture known as MDA-SDLC was proposed [8], which considers security requirements, security models and system requirements and modelling throughout the SLDC stages. It illustrates the main roles and responsibilities along with technical skills set needed for requirements and software model requirement. At design model, architectural model and pattern model is suggested. The rest of stages contain few more activities suggested at each stage as illustrated in Figure 8.

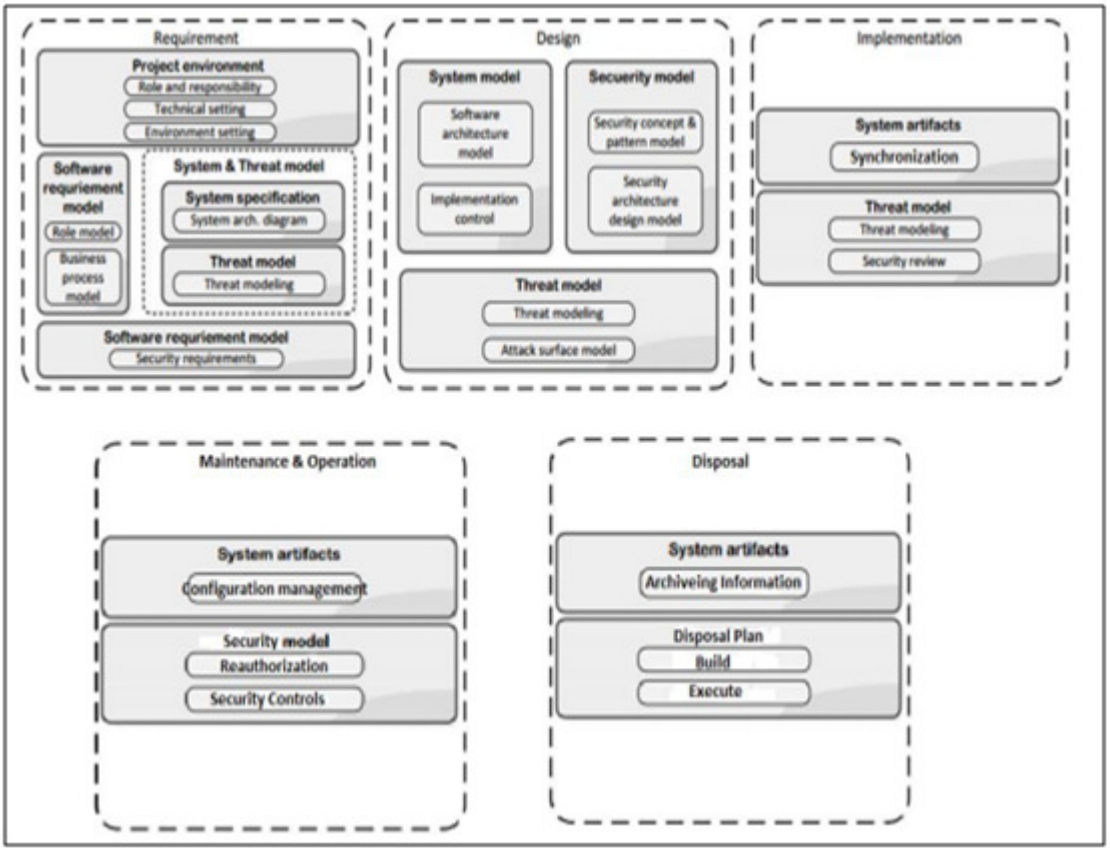

Figure 8: Overview of architecture of MDA SDLC [8]

A number of formal methodologies have been proposed to aid in the development of secure software systems and they cover several different SDL stages. The methodologies are mathematically based on specifications that represent software system behaviours. The specifications themselves employ a formal syntax and can be used to garner key information about a software system. Developers can produce software programs in a formal manner by using a formal methodology [7]. An illustration of a typical formal methodology is provided in Figure 9.

With respect to the SDLC, there are two types of formal methodology that can be employed: the software security assessment instrument method and the construction method. The first type involves the development and use of tools and a variety of information resources to ensure the security of software, for example by using model checkers. The second type uses a range of formal methodologies for the entire SLC, including formal description language for the system

specification and unequivocal programming language and bug prevention fixes, which enables analysis of the software to be very stringent in even the earliest stages of software development.

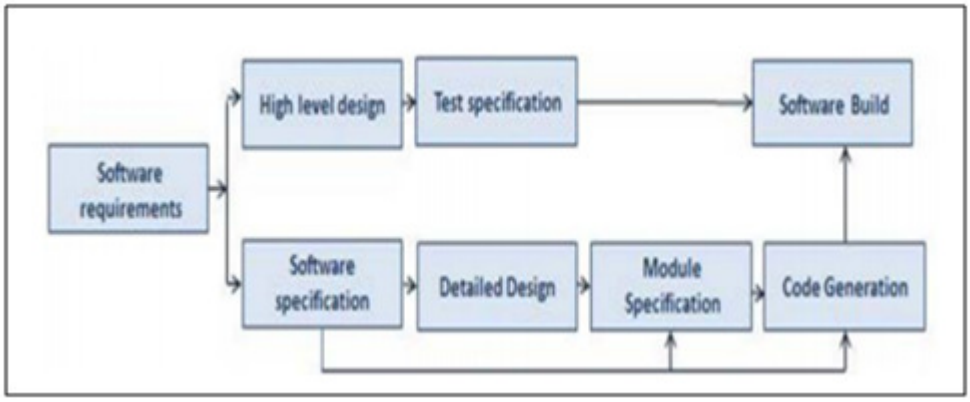

Figure 9: Application of typical formal methodology to SDL

4.4 ASPECT-ORIENTED MODELLING FOR REPRESENTING AND INTEGRATING SECURITY CONCERNS IN UML

This study suggests a new way of specifying security aspects in UML and, moreover, it enables security aspects to be systematically and automatically woven into UML. The main aim is to identify and deal with security-related concerns and model them during the design stage of the SDLC. To do this, it uses the class diagram as one of the UML design diagrams. Figure 10 shows the security aspect by using the class diagram.

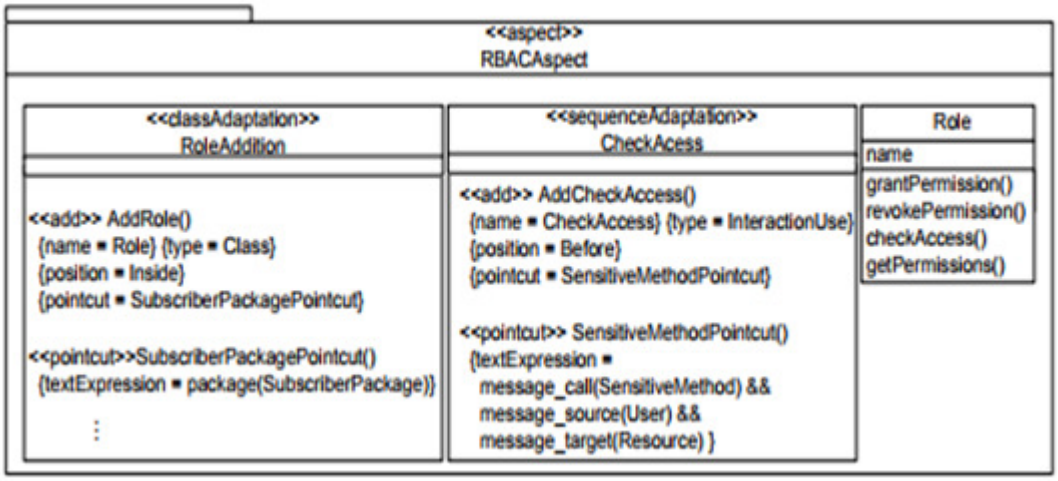

Figure 10: Class diagram of security aspect

Other approaches, such as [14], implement a risk management analysis in order to incorporate security into the SDLC. Other related works such as [27-32] have attempted to improve security by using AOP at the SDLC implementation stage. Moreover, among these works [27] and [28] have also proposed a method to integrate security using AOP at the implementation stage, while [29] and [30] have investigated aspectizing security at the programming stage. Additionally, [31]

and [32] have considered using AOP only during the programming stage to ensure that the system is trustworthy during the development process. Generally, less attention has been given to utilizing the benefits of AO and its related concepts for other (earlier) SDLC stages as a means to improve the security of software. However, it makes sense to employ AO as early as possible in the SDLC, otherwise it might be too late to address all the security dimensions.

5. ASPECT-ORIENTED SOFTWARE SECURITY DEVELOPMENT LIFE CYCLE

This section focuses on describing the architecture of the AOSSDLC proposed in this paper. As mentioned in the introduction, finding ways to ensure the highest level of security during the development of complex software systems is now more critical than ever because software now pervades almost all aspects of our lives both professional and personal. Aspect orientation has been shown to be effective in dealing with the crosscutting nature of security requirements so it could be particularly useful not only when developing and designing applications but also when implementing them. In our work, we aim to bring different fields together to discover whether the AO concept can be successfully utilized to improve SDLC security and consequently reduce security-related attacks and vulnerabilities. Figure 11shows how we see the various fields linking together.

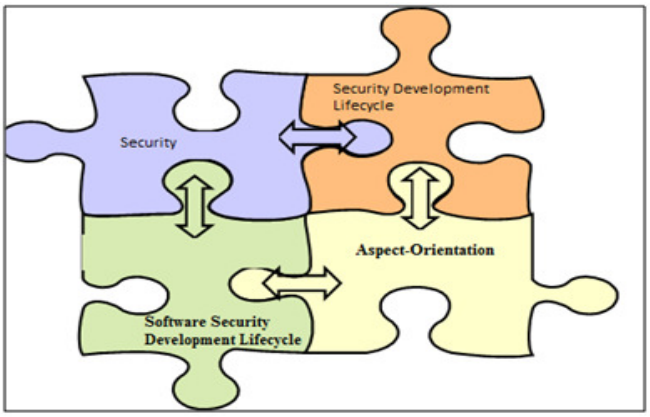

Figure 11: AOSSDLC areas of collaboration

Perhaps the major benefit of using AO is that it can weave any kind of crosscutting concern, including security and security-related concerns, into a system, even if they are scattered and tangled [26] throughout the system, without having an adverse effect on other concerns. Moreover, this weaving process can occur at any stage in the SDLC. In other words, adding or removing aspects in any stage of the SDLC becomes less problematic and less time consuming [15][17]. One more dimension to consider which motivated this work is that, the evolving field of securing SDLC [1] [8]. Having said that, it was essential to investigate and explore the ability to connecting these topics with each other to come up with relativity reliable secure software development life cycle depending on AO. Our proposed model, the AOSSDLC combines these advantages and addresses these issues. Figure 12 illustrates the application areas of the proposed model.

In the design, development and implementation of a secure system the security-related properties in the system should be abstracted out of the main system to improve clarity, maintainability, manageability and reuse [18]. Also, where legacy source code has identified or potential security

vulnerabilities the code should be patched by adding the smallest possible amount of new code and ideally the original code should not be changed. In addition, where appropriate, it should be possible to reuse security-related properties in a range of applications [28]. It is noteworthy that all the above can be achieved by using AO [18] because AO automatically checks for errors in security-sensitive calls, automatically logs data on security concerns, replaces generic code with secure code and specifies privileges, abstracts some concerns, replaces concerns with the minimum changes necessary and its changes are reusable in any stage of the SDLC. Thus, it becomes clear why our model is designed in accordance with the concept of AO in order to attempt to integrate the above mentioned highly desirable security-related activities into the SDLC. In AO there are two kinds of crosscutting concerns (aspect concepts) that applicable to any stage of the SDLC: dynamic crosscutting and static crosscutting. They can both be utilized to embed security-related crosscutting concerns into any part of SDLC. These types of crosscutting are described in brief in the following subsections.

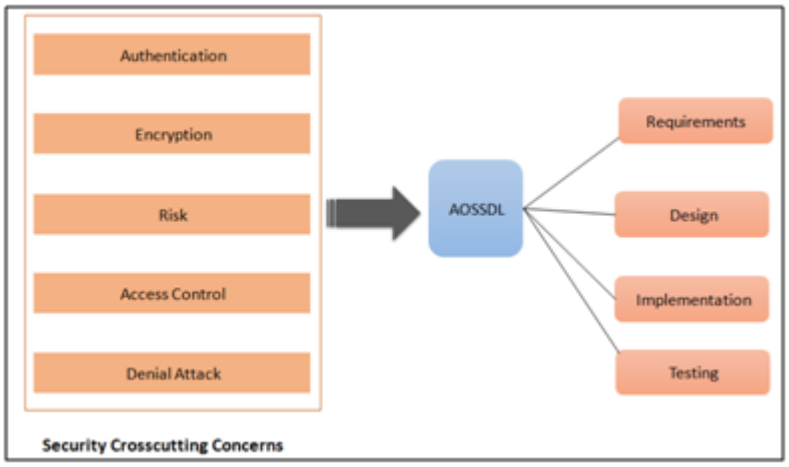

Figure 12: Application areas of proposed AOSSDLC model

5.1 DYNAMIC CROSSCUTTING OF AOSSDLC

Dynamic crosscutting is a technique that allows points to be defined and changes (pieces of code) to be recommended in the SDLC coding stage of the dynamic execution of a program. Our proposition extends this dynamic crosscutting technique to other SDL stages. Our proposed model utilized the dynamic crosscutting sub concepts join point, pointcut, and advice not only in the programming stage but in all the other SDLC stages as well in order to include dynamic security-related crosscutting concerns in the SDLC. In the AOSSDLC model, join points are predictable points in the execution of a program, and they are the points at which security activity must be added at a specific SDLC stage to ensure the security of the software. It is the pointcut in the AOSSDLC model that is designed to identify and select the join points where the security activity needs to be added. When the model gives advice this refers to the model identifying the actual security activity that needs to be injected and executed when a join point is reached. Figure 13 illustrates the proposed AOSSDLC model.

Join points are predictable point in the execution; it would represent the point where we would need to add the security activity at a specific SDL stage. Pointcut of AOSSDL is designed to

identify and select join points where the security activity will be added. Advice of AOSSDL is the actual security activity to be injected and executed when a join point is reached.

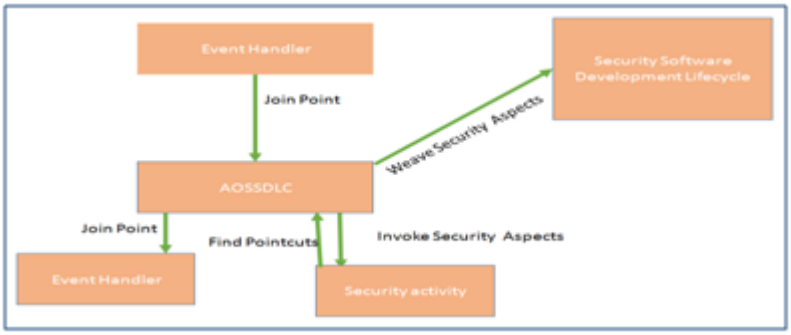

Figure 13: AOSSDLC model

5.2 STATIC CROSSCUTTING IN THE AOSSDLC

Static crosscutting is an inter-type declaration that can add attributes and/or methods to an existing structure. It is a powerful technique because it provides the developer with the capability to add new attributes and operations to a class or aspect, as well as a whole range of other declarations that affect the static-type hierarchy. In our proposed model, we use it to add attributes to a specific classification in the SDLC.

5.3 ASPECT WEAVING STEP

In this model, it is the weaving of the security aspects into the SDLC that makes the SDLC secure. This is done mainly by the following generic steps:

1) Locating the security join points, which involves identifying the locations at which the SDLC stage/activity and the security requirements/design aspects interact; analysing the vulnerabilities of and the threats posed to the software based on the security requirements and security design; and specifying the security join point setting for the connectors in the SDLC.

2) Constructing security advice, for which actions are defined in order to enforce security in the required SDLC stage through locating the join points that have the same vulnerability and grouping them together as a pointcut.

3) Weaving the security aspect into the SDLC in order to incorporate the security aspect into the SDLC, this involves systematically searching for join points so that the security advice/aspect can inject the required security behaviours into the SDLC in the correct places.

From a comparison of the proposed model with those in the literature, it would seem that the proposed model is better structured because not only does it have clear steps for defining changes in security at specific points in the SDLC stages, it also contains a weaving step to enable the injection of aspect changes. Moreover, the proposed model is based on a bottom-up technique

that maps the AOP elements (such as AspectJ) so that they can be embedded into the early stages of the SDLC.

5.4 APPLICATION OF THE AOSSDLC

The AOSSDLC model shows how the aspects are used and woven to inject a change and a modification at any stage of the SDLC without the need to work manually on the change. Figure 14 illustrates how AO is used in each stage.

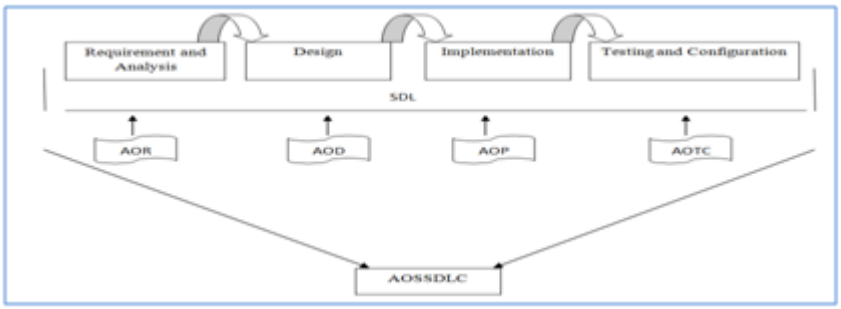

Figure 14: AO in the SDLC stages

Due to the limitation of space, here we illustrate how AO can be used in the requirement stage of the SDLC only. If the requirements have already been elicited from the clients, no major changes should need to be made to the actual requirements. Where a change does need to be made to a specific part of the natural text/requirement this will result in a change to another requirement because all requirements are connected and traceable. When there is a need to make such a change manually, the AOSSDLC model suggests utilizing the dynamic concept and structure of the aspects and aspect weaving, as shown in Figure 15.

```
Aspect RequirementChangeAspect {
        At Poincut RequirementChange (InitiateRQChangeJP _RQx): Execution
        (RequirementChangeAspect.Begin(..) && Args (RQx) )
        BeforeAdvise (InitiateRQChangeJP) {
                Check and add any prerequisite before changing and amending
                requirement/s
                Proceed (_InitiateRQChangeJP)}
        AroundAdvise (InitiateRQChangeJP) {
                Check and add roles of amending the requirement/s
                Proceed (_InitiateRQChangeJP)
                }
        AfterAdvise (InitiateRQChangeJP) {
                Add the rules of after changing and amending requirement/s
                Proceed (_InitiateRQChangeJP)
                }
}
```

Figure 15: Example of using of aspect orientation to make a change at the requirement stage of the software development life cycle

In light of the above discussion, we believe that this study was able to achieve its main objectives of identifying what models are currently being used to secure the SDLC and how AO can be used to make the software development process more secure.

6. CONCLUSION AND FUTURE WORK

This paper proposed an AO-based model for embedding security activities in the SDLC. Our ultimate aim is to develop a model that is practical and extensible for different SSDLCs and research projects. Our next step is to apply the AOSSDLC model to some real-life case studies, which will help us in assessing its performance in terms of how it deals with threats, the nature of its limitations, and its potential for scalability. The results, outcomes and feedback will be used to enhance the model and improve its feasibility and, consequently, promote its usage. We also intend to investigate the usage of AO in the so-called agile SDLC because this type of development life cycle has less stringent guidelines for the initial stages of development and then adjustments are made as and when needed throughout the remainder of the process, which is AO kind of behaviour where it does not affect any other processes and stages.

REFERENCES

[1] Microsoft. Microsoft security intelligence report: Julydecember 2016. http://www.microsoft.com/technet/ security/default.mspx, 2016

[2] Http://msdn.microsoft.com/en-us/library/ms995349.aspx#sdl2_topic1_1, retrieved July 2017

[3] Selecting a Development Approach, Retrieved May 2017.

[4] Ebse, B. & Charters, S. 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering.

[5] Maryati, Y. 2011. Systematic Review Hand-on Workshop. Research Center of Software Technology and management, Faculty of information and Technology, UKM.

[6] Howard, M., &Lipner, S. (2006). The security development life cycle (Vol. 8). Redmond: Microsoft Press.

[7] Sasikala D, The Most Common Methodologies for Secure Software Development, International Journal of Multidisciplinary Research and Development, Volume 3; Issue 3; March 2016; Page No. 194-197

[8] Muhammad Asad, Shafique Ahmed, Model Driven Architecture for Secure Software Development Life Cycle, International Journal of Computer Science and Information Security (IJCSIS), Vol. 14, No. 6, June 2016

[9] Zhendong Ma, Christian Wagner, Arndt Bonitz, and Thomas Bleier , Model-driven Secure Development Life cycle, International Journal of Security and Its Applications Vol. 6 No. 2, April, 2012

[10] Microsoft, Security Development Life cycle SDL Process Guidance Version 5.2, May23, 2012

[11] F. Truyen, "The Fast Guide to Model Driven Architecture; The Basics of Model Driven Architecture," Cephas Consulting Corp , 2006

[12] Eoin Keary, Jim Manico, "Secure Development Life cycle," in OWASP, Hamburg

[13] Richard Kissel, Kevin Stine , Matthew Scholl , Hart Rossman , Jim Fahlsing , Jessica Gulick, "Security Considerations in the System Development Life Cycle," National Institute of Standards and Technology , Gaithersburg, 2008 .

[14] Bart De Win , Riccardo Scandariato, Koen Buyens, Johan Gre´goire, WouterJoosen, On the secure software development process: CLASP, SDL and Touchpoints compared, Information and Software Technology 51 (2009) 1152–1171

[15] Georg, G., Ray, I., and France, R., "Using Aspects to Design a Secure System", In Proc. 8th IEEE Int'l Conf. on Eng. of Complex Computer Systems (ICECCS'02), pp. 117-128, 2002.

[16] Shah, V. and F. Hill, "An Aspect-Oriented Framework", In Proc. DARPA Info.Survivability Conf. and Exposition (DISCEX'03), pp. 22-24, 2003.

[17] Yu, H. et.al., "Secure Software Architectures Design by Aspect Orientation", In Proc. 10th Int'l Conf. on Eng. of Complex Computer Sys (ICECCS'05), pp. 45-57, 2005.

[18] Viega, J., Bloch, J.T., and P. Chandra, "Applying Aspect Oriented Programming to Security", In Cutter IT Journal, 14(2):31-31, 2001.

[19] Symatec, Internet Security Threat Report, Volume 21, April 2016

[20] Cisco, Cisco Annual Cybersecurity Report 2017 Infographic

[21] Schwanninger, C., &Joosen, W. (2011).Transactions on Aspect-Oriented Software Development VIII (Vol. 6580). S. Katz, & M. Mezini (Eds.). Springer Science & Business Media.

[22] Smith, B. (2015). Object-Oriented Programming. In Advanced ActionScript 3(pp. 1-23). Apress.

[23] Avison, D., & Fitzgerald, G. (2003). Information systems development: methodologies, techniques and tools. McGraw Hill.

[24] Magableh, A., Shukur, Z., & Ali, N. M. (2013). Aspectual UML approach to support AspectJ

[25] Filman, R., Elrad, T., Clarke, S., &Akşit, M. (2004). Aspect-oriented software development. Addison-Wesley Professional.

[26] Groher, I., &Voelter, M. (2007, March). XWeave: models and aspects in concert. In Proceedings of the 10th international workshop on Aspect-oriented modeling (pp. 35-40).ACM.

[27] B. De Win, B. Vanhaute, B. De Decker, "Security through Aspect-Oriented Programming", Advances in Network and Distributed Systems Security, pp. 125-138, 2001.

[28] J. Dehlinger and N. V. Subramanian.Architecting Secure Software Systems Using an Aspect-Oriented Approach: A Survey of Current Research. Technical report, Iowa State University, 2006

[29] De Win, Bart, WouterJoosen, and Frank Piessens."Developing secure applications through aspect-oriented programming." Aspect-Oriented Software Development (2005): 633-650.

[30] Marchand de Kerchove, F., Noyé, J., &Südholt, M. (2013, March). Aspectizing javascript security.In Proceedings of the 3rd workshop on Modularity in systems software (pp. 7-12).ACM.

[31] Safonov, V. O. (2008). Using aspect-oriented programming for trustworthy software development (Vol. 5).John Wiley & Sons.

[32] Mourad, A., Laverdière, M. A., &Debbabi, M. (2008). An aspect-oriented approach for the systematic security hardening of code. computers & security, 27(3), 101-11

[33] Mouheb, D., Talhi, C., Nouh, M., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML. In: R.Y. Lee, O. Ormandjieva, A. Abran, C. Constantinides (eds.) Proceedings of the ACIS Conference onSoftware Engineering Research, Management, and Applications, Studies in Computational Intelligence, vol. 296,pp. 197–213. Springer (2010)

[34] A. van den Berghe, R. Scandariato, K. Yskout, W. Joosen, "Design notations for secure software: a systematic literature review", Software & Systems Modeling, 2015.