2011

Security-Based Risk Assessment for Software Architecture

Fadi Haj Said
West Virginia University

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Haj Said, Fadi, "Security-Based Risk Assessment for Software Architecture" (2011). *Graduate Theses, Dissertations, and Problem Reports*. 4728.
<https://researchrepository.wvu.edu/etd/4728>

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Security-Based Risk Assessment for Software Architecture

Fadi Haj Said

**Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of**

**Doctor of Philosophy
in
Computer Engineering**

**Hany H. Ammar, Ph.D., Chairperson
Yousef Hassouneh, Ph.D.
Matthew C. Valenti, Ph.D.
Arun Ross, Ph.D.
Hong Jian Lai, Ph.D.**

Lane Department of Computer Science and Electrical Engineering

Morgantown, WV 26505

2011

**Keywords: Attack Graph, Probability of security failure, Security risk factor,
Severity of security failure, Software Architecture.**

Copyright 2011

Fadi Haj Said

ABSTRACT

Security-based Risk Assessment for Software Architecture

Fadi Haj Said

Enterprise, medium and small companies develop and maintain different types of large-scale software systems for public and financial institutes. A security failure in such systems could lead to a significant impact. It is therefore essential to analyze security risk in the system components early on during the development process in order to prevent the occurrence of security failures.

The objective of this research is to develop a methodology for security risk assessment during the early software development phases to identify high security risk elements, thus enabling us to enhance security features in the early phases of software lifecycle. This methodology can be integrated with other methodologies for assessing risk in software attributes such as performance, reliability, and maintainability in order to enhance resource allocation decision and to improve the quality of software products.

We propose an architectural level security assessment methodology to assess the security risk of software systems early on in the software life cycle. It combines the probability of security failure and the severity associated with such failures to estimate the risk factors. This research presents first a methodology that uses UML specifications to estimate the probability of security failure for each component for a given scenario of a given

use case. The research also proposes a new methodology to estimate the severity of security failure based on the data sensitivity, access rights, and reachability matrix. These methodologies define a security risk assessment model that enables us to identify high security risk elements. We illustrate this model using a scenario of an ecommerce case study.

Validation of security risk assessment is a challenging and a relatively unexplored area. In this research, we conduct two validation studies of the proposed methodologies based on Microsoft Security Bulletins and two security design patterns.

DEDICATION

I dedicate my dissertation work to my family. Special feelings of gratitude are to the memory of my great father, loving mother, patient wife, wonderful sons, two brothers, sisters, every member of my family, and to all those who help me in this work.

ACKNOWLEDGEMENTS

I would to thank my committee members who were more than generous with their expertise and precious time. Special thanks are to Dr. Hany Ammar, my committee chairman for his countless hours of reflecting, encouraging, and most of all patience throughout the entire process. I would like to thank Dr. Yousef Hassouneh for his ever presence guidance during this research effort. I wish to thank Dr. Matthew Valenti, Dr. Arun Ross, and Dr. Hong Lai for their valuable advices. I would like to thank Microsoft teams and managers who have helped me through all these years. I would like to thank Pearson support team.

Table of Contents

ABSTRACT.....	ii
DEDICATION.....	iv
ACKNOWLEDGEMENTS.....	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
Chapter 1	1
Introduction.....	1
1.1 Overview:	1
1.2 Background:	2
1.2.1 Software Architecture:.....	3
1.2.2 The Unified Modeling Language	4
1.2.3 Software Security	7
1.2.4 Security- Based Risk Assessment:	8
1.3 Dissertation Organization.....	10
Chapter 2.....	11
Problem Statement and Research Objectives	11
2.1 Problem Statement:	11
2.2 Research Objectives	12
2.3 Research Contribution.....	14
Chapter 3.....	15
Literature Review.....	15

3.1	Software Security metric and security threat modeling:	15
3.2	Surface Attack Metric:	23
3.3	Attack Graphs:.....	26
3.4	Security Risk Assessment:	27
3.5	Summary:	28
	Chapter 4.....	30
	Proposed Techniques and Methodologies	30
4.1	Introduction	30
4.2	Security Based Risk Assessment:	30
	4.2.1 Probability of Security Failure	32
	4.2.2 Severity of Security Failure	38
	4.2.3 Security Risk Factors.....	41
4.3	Case Study.....	42
4.4	Conclusions and future work:	60
	Chapter 5.....	61
	Validation Studies.....	61
5.1	Introduction:	61
5.2	Validating Software Measures and Prediction	63
	Systems.....	63
	5.2.1 Validating a Software Measure	63
	5.2.2 Validating a Prediction System	65
5.3	Validation based on Sensitivity Analysis	67

5.3.1 Vulnerability Index analysis:	68
5.3.2 Initial Probabilities Analysis:	69
5.3.3 Severity analysis:	72
5.3.4 Risk factors analysis:	75
5.4 Security Design Patterns	78
5.4.1 Example1: Secure Communication Pattern:	79
5.4.2 Example2: Policy Pattern:	88
5.4.3 Integrating Security Risk Assessment with other quality attributes:	93
5.5 Microsoft Security Bulletin Analysis	96
5.4 Conclusions	104
Chapter 6	105
Conclusion and Future Work	105
References	108

LIST OF FIGURES

Figure 4- 1: an example of an attack graph $\{c1, c2\}$ is set of initial nodes. $\{c3\}$ is the set of intermediate nodes, cg goal node, $\{t1, t2, t3\}$ set of exploit nodes	34
Figure 4- 2 The security risk analysis algorithm	35
Figure 4- 3 Tree Diagram for a component j	38
Figure 4- 4 The security severity analysis algorithm.....	41
Figure 4- 5 E-commerce system architecture	43
Figure 4- 6 UML Sequence diagram of a buy book scenario	49
Figure 4- 7 Attack Graph of customer agent component in a given scenario.....	52
Figure 4- 8 Attack Graph of customer information database component.....	58
Figure 5- 1 Probabilities of security failures of customer agent and customer information database components when VI changes.....	69
Figure 5- 2 severities of security failures of e-commerce components with/without admin rights	74
Figure 5- 3 risk factors of software components in ecommerce example with admin rights, without admin rights, $VI=.1$	74
Figure 5- 4 Security risk factors for three components, customer agent, customer interface, and bookserver database with 9 cases, $VI=.1$	75
Figure 5- 5 The distributions of risk factors for the three components, customer agent, customer interface, and bookserver database.....	78
Figure 5- 6 Structure of Secure Communication Pattern.....	80

Figure 5- 7: UML sequence diagram without any security patterns.....	82
Figure 5- 8: Attack Graph of customer agent without any security patterns	83
Figure 5- 9: UML sequence diagram of secure communication pattern	84
Figure 5- 10: UML sequence diagram of customer agent after adding secure communication pattern.....	86
Figure 5- 11: Attack Graph of customer agent after adding a security communication pattern	88
Figure 5- 12: Structure of Policy Pattern	89
Figure 5- 13: UML sequence diagram of Policy Pattern	92
Figure 5- 14 The architecture of the Software Architecture Risk Assessment (SARA) Tool.....	95
Figure 5- 15: UML sequence diagram with MSB04-012 vulnerability and policy pattern	100
Figure 5- 16: UML sequence diagram with MSB04-023 vulnerability and secure communication pattern.....	103

LIST OF TABLES

Table 4- 1 Probabilities, severities and risk factors of all components in buy a book scenario	59
Table 5- 1 Summary of validations methods	67
Table 5- 2 Numbers of violation from MSB -2004	71
Table 5- 3 Initial probabilities values for confidentiality, integrity availability security violations.....	71
Table 5- 4 Initial Probabilities values of the case study e-commerce example	72
Table 5- 5 security risk factors of customer agent component before /after adding secure communication pattern.....	87
Table 5- 6 security risk factors of customer agent and customer information database components before/after adding policy pattern.....	93
Table 5- 7 Comparison between MSB on code level and its map on architectural level .	97
Table 5- 8 Security Risk Factor before and after applying patches	98
Table 5- 9: Vulnerability of MSB04-012 and its maps on the architectural level	99
Table 5- 10: Vulnerability of MSB04-023 and its map on the architectural level	102

Chapter 1

Introduction

1.1 Overview:

Stable software architecture is the key to build a software system with high quality attributes. Software architecture clarifies the structure of the system in terms of components and interactions among them to accomplish the desired requirements (Components and Connectors). The software architecture [51][7] for concurrent and distributed applications can be designed by means of component and connectors. The components address the functionality of a system, whereas connectors deal with communications between components in that system. Furthermore, it supports many software development paradigms such as COTS-based software development, product line engineering and component based software engineering. In [50], Shaw was the first to promote the shift to architectural view from functional view of software development, and that has been adopted widely since. As architecture became a more significant artifact in developing software systems, the need to quantitatively analyze them has become notable. The architecture quantitative analysis should reflect their related quality attributes and help us to predict the quality of the software products instantiated from it.

According to NASA-STD-8719.13A standard [38] risk is a function of the anticipated frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity.

This standard defines several types of risk, such as for example availability risk, acceptance risk, performance risk, cost risk, schedule risk, and security risk etc. Software risk management concentrates on developing a product with better quality attributes such as security, reliability, performance, .. etc. and the uncertainty associated with the product development. It helps project managers in avoiding unpredicted catastrophic problems. Also, it prevents wrong allocation of resources and taking decisions without proper knowledge or adequate information on anticipated future consequences [48]. To manage software development projects, managers and developers should rely on processes, methods and tools to facilitate assessment, prioritization and mitigation of various risk aspects. Therefore, risk assessment is an essential part in the management of software development.

In this research effort, our focus will be on the security-based risk assessment of software architecture. Security-based risk takes into account the probability that the software product will fail in the security and the consequences of that failure. In other words, the risk will take into account the probability of security failure based on attack-ability and the results of attack-ability on a software system.

1.2 Background:

This dissertation is related to the security area in the field of software engineering. The main theme works around security-based assessment for software architecture modeling using Unified Modeling Language (UML), and Attack Graphs. The following

sections give a basic background on the recent work in software architecture, unified modeling language, software security.

1.2.1 Software Architecture:

Abstracting the software system to highest level obtains us its architecture. As the size and the complexity of the software systems increase, the need for structuring and organizing it into components and connectors increases. As a result, the discourse of software system's architecture becomes essential [48]. Software architecture is the centerpiece of modern system development [55]. The goal of architecture-centric development is the effective, efficient, competitive development of software products. The goal of software architecture is an important asset because of the following:

- The architecture can be used for communication purposes, as it provides an understandable abstraction by stakeholders, not only software developers but also users and managers.
- Early in the development process of new software, architecture can be available for early analysis of the system's properties.
- For the evolution of existing systems, they can be analyzed at the architectural level to provide a foundation for further development.

1.2.2 The Unified Modeling Language

As software systems become more complex, modeling them to guide development or to help secure becomes essential. System models are used to document the analysis and design and to communicate the system artifacts among the developing and security teams. Therefore, to have a modeling language standard is an important factor for the success of an application development. The Unified Modeling Language UML has become the de-facto standard for building Object-Oriented software. UML unified the efforts of [3] [45][23] [56]. That effort has matured into UML becoming an OMG (Object Management Group) standard [40]. Adopting UML as a standard is motivated by:

- It is programming languages independent.
- It provides a rich language for visual modeling to develop and communicate meaningful models.
- It integrates lots of efforts over the years and blends many models developed.
- It provides the means to extend and specializes the core concepts.

1.2.1.1 UML Definition:

According to OMG specification: [40]

"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

[17] Fowler defines UML as a family of graphical notations, backed by single meta model in describing and designing software systems, particularly software systems built using the object oriented (OO).

It is important to note that UML is a 'language' for specifying and not a procedure or method. The UML is used to define a software system; to detail the system artifacts, to document and construct. It is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology but in itself it does not specify that methodology or process.

1.2.1.2 UML Models

A single model cannot capture static and dynamic system properties. The models used influence how to tackle the problem and how to come up with an appropriate solution. Therefore, complex systems should be analyzed by examining independent views. Static models define the static architecture of the system. They are used to model the elements that make up a system - the classes, objects, interfaces and physical components. Furthermore, they are used to model the relationships and dependencies among the elements of the system. Class diagrams, Package diagrams, Component diagrams, and Deployment diagrams are some of the static views of the system. Dynamic models define the interaction among the system elements to accomplish a system behavior. They contain events, responses, messages, and invocations. Use-Cases, Scenario and Collaboration diagrams, and State-charts diagrams are some of the dynamic views of the system. The following summarizes the modeling diagrams supported by UML [56]:

- Use Case diagrams: describe the boundary and interaction between the system and users. They conform in some regards to a requirements model. A use case designates a situation in which the system is used. It defines the system inputs, actions and possible outputs. Use cases are analyzed to construct possible scenarios.
- Class diagrams: A class diagram defines the basic building blocks of a model: the types, classes and general materials that are used to construct a full model. They depict possible classes and their relationships. Details of the design are communicated through detailed class diagrams, which include the attributes and the methods of the classes.
- Package diagrams: are used to divide the model into logical containers or 'packages' and describe the interactions between them at a high level.
- Behavior diagrams: capture the varieties of interaction and instantaneous state within a model as it 'executes' over time.
- State-chart diagram: State-charts are used to model the behavior of complex systems.
- State charts describe the states or conditions that classes assume over time
- Interaction diagrams: They include sequence diagrams and collaboration diagrams

- Sequence diagrams: show the sequence of messages passed among objects using a vertical timeline. A sequence diagram reflects a scenario of interactions in the system to manifest a use case of the system. Normally, there are one or more scenarios for each use-case.
- Collaboration diagrams: is another view of scenarios. They show the network and sequence of messages between objects at run-time during a collaboration instance.
- Component diagram: are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well-defined interface.
- Deployment diagram: show the physical disposition of significant artifacts within a real-world setting. Deployment diagrams are related to component diagrams in that a node typically encompasses one or more components.

1.2.3 Software Security

According to IEEE journal [33], software security is defined as follows:

“Software security is the idea of engineering software so that it continues to function correctly under malicious attack. Most technologists acknowledge this undertaking’s importance, but they need some help in understanding how to tackle it.”

Security could be built at all levels and more specifically should be integrated at the design and architecture level. At the design and architecture level, a system must define a unified security architecture that takes into account security features (access rights).

Architects must clearly document assumptions and identify possible attacks on the systems and security *risk analysis* is a mandatory.

Software security is to build secure software, design secure software, and educate software developers, architects, and users how to build secure things. It is the process of designing, building, and testing software for security by identifying and problems in the software itself and that consequently will allow the software security practitioners to build and use software that can resist attacks proactively.

There are many reasons why software security must be part of a full lifecycle approach; one important reason is to consider security as an emergent property of software system. A secure problem will show up because of a problem in a standard-issue part of the system.

1.2.4 Security- Based Risk Assessment:

Risk assessment is an essential part in the management of software development. Performing it in the early phases of software development can enhance allocation of resources within the software lifecycle. Also, it provides useful means for identifying potentially troublesome software components that require careful development and allocation of more testing effort. We are concerned with security-based risk of a software architecture, which takes into account the probability of the software security failure and the consequences of that failure.

Popstojanova [43] proposes a methodology for risk assessment using UML specifications such as use cases, sequence diagrams, and states diagrams that can be used in the early phases of the software life cycle. The risk methodology used in that paper depends entirely on the analytical methods by explaining two real life examples. UML

sequence diagrams are used to estimate the components and connectors dynamic risk factors. Then discrete Time Markov Chain is built for estimation of each scenario risk factor.

Manadhata [32] introduces the notion of a software systems' attack surface and present a systematic way to measure it. The attack surface measurement method in this paper is analogous to the common risk estimation method. However the authors use the systems resources (Methods, Channels, and Data) and damage-effort potential to estimate the attack surface measurement on a system. In this paper, a couple of real life scenarios have been used to measure the attack surface. Their approach applied on the small desktop applications and large enterprise systems implemented in C and Java.

Howard [22] introduces a metric to compare if one system is more secure than other systems with respect to a fixed set of dimensions. Howard et. counts the system attack opportunities in terms of three dimensions (targets and enablers, channels and protocols, and access rights). They call their metric "*attackability*".

Feng [14] proposes a flexible approach by building attack graphs, and they use the graph to measure security of crucial resources in the network. Their approach represents the network as a graph with a set of nodes and edges. They set one node as a goal node and they calculate the risk on that node by computing probabilities that lead to the goal node.

1.2.4.1 Probability of security failure:

Risk is defined as combination of probability of malfunction (failure) and the consequence of that malfunction (severity). During the early phases of the software life cycle, it is difficult to estimate the probability of security failure of software components,

however knowing that the components and connectors are sources of vulnerabilities, we use attack graphs as a probabilistic argument to estimate the probabilities of security failures. The Exploited messages conveyed over the connectors. And the intermediate components could be the path to reach to the goal component.

1.2.4.2 Severity of security failure:

Traditional software fault detection models do not take into account the fact that the consequences of various software failures caused by faults can be very different [5]. Thus, they are of limited use in the allocation of resources to the portions of a system with the greatest risk. Since software failures have different consequences, any measure of software fault proneness must include the measurement of the consequence of failure [13]. Since software failures have different consequences [52] any measure of software security risk must include the measurement of the consequence of failure.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows: in chapter 2, we present the problem statement, research objectives and contributions. We discuss the related work in chapter 3. Chapter 4 addresses the proposed methodology to estimate the probability of security failure, the severity of security failure, and the security risk factors. We also present our case study in Chapter 4 where we apply our proposed algorithms. In Chapter 5, we address the validation and verification criterion. Chapter 6 concludes the work and discusses the future research.

Chapter 2

Problem Statement and Research Objectives

2.1 Problem Statement:

Integrating security in the early stages of software system yet today does not find an appropriate consideration in the practice of software developers. Little time and effort are devoted to this aspect during the software development process and a “fix-it-later” approach is still dominant. This allows software products to obey to the “short time to market” law, but their security, as the ability to meet non-functional requirements, suffers of continuous (and sometime unaffordable) product updates after delivery. The resources, time, and cost spent in the after implementation phase to fix the security issues are increasingly significant.

This lack of validation appears even more serious if we consider that the security software and the software world have been rapidly going, in the last few years, towards component-based system configurations. Security of self-contained software components (either previously developed or acquired from other companies/teams) is changing the software development process.

Among non-functional attributes, a little significance has been given to software security risk. Wherever software plays a role of control in systems whose security

failures may be costly and severe for real life systems and real life productions businesses, the consequences of software security failures are better to be considered from the very early phases of the lifecycle. However, quantification of software security risk is also suitable in other domains (independently of an absolute risk level), in order to detect components and events that may typically put in trouble the software system and the environment where the system will be running.

The risk of a software security product can be defined as a combination of the likelihood of a security failure and severity of “damages” that the security failure may produce. The sources of security failures are usually software security faults, intended as security behaviors that do not meet security requirements.

In this research we identify the following problem

- How to define practical security-based risk assessment methodologies that capture the vulnerabilities and attacks on the software elements based on quantitative measurements rather than qualitative ones that are used in the current risk assessment.
 - Estimate the probability of security failures for components.
 - Estimate the severity of security failures components.
 - Calculate the security risk factors of software elements.

2.2 Research Objectives

In order to improve and control the quality of the software during the software development process, software developers need methodologies to support software

design. Early security risk assessment based on UML models is an important and helpful tool for managers as well as software engineers and could help them effectively improving system security, and reducing the cost significantly on the short and long term, and reducing security-based risk assessment should be well integrated into the software development. Anticipating what might go wrong and managing potential security risks should be considered into the early software development process. Security-based risk assessment is capable of pinpointing the risky components in a system and helping in mitigating these risks. We are concerned with security-based risk taking into consideration of use-case relationships, attack-ability by exploiting the system's vulnerabilities on the components and connectors.

As the need for software systems expands, development methodologies and techniques to the production of software and facilitate its security is needed. Researches, businesses, and governments look for techniques that reduce security cost and improve software system quality. Integrating security into other software attributes at the early stages of software life cycle is significant not only for quality of the software, but also for data and information of the customers, end users, consumers, and enterprise companies who handle sensitive customer information.

Our objective is to develop security risk assessment methodology based on attributes and parameters that could be collected and analyzed in the early software design phase using UML artifacts.

The objectives of this research are:

- ❖ Develop architectural level security risk assessment methodology based on measurable parameters that could be collected and analyzed in the early software

design phase based on UML artifacts. We should be able to transform security risk assessment into a structured problem with systematic solutions and quantitative approach to estimate the security risk of software elements.

2.3 Research Contribution

This dissertation introduces a new approach for security-based risk assessment methodology at the software architectural level.

- ❖ We develop a methodology to estimate the probability of security failures for the software elements in a given scenario and given use case based on the UML specifications and attack graphs.
- ❖ We develop a methodology to estimate the severity of security failures based on the data classification, access rights, and reachability matrix for each element in the software architecture.
- ❖ We propose data sensitivity analysis as an essential indicator to classify the software components in the severity analysis.
- ❖ We calculate the security risk factors of every element of software architecture in a given scenario.
- ❖ We apply the proposed methodologies on an ecommerce scenario case study.
- ❖ We use design security patterns to validate the proposed methodologies on the architectural level and apply them on Microsoft Security Bulletins.

Chapter 3

Literature Review

This chapter discusses previous related work in software security threat modeling, surface attack metrics, attack graphs and security risk assessment. It draws distinctions between prior work and the proposed research.

3.1 Software Security metric and security threat modeling:

In Oxford's American Dictionary, a metric is "a system or standard of measurement". In mathematics and physics, it is "a binary function of topological space that gives, for any two points of the space, a value equal to the distance between them, or to a value treated as analogous to distance for the purpose of analysis. "

Software metrics offer a means to understand the process and product of the software [62]. It is necessary for software quality control [16]. The most important and significant benefit of software metrics is to provide information to support managerial decision making during the software lifecycle. In his research, Shyam [53] recommended that the use of various well-constructed metrics could be a basis for managerial decision making and could provide insight into the software design process. Our focus is on software security metrics.

Software security metrics is an area of computer security that has been receiving a good deal of attention lately. It is not a new topic, but one which receives focused interest sporadically. Much of what has been written about security metrics is definitional, aimed at providing guidelines for defining a security metric and specifying criteria for which to strive. Software security metrics are seen as an important factor in making sound decisions about various aspects of security, ranging from the design of security architectures and controls to the effectiveness and efficiency of security operations. In Our research, we use software security metric to estimate the severity of security failures.

The importance of software security has been clear since most attacks discovered on real software are triggered by weak designed and developed software [59][20][21][54]. It has been clear that the earlier we incorporate security in a software system, the better this would be in terms of efforts and cost [59] [34].

Bowles [4] introduced a concept of threat effect analysis and applied Failure Mode and Effect Analysis (FEMA) to model computer security threats. They showed that the security attacks on systems and software have grown significantly. For examples the number of security incidents reported has grown from 1988 (less than a thousand) to 2003 (140 K). They classified the threats into three categories: Confidential threats, Integrity threats, and availability threats. Their approach provided a framework for modeling threats to the operation of a computer system. The systematic process dubbed “Threat Effects Analysis” analyzes the system, classifies threats, identifies threats, determines threat effects, and applies changes to minimize risk. In his research, he has not estimated the probabilities and security failures of system components, and he has not worked on the architectural level. In our research we model the attacks on the system

using attack graphs and UML, and we estimate the two components of the risk factor (probability, and severity).

The National Institute of Standards and Technology (NIST) [24] provided an overview of the security metrics area and looked at potential research that could be followed to advance the state of the art. It explains how important security metrics are in making decisions about security, starting from requirements and specifications, to the effectiveness and efficiencies of security implementation and maintenance. Our research agrees with our goals that the security is significant and should be taken as early as the requirement and design phase of software life cycle.

SANS institute also has provided a very useful paper covering all the basics aspects of security metrics [41]. It mentions how other research distinguishes between low-level metrics based on well-ordered low-level quantitative system parameters, and higher-level metrics such as conformity distance, attack graph or attack surface estimations [18].

Additionally, Meland [35] presents SODA (a Security-Oriented Software Development Framework), where the main goal is to create a system of practical techniques and tools for creating secure software with a special focus on the early phases of the development lifecycle. The authors have created a tool called SODASWeb that adapts security techniques, filters information and recommends and explains further use of tools during development. Their tool has not been used in the early phases of software life cycle. Their tool has not estimated the probabilities and severities of security failures. Furthermore, Heyman [19] has presented the method of using security patterns to combine security metrics. The approach described integrates metrics in the development

cycle of secure applications, by associating them to patterns. Their approach did not estimate the security risk factors at any level, whereas our approach specifically estimates the risk factors of software architecture components. Their approach explained the idea of security patterns and security metrics. In our research we used two security patterns for validation purposes.

Wang [60] proposed a new approach to define software security metrics based on vulnerabilities included in the software systems and their impacts on software quality. He used the Common Vulnerabilities and Exposures (CVE), an industry standard for vulnerability and exposure names, and the Common Vulnerability Scoring System (CVSS), a vulnerability scoring system designed to provide an open and standardized method for rating software vulnerabilities. In his technique, he has applied the standard on the code level. This was not sufficient as we see new vulnerabilities in the code due to weak design software. Furthermore, this could be seen on every month where big software companies release new patches to cover the security vulnerabilities and attacks on their software. In Wang approach, he introduced a method to qualitatively count the number of vulnerabilities on the code level. This is not a realistic way of measuring security attributes and very expensive to keep. Additionally, Wang's approach is not practical as his approach applies on the code level and every day the attackers are able to find hundreds of unknown vulnerabilities in the system. In contrast, we estimate the probabilities and severities of security failures for each component at the architectural level quantitatively.

Krautsevich [28] considered several security metrics that can be used for measuring security strength, and this metric is defined by number of attacks, minimal

cost of attack, shortest length of attack, maximal probability of attack, overall probability of success, attack surface metric, and percentage of compliance. However in his approach, he has not estimated the probabilities of security failure and has not calculated the risk factors. Furthermore, his analysis was not based on measuring absolute security measurements; instead it was based on compare two systems and tell which one is more secure.

Savola [47] discussed the security metrics and how they could be used to identify risks, and mitigate those risks. This approach is toward counting all possible attack on a system, yet new attacks could emerge. In our approach we look at the architecture components and connectors and we develop a methodology to draw the attacks exploited through these components and connectors without counting all attacks.

Manadhata [32] proposed to use a software system's attack measurement to measure the attack surface for a software system, and this is used to mitigate the software security risk by measuring and reducing the software attach surfaces. Their methodology is based on the identifications of system's resources (methods, channels, and data). They introduced also the damage and effort ratio to measure the attackability on a system. In their approach, they depend on counting all possible attacks through the system resource. However, in their cases studies, they have only counted the directed attacks on a system, not taking in consideration the indirect attacks. Furthermore, their methodology is applicable only on the code level and after systems are development and in implementation. In our approach, we count the direct and indirect attacks on the software components and connectors. Our methodologies are quantitative solution rather than qualitative one and applicable on all phases of software life cycle.

Vaughn [57] questioned the feasibility of “measures and metrics for trusted information systems”. According to him, metrics are possible in disciplines such as mechanical or civil engineering because they complied with the laws of physics, which could be used to validate the metrics. In contrast, the software engineering discipline is not compliant with the laws of physics and faces huge challenges in establishing correctness. Vaughn, however, suggested that effective security metrics can be defined by accepting some risk in how they are used and by validating them in the real world through empirical investigation and experimentation.

Fenton [15] identifies three classes of entities that are of interest of software engineering measurement, namely products, processes and resources. Our focus is on the security metrics at the software product level.

Jaquith [25] explains how security metrics help organizations understand security risks, spot emerging problems, understand weaknesses in the security infrastructures, and recommend technology and process improvements. In our research, our goal is to identify the security risk on system by estimating the security failures at early stages of software life cycle and by pointing out the high risk components.

Liu [29] introduced software security metrics to guide development process in the field of software products, and that includes internal software attributes that could be related to a variety of security qualities. It is clear to note that the internal attributes suggested in this paper cover only a limited view of software security. In our research, we show that the attacker could exploit vulnerabilities through the messages, connectors, and components without the need to know and identify all possible internal attributes. Security is considered part of the software attributes like reliability, performance, and

maintainability. However attacks on a system could come without knowing the internal attributes of the software, like time and other factors. Exploiting vulnerabilities in any architecture could happen through exploiting the resources of the system like messages, components. In our research, we show how the attack could reach certain component by exploiting certain messages that connect the system to the actors.

Sahinoglu [46] proposed a security measurement based on the decision tree based model's attribute to quantify risk using the threats, vulnerabilities and countermeasures to calculate the risk. In his approach he assumes the probabilities of vulnerabilities and threats are given and known, and he does not estimate the risk components and connectors in the system. In contrast, our research develops methodologies to estimate the risk factor of components like probabilities and severities and security risk factors for all components in a given scenario and a given use case. Our approach is quantitative whereas Sahinoglu's approach is qualitative. Furthermore, our approach is applied on the architectural level, whereas his approach is not.

Madan [31] presented an approach for quantification assessment of security attributes for an intrusion tolerant system using a state transition model. Their method did not estimate the risk factors of the components in that system. It showed only how system could transition to different states in case of security failures. This approach did not measure the risk factors of the system components and connectors. In our approach we specifically estimate the probabilities and severities of security failures for each component based on mathematical analysis.

Sharma [49] presented a hierarchical approach for predicting security attributes of software system based on its software architecture and the components attributes using

DTMC. They introduced the Vulnerability index (VI) of a system. However this paper does not estimate the probability of security failure of the system's components and severity of such a failure. Furthermore, they have assumed that the severity of security failure is known. The Focus in our research is on estimating the probabilities and severity of security failures at early stages of software life cycle based on UML specifications given the fact that UML artifacts have been used as foundations to estimate different types of risks.

Chen [8] presented Threat Modeling using Attack Path Analysis (T-MAP) to measure software system security by calculating the total severity weights of attack paths. They used the UML class diagram to model the steps in an Attack Path. In their approach, they don't measure the potential security failure of software architecture. T-MAP method requires comprehensive, accurate and up-to-date vulnerability information. Their approach needs a big database in order to calculate weights. In our approach, we estimate the risk factors based on the software elements that form the architecture of that system. Knowing the components and connectors and communication between the system and outside world would help estimate the probabilities and severities of software elements. UML specifications provide us with enough information about the software elements through the UML sequence diagrams, state diagrams, use cases, and scenarios. Chen's approach is applicable on code level and can't be used to estimate risk at the architectural level. Additionally, their approach is qualitative solution and is expensive to keep up as we see and hear new unknown types of attacks

Neuhaus [39] has showed that some systems' features are correlated with vulnerabilities on the code level. They have conducted empirical study on Mozilla code

base. Their approach depends on the history to predict the future vulnerabilities. This analysis is only valid on the code level and reactive approach after the system is built. However, our approach is proactive and applicable on the design and architectural level in order to identify any possible attacks on the software elements.

Lund [30] showed the need to a UML language to model security assessment in IT systems. UML used in modeling threats, vulnerabilities, risk estimate, modeling treatments, and evaluating the treatment. Our research agrees of integrating the UML at early phases of software life cycle for risk analysis. Our methodologies use the UML specification as the first reference to estimate the probabilities of security failures. Lund has not estimated the probabilities of security failures. We however in our research estimate the probabilities and severities of security failure for every component at the architectural level. Our methodology shows in analytical way and based on probabilistic analysis the path of attacks on the components and it shows the high risk components.

3.2 Surface Attack Metric:

Security attacks on a system happen through data exchange or communication from attackers with that system more specifically that can happen by sending data and or receiving data from the system. In either way, an attacker can connect to the system using systems entry or exit point, or execute actions, send data or receive data from the system. We define here the systems resources to be methods (actions), data, and means to execute these methods and receive/send data from/to that system. On the architectural level, these resources will be represented as components, connectors, and messages. Components have a list of messages that can be executed, connectors that convey the data and

messages between the components and or external entities (attackers), and data is to be transferred and handled by the components and external entities (attackers). The system's attack surface consists of the combination of the systems actions externally visible to the users and the resources accessed or modified by these actions. The more actions or resources are available to the users, the more exposed the system is to successful attacks, and so the more insecure it is.

Manadhata [32] used I/O Automation representation to represent a system s and its environment, and defined the entry points, exit points, schedule, methods, channels, and untrusted data that would help define the attack surface. Additionally, they defined a damage potential and effort ratio to determine the severity of attack surface.

Manadhata and Wing took a conservative approach by assuming the probability of vulnerabilities coming from methods, channels, or untrusted data is 1 and they use the damage effort ratio to obtain the severity of the attack on the system using the damage effort ratio (methods, channels, and untrusted data). And yet the authors are implementing their approach on a system that is already in the implementation or production phase. In our approach we will introduce a methodology that will help us evaluate the probability of security failure based on attack-ability considering the architectural level UML analysis and the early phases of software life cycle. Their approach offered security measurement based on the number of resources the system is using in communication with the outside environment. Their approach applies on the implementation phase regardless of technologies. However, their approach agrees with the current and other approaches that depend on reactive patching and change the code

criteria. This criteria still does not offer a fundamental solution that would help the industries to save money and resources to the companies. In our research, we developed our methodologies based on the UML specifications. UML specifications are clear to identify the system architecture and show how the components are coupled with each other and how the system is connected to the outside world. Our approach takes the risk from early stages of software life cycle, whereas I/O automaton approach takes the risk at late stages of software life cycle. Furthermore, we develop a methodology to estimate the probability of security failure of every component in the system, whereas their approach considers the probability equals 1. In our approach we present a quantitative approach to estimate the severity of security failure in certain scenario based on data sensitivity, access rights, and reachability matrix. However their approach presents incomplete qualitative approach by counting only the direct attacks on the system. Changing the code level and adding patches to the system would make even their approach more difficult to evaluate the attackability on a system. In contrast, adding security patterns on the design level would make the process clearer and easier to re-evaluate the risks of components since we depend on the attack graphs and UML specifications.

Howard [33] proposed an Attackability metric to measure and compare the security of two systems using three dimensions targets and enablers, channels and protocols, and access rights. Their approach implemented on production systems (code level) using a state machine model. Their measurement of attackability does not identify the risky components in a system; it is more toward comparing two systems and determines which system is more secure than the other. Howard's approach is qualitative

solution whereas our approach is quantitative. Additionally, our methodologies are used at the architectural level, whereas Howard approach is used at the code level. In our approach we identify the components with high security risks and that would help the developers and designer build and add more secure layers to these components. Whereas in Howard's approach; he looked at the system as one entity without pointing out the high risky pieces. In conclusion, our approach is much less expensive and practical to use than Howard's approach.

3.3 Attack Graphs:

An attack graph is a graph to represent our architectural system in a set of nodes (condition nodes, initial nodes, exploit nodes, and goal nodes) and edges. The nodes represent the vulnerabilities, and edges represent the paths. These nodes have certain probabilities. The exploit nodes are the messages carried over connectors. The attack graph is a mathematical model to represent how the attacker can reach certain destination in the system and cause a security failure in terms of attackability on the components or connectors.

Jha [26] presented a security analysis using attack graph for attack detection, defense and forensics. They presented a way to produce an attack graph. Their cases studies applied only on the production and late phase of software life cycles, whereas our approach considers the system at the architectural level. Feng [14] proposed an approach based on attack graph to measure security of critical components in a network. Attack graph is a set of condition nodes and exploit nodes. They used probabilities to calculate security risk. However they have not taken in the account the severity and they assigned

values to the exploits nodes probabilities. In our research these exploits nodes probabilities will be estimated and calculated based on the architecture resources extracted from UML sequence diagram. In our research the attacks are initiated through the contact points between the actors and our system components. Their approach depends on knowing the protocols, every single user, and every single machine that are used in the network. In our research, we focus on the UML specifications of the system where UML shows the components and connectors and the messages carried between components without the need to know all protocols. Furthermore, Feng doesn't consider scenario and use cases analysis. Consequently, their approach is not practical for big systems with thousands of machines. In contrast, our methodologies depend on use cases and scenarios and less expensive to adopt. Furthermore, we estimate the risk factors of system components, whereas Feng's approach did not estimate the risk factors in terms of probabilities and severity.

3.4 Security Risk Assessment:

Aagedal [1] introduced a Consultative Objective Risk Analysis System (CORAS) approach to conduct risk assessment of security for critical systems. They proposed models for each step in the risk process. In their approach, they have mentioned that UML graphical style would help speed up risk analysis. In our research we build our methodologies based on UML specifications and consider these specifications as the foundation to estimate the risk factors. CORAS approach explains the risk assessment from management perspective without estimating the risk factors of system, components, and scenarios. In contrast, we build a mathematical model based on UML, attack graph,

and other factors to estimate the risk factors parts (probabilities and severities) of every component in the system in a given scenario and a given use case.

Dimitrakos [12] introduced a CORAS framework for model based risk assessment and apply their methodology on web-based application. Their approach was applied on a system (late phase of software cycle) without estimating any risk values (likelihood, consequences) of a system.

Lund [30] showed the need to a UML language to model security assessment in IT systems. UML used in modeling threats, vulnerabilities, risk estimate, modeling treatments, and evaluating the treatment. In their approach they explained the risk process from the management point without estimating the two parts of security risk factors, whereas our approach takes into the account estimating the probabilities and severities of every component in the system based on mathematical and probabilistic approach. Our approach is applicable on the system at early stages of software lifecycle.

3.5 Summary:

This Literature review indicates the absence of research for early security risk assessment of software architecture. All previous works did not estimate the risk assessment in terms of probabilities and severity on the architectural level. They rather discussed the risk assessment management process on the late phases of software lifecycles. This literature review indicates that there is a need for new methodology for security risk assessment of software architecture. It motivates us also to explore the attack graph based on the UML artifacts. All previous work showed a lack in determining the risk components on the early phases of lifecycle, they have only focused on patch

management and reactive security supports on maintenance phases that would cover the defects in the software systems. Previous work has focused only on counting all possible vulnerabilities that could be in a system; however this qualitative way was not practical as new vulnerabilities merge every day, and it is an expensive way of evaluating security on a system as it consumes a lot of resources without really estimating the security risk factors of the system or the elements in that system. In our methodologies, we identify high security risk components at early phases of software life cycle, so extra efforts could be added to make sure security patterns are planted with the risky components.

Chapter 4

Proposed Techniques and Methodologies

4.1 Introduction

In this chapter, we illustrate the proposed security risk algorithms. We then describe the probability of security failure methodology and the severity of security failure in a given scenario. We calculate the security risk factors. We apply the proposed algorithm on a case study (ecommerce example).

4.2 Security Based Risk Assessment:

In this section, we introduce our security risk assessment methodology by explaining each step of the algorithm. The proposed methodology is based on uses cases and scenarios. In a given use case and given scenario, we estimate the probability and severity of security failures for each element in the system. We will use the UML sequence diagram as the first reference where it shows the messages exchanged between the components and the time lines of the execution. A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the

sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams typically are associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams. In our research, UML sequence diagrams show how the external actors communicate with our system through messages and how the system components are interacting with each other through messages. Figure 4- 2 shows the proposed algorithm.

- The system is represented as a group of UML use cases, and each use case is realized with one or more independent scenarios.
- Each scenario is represented with a UML sequence diagram where it shows a number of components and messages exchanged between them.
 - Extract the components, connectors, and schedules directly from the UML sequence diagram
 - For each component in the scenario, we do the following:
 - ❖ Extract the messages and paths to develop the attack graph. The attack graph for each component shows the paths an attacker could exploit to reach and cause security breach.
 - ❖ Apply the attack graph probability equations to estimate the probability of security failure.
 - ❖ Estimate the severity of security failure based on the component classification, Access rights and reachability Matrix.
 - ❖ Calculate the security risk factor after estimating the two parts of the risk factor.

- Sort the components based on the security risk factors values in the scenario.

We present an ecommerce application to illustrate the steps of our methodology. We choose a typical scenario that allows customers, attackers, or administrators to communicate with the system. In such an application, security attacks could easily happen with significant damages such as loss of customer data records. Figure 4- 6 shows the Sequence Diagram for a buy a book scenario. In the following subsection 4.2.1, we present the methodology of estimating the probability of security failure. Then in subsection 4.2.2, we present the methodology of estimating the severity of security failure. In subsection 4.2.3, we estimate components risk factors.

4.2.1 Probability of Security Failure

In this section, we describe the process of estimating the probability of security failure for each element based on the UML sequence diagrams and attack graphs. We describe the methodology of developing the attack graph for each component from a given UML sequence diagram. Then we use probabilistic arguments to estimate the probability of security failure for that component. We first define the attack graphs as follows. The Attack Graph [14] can be represented as a Tuple:

$$AG = (C_0, T, C_d, E, C_g) \quad (1)$$

- 1- C_0 is a set of initial nodes. The initial nodes are the initial contact points where the attacker initiates the attack. They are subset of the actors where the attackers impersonate the actors.

- 2- T is a set of exploit nodes. They represent the messages between the actors and the system or messages between the components. The attacker exploits these messages in order to reach to other elements in the system.
- 3- C_d is a set of intermediate nodes. The Intermediate nodes are the components of the system where that attacker uses to reach to the goal component. C_g is the goal node. It represents the goal component.
- 4- E is a set of edges between nodes (conditions and exploits).

Figure 4- 1 shows a simple attack graph. $C_0 = \{c_1, c_2\}$, $C_d = \{c_3\}$ $T = \{t_1, t_2, t_3\}$

$Eprob(t)$ denotes the exploit success probability of t . It is the probability of exploiting a message t . For the direct messages between actors and the system, this should be inputs to the algorithm. For the messages between components inside the system, the algorithm will estimate them based on the reachability matrix.

$Cprob(c)$ denotes the condition obtained probability of component c . It represents the probability of reaching component c . For the initial nodes set, $Cprob(c)$ will be input to our algorithm. For intermediate and goal nodes, the algorithm will generate the values of $Cprob(c)$.

$Oprob(t)$ denotes the successful occurrence probability of exploit t . It represents the probability of successful exploitation of a message t coming from components $c_1, c_2, \dots, c_k$. The following algorithm will generate it.

Figure 4- 1: an example of an attack graph $\{c_1, c_2\}$ is set of initial nodes. $\{c_3\}$ is the set of intermediate nodes, c_g goal node, $\{t_1, t_2, t_3\}$ set of exploit nodes

$$\text{Oprob}(t) = \text{Eprob}(t) \cdot \text{Cprob}(c_1) \dots \text{Cprob}(c_k) \quad (2)$$

K is the number of condition or initial connected directly to exploit node t.

$$\text{Cprob}(c) = \sum_{j=1}^n \text{Oprob}(t_j) \quad (3)$$

N is the number of exploits nodes connecting to component c.

Figure 4- 2 The security risk analysis algorithm

Node c is an intermediate component or goal component. Where $t_1, \dots, t_n$ are the messages reaching component c .

The following steps describe the proposed methodology to develop the attack graph based on UML sequence diagram:

1. Choose a component as a goal node.
2. Extract the initial set C_0 from the UML sequence diagram. $C_0 = \{c_1, c_2, \dots, c_m\}$, where m represents the number of the total direct messages exchanged between the

system and outside world. We assume there is one attack on the system through one of these initial nodes. We assume all initial nodes have equal initial probabilities, and we consider that we have 8 malicious users initiating the attacks from 8 different points of contacts. In this situation we are considering the most general scenario (8 attackers).

$$Cprob(c_1) = \dots = Cprob(c_m) = VI \quad (4)$$

VI [49] denotes the Vulnerability Index (VI). Vulnerability Index is defined as the number of successful attacks on the system to the total number of attacks. The value of VI is determined by domain experts. In equation 4, we assumed that all initial points have the same initial probabilities. We refer to section 5.3.2 in chapter 5 to discuss the validity of this assumption.

3. Extract a partial set T' of exploit nodes set T ($T' \subset T$). $T' = \{t_1, t_2, \dots, t_m\}$. T' represents the set of direct messages exchanged between the external actors and the system. For simplicity, we assume all these external messages have equally likely distribution, and we will be discussing various possible cases in chapter 5. We will apply the same methodologies on the new cases where we show maximum risk factors under the cases in the validation studies chapter. The following assumption in equation 5 is considered only for simplicity reason even it assumes lower estimate of risk factors as it will show in chapter 5.

$$Eprob(t_1) = \dots = Eprob(t_m) = \frac{1}{m} \quad (5)$$

4. Extract a partial set of the intermediate condition nodes. These represent the components in our system connected to the external actors through set T' .
5. Extract a partial set of exploit nodes set that represents the internal messages produced by components in step 4.
6. Repeat step 5 and 6 above until we include all internal messages and internal components that lead to the goal node.
7. From UML sequence diagram, we extract the schedules [32]. A schedule is defined as a sequence of messages executed to do a certain process. These schedules will help count the effect of one message on a component only once.

Form the reachability Matrix M . Elements in Matrix M represent the numbers of exchanged messages between every two components in a given scenario. p_{1n} is the number of messages leaving component 1 and entering component n .

$$M = \begin{bmatrix} 0 & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & 0 \end{bmatrix}$$

Figure 4- 3 shows the tree diagram of a component j in a given scenario that describes the above steps.

Figure 4- 3 Tree Diagram for a component j

4.2.2 Severity of Security Failure

In this section, we consider the severity of consequences of security failure. Our approach takes into the account the severity related to each component in the software architecture. Our approach depends on how the security failure for each element is impacting the system. Severity of security failure for certain element should consider the worst case consequence of such failures. Pfleeger [18] discusses the three common categories of impact that could affect elements' confidentiality, Integrity and availability. From the three categories, we come up with two classifications of architectural components. One is a database component category, and the second is non-database component category. The reason behind this classification is the intent of the attacker to get as much information as

possible. One of the hacker's main goals is to steal as much data as he or she can. For example, [9] shows that hackers could have access to millions of customer sensitive information and records such as customer names, account numbers and contact information (email addresses) were all exposed. Another example is the Sony Playstation data security breach [10] where 77 million PlayStation accounts were stolen. The later security breach on data included user names, passwords, users email addresses, and dates of birth. A third real life example is shown in [58] where hackers were able to view password and other sensitive information on the cell phones. Another incident shows the data is a goal for the attacker to steal the data is [11], the attacker stole personal information from over 1.2 million customers of a Japanese gaming company. The stolen information includes names, email addresses, and encoded passwords. Another example shows that there are 540,613,790 record breaches since 2005 [64]. Therefore, it is very important to focus on the database components where sensitive data might be stored and saved. The database components could be further classified according to their sensitivity. Some database components may store high sensitive records that could lead to a full access to the system. Some database components may store information that leads to no other components in the architecture. In our algorithm we take this in our consideration where the higher sensitive information is stored in the database, the higher severity we expect. In our analysis, we test the access rights when accessing the database component; we find the higher the access rights, the more severe when the security failure occurs. The non-database components can be classified into two types, one has a direct connection with a database component and offer services to it and consequently has the same level of severity of that database component. The second type does not connect directly to a database component and we check the reachability. We identify five severity levels.

- Catastrophic: A security failure could cause security breach to the whole system and whole database (all records).
- Critical: A security failure could cause security breach to the whole system (one record) or two database components (all records).
- Major: A security failure could cause security breach to one database component (all records) or two database components (one record).
- Minor: A security failure may cause security breach to one database component (one record) or security breach to non-database component with high reachability.
- Low: A security failure may cause security breach to non-database component with low reachability.

Values of 1, 2, 3, 4, and 5 are assigned to low, minor, major, critical, and catastrophic levels respectively.

Figure 4- 4 describes the steps to estimate the impact on the system when security failures occur.

Figure 4- 4 The security severity analysis algorithm

4.2.3 Security Risk Factors

In this section, we calculate the risk factor for each component in a given scenario based on the probability of security failure and severity of security failure using the following equation:

$$rf(c)=Prob(c)*Severity(c) \quad (7)$$

Where $\text{Prob}(c) \{0 \leq \text{Prob}(c) \leq 1\}$ is the probability of security failure of a component in a scenario, and severity (c) $\{0 < \text{severity}(c) \leq 1\}$ is the severity level of a component in the same scenario. After calculating the risk factors for each component in a given scenario, we form the scenario list risk factors and sort them.

4.3 Case Study

We have selected a case study of ecommerce application to illustrate how the proposed methodology works. Figure 4- 5 shows Ecommerce system architecture [44]. The ecommerce application allows customers and other actors to interact with each other over the internet. In this type of application, security attacks could easily happen with significant damages such as loss of customer data records. The severity of security failure depends on the type of data sensitivity and usually is different for different types of security failures.

The e-commerce system allows a customer to browse products provided by the system, select the item to be purchased and place the order. The order is then processed by checking that the customer has enough funds with the financial institute. The customer during this process communicates with the system through connectors. Figure 4- 6 shows the UML Sequence Diagram for a buy a book scenario. We show how the proposed methodology applied using this case study as illustrative example.

Figure 4- 5 E-commerce system architecture

Stage 1: Identify Components, connectors, and schedule in that scenario

Components:

- 1- Customer <<External actor>>
- 2- Customer Interface
- 3- Customer Agent
- 4- Delivery Agent
- 5- Customer information <<database>> includes customer usernames, passwords, customer IDs, orders numbers,
- 6- Books server <<database>> includes the databases of all books
- 7- Orderserver <<database>> include Order ID, customer IDs, Addresses, order details
- 8- Financial Institute <<external actor>>

Connectors:

- 1- Connector1: customer:: customer interface
- 2- Connector2: customer interface:: customer agent
- 3- Connector3: customer agent:: customer information
- 4- Connector4: customer agent:: books server
- 5- Connector5: customer agent:: delivery agent
- 6- Connector6: customer agent:: financial institute
- 7- Connector7: customer agent:: order server

Schedules:

1. Schedule1:Accessmainpage(){1.1}→mainpage(){1.2}.
2. Schedule 2: login(username, password) {2.1}→ send(username, password)
{2.2}→ userinformationquery(username,password) {2.3}→ verified(){2.4}→
loginconfirm(){2.5}→ loginconfirm(){2.6}
3. Schedule 3: buybook(book,creditcard) {3.1}→ buybook(book,creditcard) {3.2}→
search(book) {3.3}→ bookavailable(){3.4}→
reservefunds(creditcard,customerinformation) {3.5}→ fundreserved(){3.6}→
generateordernumber(){3.7}→orderstatus(){3.8} → orderstatus(){3.9}
4. Schedule 4: buyrequest(ordernumber, address, book) {4.1}
5. Schedule 5: writeorderdata(ordernumber, address, book) {5.1}

Stage 2: Identify the messages, data, connectors related to customer agent component

Customer agent

1- Messages:

- Send (username, password) [read username and password from connector2]
- userinformationquery(username,password) [read data from the database customer information, and do inquiry in the customer information database]
- verified() [true if found, false otherwise-- read data from connector 3]
- search(book) [do inquiry in the book server database]
- bookavailable() [true if found, false othewise-- read data from connector4]
- generateordernumber() [internal action to generate a unique order number for this specific scenario]
- reservefund (creditcard,customerinformation) [write data to connector 6 and send it to the financial institute, data is sent to the outside system]
- fundreserved() [true if reserved, false otherwise--- read data from connector6]
- buyrequest(ordernumber, address, book) [write data to connector5, send it to the internal component delivery agent]
- writeorderdata(ordernumber, address, books) [write ordernumber, address, books information over connector 7 and send it to the orderserver database]
- orderstatus() {send order status info to the customer interface over connector 2}

2- Data {username, password, book, creditcard, ordernumber, customer information,}

- 3- Connectors {connector2, connector3, connector4, connector5, connector6, connector7}

Stage 3: Build attack graph of customer agent component

The following steps describe the proposed methodology to develop the attack graph based on UML sequence diagram:

1. Choose a component as a goal node. We will pick the customer agent in Figure 4- 6 as a goal node.
2. Extract the initial set C_0 from the UML sequence diagram. In Figure 4- 6 $C_0 = \{c_1, c_2, \dots, c_8\}$, where $m=8$ represents the number of the total direct messages exchanged between the system and outside world. We assume that there is an attack on the system through one of these initial nodes. We assume all initial nodes have equally likely distribution,

$$Cprob(c_1) = \dots = Cprob(c_8) = VI \quad (4)$$

VI [49] denotes the Vulnerability Index. (VI) is defined as the number of successful attacks on the system to the total number of attacks. The value of VI is determined by domain experts.

3. Extract a partial set T' of exploit nodes set T ($T' \subset T$). $T' = \{t_1, t_2, \dots, t_8\}$. T' represents the direct messages exchanged between the external actors and the system. If we assume all these external messages have equally likely distribution,

$$Eprob(t_1) = \dots = Eprob(t_8) = 1/8 \quad (5)$$

Messages {Accessmainpage(), Mainpage(), login(username,password), loginconfirm(), buy(book,creditcard), orderstatus(), reservefund(), fundreserved() } are the exploit nodes in the set T' .

4. Extract a partial set of the intermediate condition nodes. These represent the components in our system connected to the external actors through set T' . Figure 4- 6, the customer interface component is an intermediate node, and c_{ci} denotes the customer interface component node.
5. Extract a partial set of exploit nodes set that represents the internal messages produced by components in step 4. In Figure 4- 6, The internal messages send(username, password) and buy(book, creditcard) are the messages produced by the component customer interface node. In Figure 4- 7, these two messages are (t_{ia1}, t_{ia3}) to the customer agent goal node.
6. Repeat the previous step 5 and 6 above until we include all internal messages and internal components that lead to the goal node.
7. From UML sequence diagram, we extract the schedules [32]. A schedule is defined as a sequence of messages executed to do a certain process. These schedules will help count the effect of one message on a component only once. In the ecommerce application, we define five schedules.

- a. Schedule1:Accessmainpage(){1.1} $\rightarrow$ mainpage(){1.2}.
- b. Schedule 2: login(username, password) {2.1} $\rightarrow$ send(username, password) {2.2} $\rightarrow$ userinformationquery(username,password) {2.3} $\rightarrow$ verified(){2.4} $\rightarrow$ loginconfirm(){2.5} $\rightarrow$ loginconfirm(){2.6}

- c. Schedule 3: buybook(book,creditcard) {3.1} → buybook(book,creditcard)
{3.2} → search(book) {3.3} → bookavailable(){3.4} →
reservefunds(creditcard,customerinformation) {3.5} → fundreserved(){3.6} →
generateordernumber(){3.7} → orderstatus(){3.8} → orderstatus(){3.9}
- d. Schedule 4: buyrequest(ordernumber, address, book) {4.1}
- e. Schedule 5: writeorderdata(ordernumber, address, book) {5.1}

Figure 4- 6 UML Sequence diagram of a buy book scenario

Stage 4: Estimate the probability of security failure of customer agent component:

After developing the attack graph for each component, we use the equations (2&3) to estimate the probability of security failure of the goal node. Figure 4- 7 shows the attack graph of customer agent component.

$$Cprob(c_{ci}) = Oprob(t_1) + \dots + Oprob(t_6) = VI * (\frac{6}{8})$$

The exploits nodes (t_{ia1}, t_{ia3}) represent the messages send(username, password) and buy(book, creditcard) going between the components customer interface and customer agent. These two messages will carry over any attack coming from outside. $Eprob(t_{ia1})$ and $Eprob(t_{ia3})$ are estimated:

$$Eprob(t_{ia1}) = 1/Z \quad (6)$$

Where Z is the total number of messages exchanged between the two components. Matrix MA shows the numbers of exchanged messages between every two components i, j in our system. Figure 4- 6 shows we have 6 components, the following MA (6*6) shows messages exchanged in this example.

$$MA = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Eprob(t_{ia1}) = Eprob(t_{ia3}) = 1/4$$

We can estimate $Oprob(t_{ia1})$ & $Oprob(t_{ia3})$ as following

$$Oprob(t_{ia1}) = Eprob(t_{ia1}) * Cprob(c_{ci}) = \frac{1}{4} * (VI \frac{6}{8})$$

$$Oprob(t_{ia3}) = Eprob(t_{ia3}) * Cprob(c_{ci}) = \frac{1}{4} * (VI \frac{6}{8})$$

$$Cprob(g) = Oprob(t_7) + Oprob(t_8) + Oprob(t_{ia1}) + Oprob(t_{ia3})$$

$$Cprob(g) = \frac{5}{8}VI$$

$Cprob(g)$ is the estimated probability of security failure for customer agent component. In Figure 4- 7, we notice the higher number of internal messages between components, the higher the probability of security failure. The attack starts initially on any component from the touch points and carried over through the direct messages. Then the internal messages carry such attacks further to the internal components that have no direct connection with the external system. We can conclude the higher coupling between components the higher probability of security failure could be. This is shown in the attack graph through the edges that connect the exploit nodes and intermediate nodes. The higher number of paths in the attack graph, the higher the probability of security failure. We assume there is an attack on a system, and this attack starts off through one of the direct messages that connect the system with the external actors. Similarly, we can estimate the probabilities of security failures for the other components. It is important to mention that UML sequence diagram will help us sort the messages exchanged between components based on the execution time. The execution sequences of messages will help us detect where the attack is coming from and when could be happening. Table 4- 1 shows the probabilities of security failures for each component in this specific scenario.

Figure 4- 7 Attack Graph of customer agent component in a given scenario

Stage 5: Estimate the severity of security failure of customer agent component:

In this stage, we estimate the severity of security failure of customer agent component. We need first to classify this component based on the algorithm in Figure 4- 4. This component is not a database component; however it has connections to more than a database component. In the severity analysis, we consider always the worst consequence when a security failure occurs. Since the customer agent component has a direct connection with customer information database record component, we only need to estimate the severity of the customer information database component. The customer information database keeps the customer information such user ID, passwords, and other sensitive information. Based on the sensitivity analysis, any security breach (unauthorized read or modify) to this data could simply lead access to the whole system on one record or all records depending on the access rights. We can conclude the sensitivity is high and as a result the severity is critical. However if we consider the access right is admin, the severity even would be worse and could climb to catastrophic level. This means the severity of customer information database is 5. After normalizing the severity the value of customer information database component severity is 1. Consequently, the customer agent severity is also 1. The following pseudo code is used to estimate the severity of security failure given certain information to the code (component classification, data sensitivity). From the UML sequence diagram, we extract the reachability matrix, and access rights. There should be user interaction with the system where the user will enter the data sensitivity of each component. The output of this algorithm is the severity of each component. The UML sequence diagram provides reachability matrix, name of each component, specification of each component.

```
Severity_Analysis_Code;  
Variables;
```

```

String Component_sensitivity[], Component_Severity[], Component_name[],
Reachability[];
Boolean isdatabasecomponent[], direct_connectionwithdatabasecomponent;
Integer i, j, numberofcomponent;
Real normalizedseverity[];

Main // begin of Main program
{
Input(numberofcomponent); //enter number of components

For (i=1,numberofcomponets,1)
    Input(Component_name[i]); //enter the names of each component

For (i=1,numberofcomponets,1) // check if a component is a database component and
//assign a sensitivity value to the component
{
    If (Component_name[i] is database)
    {
        Isdatabsecomponent[i]=1;
        Component_sesnitivity[i]=Checksensitivity(Component_name[i]);
        // “low”, or “medium”, or “high”
        Component_Severity[i]=AssignSeverity(Component_sensitivity[i]);
    }

    Else
    {
        Isdatabsecomponent[i]=0;
        Component_sesnitivity[i]=”zero”;
    }

} // end of For loop

For (i=1,numberofcomponets,1) // check if a nondatabase component has a
//directionconnection with a database component
{
    If (Isdatabsecomponent[i]==0)
    {
direct_connectionwithdatabasecomponent[i]=checkconnection(Component_name[i])

        if (direct_connectionwithdatabasecomponent[i]==1)
        {

```

```

        Component_Severity[i]=Component_Severity[j]; //j is the
//database component index, i is the nondatabase component index that has connection with the
//database component
    }
    else
    {
        Reacability[i]=Comput_Reachability(Component_name[i]);
        If (Reachability [i]>1)
        Component_Severity[i]="minor";
        Else
        Component_Severity[i]="low";

    }// end of else
} // end of if

} // end of For loop

For (i=1,numberofcomponets,1)// check the access right of a component
{
    If (Component_name[i].Accessright==admin)
        Component_Severity[i]=IncreaseComponentseverity(Component_Severity[i]);
}

For (i=1,numberofcomponets,1)// Normalize the severity
{
    Normalizedseverity[i]=Normalize(Component_Severity[i]);
}

} // end of Main program

String AssignSeverity(String st1);
{
    If (st1=="low")
        Return "minor";
    Elseif (st1=="medium")
        Return "major";
    Else
        Return "critical";
}

String IncreaseComponentseverity( String st2);
{
    If (st1=="low")

```

```

        Return "minor";
    Elseif (st1=="minor")
        Return "major";
    Elseif (st1=="major")
        Return "critical";
    Elseif (st1=="critical")
        Return "catastrophic";

}

real Normalize (String str)
{
    If (str=="low")
        Return .2;
    Elseif (str=="minor")
        Return .4;
    Elseif (str=="major")
        Return .6;
    Elseif (str=="critical")
        Return .8;
    Else
        Return 1;
}

```

Stage 6: Calculate the security risk factor of customer agent component:

In this stage, we calculate security risk factor of the component in a given scenario. This can be done by using equation 7.

$$Rf(\text{customer agent component}) = \text{Probability} * \text{Severity} = (5/8)VI * (1) = 5/8VI$$

We repeat the stages 2, 3, 4, 5, and 6 so we can calculate the security risk factors of every component in the system in this given scenario. We study another component (database customer information component) and estimate the probability of security failure, severity of security failure and the security factor. Figure 4- 8 shows the attack graph for a customer database information component. We use the equations (2 & 3) to estimate the probability of security failure. From the attack graph of customer information database, we can clearly see that the path to this component is coming from the customer agent component through the message login.

$$C_{prob}(g) = \frac{5}{16} VI$$

The severity of customer database component when a security failure occurs is 1. Customer information component is a database component that keeps the username, password for every customer and for administrator. The data sensitivity is high as it could lead to access to the whole system. Thus the severity should be considered Critical. Taking the access right in the consideration, this severity would become catastrophic.

Rf(customer information database component)=Probability * Severity=(5/16)VI*(1)=5/16VI

Figure 4- 8 Attack Graph of customer information database component

We form a list of risk factors for all components. Table 4- 1 shows the probabilities of security failures, the severity of security failures, and the risk factors of all components in the buy a book scenario.

Component	Probability	Severity	Risk Factor
Customer Interface	VI(6.5/8)	.6	.48VI
Customer Agent	VI(5/8)	1	.62VI
Customer information database	VI(5/16)	1	.31VI
Books database	VI(5/16)	.6	.18VI
Delivery agent	VI(5/8)	.6	.37VI
Orders database	VI(5/8)	.6	.37VI

Table 4- 1 Probabilities, severities and risk factors of all components in buy a book scenario

4.4 Conclusions and future work:

In Summary, this research describes methodologies for estimating probability of each software architectural element at the software architectural level as well as severity of security failures. The methodologies are based on UML specifications, Attack Graphs, Data Sensitivity, Reachability Matrix, and Access rights. In estimating the probability of security failures, the methodology uses the UML sequence diagram to develop the attack graph for each element in a given scenario. Then we use probabilistic arguments to estimate the probability of security failure for each component. The second methodology uses important factors such as data sensitivity, reachability matrix, and access rights of each element to estimate the severity of security failure.

In the future work we would develop a tool support that will help domain expert apply these methodologies.

Chapter 5

Validation Studies

5.1 Introduction:

Evaluation techniques can be one of these categories [42]:

- 1- Feature analysis
- 2- Surveys
- 3- Case study
- 4- Formal experiment

Validating security metrics requires huge efforts. It is difficult to validate software attributes [27]. There is no agreement on a validation framework [6][27][36][61]. One of the software features is security, and this feature is difficult to measure, therefore validating a security measure is even more difficult.

Validation can be done generally theoretically or empirically [6] in software engineering. Theoretical validation will show the measure is really measuring the attribute it is purporting to measure. Empirical validation will show the measure is useful so that it is related to other variables as defined in the theories.

Measurements are involved in evaluation, and they help separate typical from unusual situations [42]. Measure is a mapping from a set of entities and attributes in the real empirical world to a representation or model in the mathematical world. For example, Line of Code (LOC) is a measure of software “length”, and then this value is handled in the mathematical

world to get more information about the real world. Most of the software measures are related to information about product, process, or resource's attributes.

To understand software measurement, two types systems are considered:

- ❖ Measurement systems: used to assess existing entity by numerically characterizing one or more of its attributes.
- ❖ Predication systems: used to predict some attribute of a future entity, involving a mathematical model with associated prediction procedures.

Validating Software Measures: This type of validation must ensure the measure captures the attribute property it is supposed to capture.

Validating Prediction Systems: This type of validation is done on prediction system in a given environment by establishing its accuracy by empirical means. A comparison between the model's performances with known data in the given environment should be done. Two different types of prediction systems in this category must be considered, deterministic prediction systems, and stochastic prediction systems.

Note: some measures can serve both purposes, as an attribute measure and as input to a prediction system. In our research, the security risk assessment plays dual role, attack graph and Data Sensitivity, Access Rights, and Matrix Reachability are measures of software attribute and that is security and also a prediction systems to indicate the security risk of that software.

5.2 Validating Software Measures and Prediction Systems

5.2.1 Validating a Software Measure

The data collected from real life world can be used to validate a software measure. One example is Microsoft Security Bulletins [37]. Statistical analysis can be conducted using the data collected from Microsoft Security bulletins. The Bulletins show how the vulnerabilities can be exploited on the code level using the software resource from components, connectors and data. This can be done also on the architecture level by mapping the components and connectors that are exploited. One of validation methods is compatible with validations used in Kitchenham et al. [27]. In order to decide whether a measure is valid, the following should be confirmed:

- 1- Attribute validity: i.e. whether the attributes we are interested in are exhibited by the software.
- 2- Unit validity: the measurement unit is appropriate for an attribute.
- 3- Instrument validity: the model underlying a measurement is valid.
- 4- Protocol validity: an appropriate measurement protocol is adopted.

In our research, Data Sensitivity, Access Rights, and Matrix Reachability are used to determine the severity of risk assessment. Messages, connectors, and data are ways to establish attribute validity along with access rights for estimating the severity of security risk. Similarly attack graphs use components and connectors along with messages exchanged between components to establish attribute validity. UML model is used in the research and can establish theoretical validity, and empirical studies based on Microsoft Security Bulletins

can establish the attribute validity empirically. In the upcoming section, validation studies and surveys can be used for validation.

In our research, there are no units used in the security risk assessment. However, within the security risk assessment there are two elements; probability and severity. In regards to the probability, there is no explicit unit. However the severity unit could be Catastrophic, Critical, Major, Minor, or Low. Using Microsoft Bulletin and empirical study, we use severity level to establish the unit validity. These levels for example could be high, moderate or low level (high>moderate>low). Surveys are retrospective studies to try to document relations and outcomes in a given situation. For example, software engineers' surveys are similar to those when recording data to determine how project participants reacted to a particular method, tool, or technique or to determine trends or relationships. Surveys are commonly used to gather data from participants, engineers, developers, customers, and end users. Measures from these groups about beliefs, trends, and attitude are taken.

In our research, we use UML model along with attack graphs and Data Sensitivity, Access Rights, and Matrix Reachability to determine the two parts of security risk assessment. UML model is valid and widely used across researchers, communities, projects, and companies. Therefore our security risk measurement consequently establishes instrument validity.

Regarding the protocol validity Kitchenham et al.[27] suggests that a valid measurement protocol must be clear and must prevent problems and invalid measurement procedures such as double counting, so any protocol that satisfies this criteria is validated

through researchers' community agreement. Our research meets Kitchenham's criteria; hence our measurement is a valid measurement protocol.

In conclusion, our software measure (security risk assessment) satisfies the four criteria of Kitchenham [27] as explained above, so we can conclude that our measure is valid.

5.2.2 Validating a Prediction System

In our research, we can show that a higher security risk (probability, severity) on any component in the software architecture would lead to a higher number of attacks on that component. Let us choose two components from the UML sequence diagram in a certain scenario and a certain use case. We showed in research that component1 has higher probability of security failure than component2 {for example customer interface has probability of security failure $P1 = \left(\frac{6.5}{8}\right) VI$, customer information database component has probability $P2 = \left(\frac{5}{16}\right) VI$ } that would lead us that component 1 has higher attackability chances than component 2. Consequently the security risk factor would be $rf1 > rf2$ assuming that they have the same severity level. In this way we are showing that using the theoretical approach (UML, Attack graphs, Data Sensitivity, Access Rights, and Matrix Reachability,) we are proving that a larger security risk factors on any component would lead to a larger number of security attacks on our software.

Correlations between security risk measurements and patches of security vulnerabilities in software can be built. Patching a software system improves system's security by reducing the probabilities of security risk and/or reducing severity of security failure. Consequently the patch will decrease the security risk factors. Microsoft Security

Bulletins have explanations on the severity resulting from the security failure and types of components and data used to exploit vulnerabilities. Patching generally is implemented on the code level, however on the architecture level we can prove that adding security protection to the components exposed to high security risk would decrease the security risk factors of these components. In our research, the most exposed components to the attackability are customer interface and customer agent components. Patching or adding more secure layers and technologies to these two components would lead us to reduce the security risk factors of these components and consequently reduce the security risk factors of the scenario and the use case.

In the following Table 5- 1, a summary is provided on the validation of a Software Measure and the validation of a Prediction System.

Measuring systems	Validity	Theoretical	Empirical
Software Measure	Attribute Validity	UML Model,	Microsoft Security
	Unit Validity	Attack Graph, Data	Bulletin (MSB)
	Instrument Validity	Sensitivity, Access	analysis, Survey
	Protocol Validity	Rights, and Matrix Reachability	
Prediction System	Prediction Validity	UML, Attack Graphs, Data Sensitivity, Access Rights, and Matrix Reachability,	Microsoft Security Bulletin, Survey, Security Pattern

Table 5- 1 Summary of validations methods

5.3 Validation based on Sensitivity Analysis

In this section, we make some changes to the parameters in the case studies in regards to the vulnerability index (VI) and check the probabilities of security failures of the goal components in that given scenario. Additionally, we make changes to our original assumption on the initial probabilities where we assign different probabilities values to the initial nodes in the attack graph and test how that impacts the probabilities of security failures to certain goal components.

5.3.1 Vulnerability Index analysis:

In this section we will relax the values of vulnerability index and see how that would change the values of probabilities of security failures of customer agent and customer database information components. Figure 5- 1 shows how the probabilities of two components change when VI changes. When $VI = 1$, the estimated probabilities of customer agent and customer information database components are .625 and .31 respectively assuming the initial nodes have equally likely distributions. However in the coming sections we will show the estimated probabilities of security failure when this assumption changes. In Figure 5- 1, we have changed the values of VI by .05.

Observations: We conclude the following:

- 1- Components with higher probabilities of security failures are more sensitive to VI values than components with low probabilities of security failures. For example, the sensitivity to VI for the customer agent component is significant. However the sensitivity to VI for customer information database is insignificant because even if VI is equal to 1, the probability of security failure of customer information database is around .31. The estimated probability value .31 is still considered low comparing to the other estimated probabilities of other components like customer interface or customer agent.

Figure 5- 1 Probabilities of security failures of customer agent and customer information database components when VI changes

5.3.2 Initial Probabilities Analysis:

The methodology of risk assessment presented in chapter 4 assumed that the initial probabilities of security attacks are equal to the vulnerability index as specified in equation 4. In this section, we analyze the validity of this assumption using Microsoft Security Bulletin data collected from real life attacks on real systems. It is important to consider other factors when assigning the initial probabilities values. The following point should be taken when considering the initial probabilities values:

- The outcome of any security attack falls in one of these categories, integrity violation or confidentiality violation or availability violation or combinations of these violations. Integrity means that there is change in data that includes modify or delete or add to the data. Confidentiality means the attacker is

reading a data without having the proper authorization. Availability in security means that the component is not available because certain attacks have flooded the component or the system with a lot of requests (denial of service attacks are typical attacks that cause availability problems). Integrity violation happens in terms of entering data to the system. Confidentiality violation happens in terms of reading data from the systems. Availability violation happens in terms of flooding the system with requests by sending requests to the system and waiting for a response.

In this section, we show how initial probabilities values change according to the above factors by using Microsoft Security Bulletins (MSB) [37] statistics. MSB could help us determine what the favorite components that attacker prefers, and what security violations happen more than others in terms of integrity, confidentiality, and availability. They could have different values depending on the favorite systems components and on the violations in confidentiality, integrity and availability. We will use the Microsoft Security Bulletins (MSB) in 2004 as a reference to provide us with statistics regarding the integrity, confidentiality and availability. We will use MSB database in 2004. Table 5- 2 shows the number of total attacks, number of integrity violations, number of confidentiality violations, and number of availability violations.

	Number of violations	Percentage %
Total number of security attacks	45	100
Number of security attacks in terms of integrity violations	36	80
Number of security attacks in terms of confidentiality violations	38	84
Number of security attacks in terms of availability violations	9	20

Table 5- 2 Numbers of violation from MSB -2004

In Table 5- 3, we will assume initial probabilities values based on the previous analysis of favorable components and types of violations.

Types of attacks	Confidentiality	Integrity	Availability
Initial probability	.84	.8	.2

Table 5- 3 Initial probabilities values for confidentiality, integrity availability security violations

In Table 5- 4 we will connect the initial points of contacts in our case study to the type of violations resulted from security attacks.

Initial point of attacks	Potential security violations	Initial probability
C1	Availability	.2VI
C2	Availability	.2VI
C3	Integrity	.8VI
C4	Confidentiality	.82VI
C5	Integrity	.8VI
C6	Confidentiality	.82VI
C7	Confidentiality	.82VI
C8	Integrity	.8VI

Table 5- 4 Initial Probabilities values of the case study e-commerce example

5.3.3 Severity analysis:

In this section, we test the severity of e-commerce components when we change the access rights. Figure 5- 2 shows the severities of security failures for each component in the ecommerce example in two states, with admin rights and without admin rights. The components severities with admin rights will increase the severity level by one level and that is in our case by .2. Figure 5- 3 compares the risk factors of ecommerce components taking into the account the admin rights, and vulnerability index value is .1.

Observation: we conclude the following

- 1- There are components with high level of severity with admin access rights or without admin rights. These are customer interface and customer agent. These

components are not sensitive to admin access rights because their severity levels stay at the high levels in both cases with admin or without admin access rights.

- 2- There are components that could be impacted when admin access rights are considered. The severity level of these components will increase from minor to major, which is considered a significant jump. For example, the severity level of the order server database component will increase from minor to major. That means the attacker could have access to the whole database or to one record. These types of components are sensitive to the admin access right. This is an intuitive conclusion.
- 3- When considering the severity analysis and the risk factor we conclude the following:
 - a. Customer agent component is considered high risky components with admin access right or without it.
 - b. Bookserver database component keeps low risk factor although his severity sensitivity to admin access right is significant. This is because the probability of security failure of this component is so low. That means although this component is sensitive to the access rights, the attackability on this component is difficult.

Figure 5- 2 severities of security failures of e-commerce components
with/without admin rights

Figure 5- 3 risk factors of software components in ecommerce example with
admin rights, without admin rights, VI=.1

5.3.4 Risk factors analysis:

In the previous chapter we considered in the case study an equal likely distribution for direct messages, however this might not be the case. In this section we test the security risk factors for three components (customer agent and customer interface, bookserver database).

Figure 5- 4 Security risk factors for three components, customer agent, customer interface, and bookserver database with 9 cases, VI=.1

We take into the account the values of $x_1, x_2, \dots, x_8$ are not the same. We consider 9 cases. In each case we estimate the probabilities of these three components with the assumption that $V_i=1$. Figure 5- 4 shows how the security risk factors of the three components change when the initial nodes probabilities change in 9 different cases.

Case 1: we assume equally likely distribution across all initial nodes.

Case 2 – case 9: we assume one initial node has a probability of 1 and the rest nodes have zero probabilities.

Observations: we conclude the following:

- 1- The risk factor of customer agent component is high in case 8 and case 9 when the probability of initial nodes x_8 is one or x_9 is one. This represents a direct attack coming from the actor on the customer agent component through the exploitation of the direct messages (2 direct messages); intuitively this is an expected result.
- 2- The risk factor of customer interface component is high in case 2 through case 7 when the probability of initial nodes x_1 is one or $x_2=1, \dots, x_6=1$. This represents a direct attack coming from the actor on the customer interface component through the exploitation of the direct messages (6 direct messages); intuitively this is an expected result.
- 3- The risk factor of book server component is high in case 8 and case 9 when the probability of initial nodes x_8 is one or x_9 is one. This represents an indirect attack through customer agent component; therefore this is an expected result. The bookserver database component does not have a direct connection with the actors; however the attack could happen through the message coming from the customer agent component.
- 4- In case 1, the risk factors of customer agent, customer interface, and book server database show the averages of the two extreme cases explained in the above conclusions 1, 2, and

3. This case does not show a real behavior in which there is an attack coming from a source than another. For simplicity, we considered our example that all initial nodes have equally likely distribution.
- 5- Figure 5- 5 Shows that the distributions of risk factors for each of the three components, customer agent, customer interface, and bookserver database considering 9 cases.
- 6- The customer agent and book server database components have a similar trend, but the only difference is the value of risk factors. This conclusion tells us that the direct messages have more impact than the indirect messages on components from the security failures standpoint.
- 7- The pattern of customer interface risk factor is different from the ones of customer agent and book server, and this is intuitively because 6 out of 8 direct messages are connected through customer interface and 2 out of 8 direct messages are connected to the customer agent.
- 8- The customer interface component stays at its high risk value in more cases than the customer agent. This conclusion results from the number of direct messages on customer interface is more than the number of direct messages on the customer agent component.
- 9- In next section, we introduce the design security patterns that could be used on the architectural level and will add more protection layers to the components. These patterns will help lower the security risk factors of components.
- 10- This proposed algorithm helps identify the consequences of security failure. By knowing the severity of security breach, we can add extra security mechanisms around the components with high severity and adding more security patterns so the probabilities and severities of security breaches on the database components for example are reduced.

Using security pattern as we discuss in the next section helps reduce the security risk factor by either reducing the probability or severity of security failure.

Figure 5- 5 The distributions of risk factors for the three components, customer agent, customer interface, and bookserver database

5.4 Security Design Patterns

There have been many security approaches to patch the systems vulnerabilities. However these approaches are considered temporarily solutions and after passing certain time they become out of date or insufficient solution to today's problems. These patterns are mostly implemented to cover security problems on the code level and to address specific problem. They don't know address the root of the problem. In other words they are reactive

to specific issues. In contrast, design patterns tell the architects how to design a system given a statement of problem and set of forces that act upon the system. Patterns provide information system architects a method for defining reusable solutions to design problems without ever having to talk about or write program code; they are truly programming language-independent. The following two examples [2] provide solutions of security design patterns, and a methodology for using those patterns to design a secure system, which will enable software architects and system designers to produce system architecture that meets security requirements and could be used methods of validation when using these patterns.

5.4.1 Example1: Secure Communication Pattern:

A Connector between two components or component and an actor may be subject to various security threats. The security provided by an external actor will not be effective if it is exposed by attackers on the connectors. Therefore it is desirable to protect the connector.

Some attacks against the connectors carrying the messages:

- 1- Unauthorized Disclosure of Traffic
- 2- Unauthorized Modification of Traffic
- 3- Impersonation of an actor to the connection

Applicability:

We consider using this pattern when:

- 1- Sensitive information is exchanged between the actors and the system
- 2- Traffic in the connectors may be subject to security attacks

Structure: Figure 5- 6 shows the structure of Secure Communication Pattern.

Figure 5- 6 Structure of Secure Communication Pattern

Participants:

- 1- External Actor/Component: the source and or/destination of messages to be sent over the connector
- 2- Connector: Carries messages exchanged between actors and system's components
- 3- Proxy: Protect Traffic sent over the connector using one of a variety of protection mechanisms.

Collaborations:

- 1- An external actor submits a message to its proxy for protection
- 2- The actor proxy applies appropriate protection to the message
- 3- The actor proxy uses the connector to transmit the message to the component proxy.
- 4- Component proxy will verify the message and decode the message. And return the verified message to the receiver.

Figure 5- 7 shows the original sequence diagram without any security patterns. Figure 5- 8 shows the attack graph when customer agent is considered a goal node without adding any security patterns. Figure 5- 9 represents the sequence diagram of the Secure Communication Pattern. The actor will send the message to his proxy to protect this message.

The proxy then sends out the message on the connector. The component proxy then receives the protected message and does the verification part, and passes the message to the destination component.

Figure 5- 7: UML sequence diagram without any security patterns

Figure 5- 8: Attack Graph of customer agent without any security patterns

We choose one message from our scenario, and that is login(username, password). The actor sends this message to the actor proxy. The actor proxy adds more protection to this specific message and sends it out to the system through a connector. The customer interface proxy will receive the message and verify it and pass it to the customer agent. Figure 5- 10 shows the UML sequence diagram of the buy a book scenario when customer agent is chosen as a goal node and after adding the security communication pattern to it.

Consequences:

When using a secure communication pattern:

- 1- We ensure the messages and data carried inside the message sent over potentially insecure communication connector is protected against known threats
- 2- It may increase communication latency and costs
- 3- It may reduce communication throughputs
- 4- It may need to use certain security algorithms and need to comply with international standards

Figure 5- 9: UML sequence diagram of secure communication pattern

Implementation:

Secure communication proxies may apply one or more of the following types of protection to messages in order to counter threats anticipated in the communication connector:

- 1- Data Authentication protects against misrepresentation of identity of a sender of message
- 2- Peer Authentication protects against impersonation attack
- 3- Data Integrity protects against Tampering with data attacks

Figure 5- 10: UML sequence diagram of customer agent after adding secure communication pattern

Figure 5- 11 represents the attack graph of customer agent component after adding a security communication pattern to one connector. The path from C3 t3 Cci tia1 is eliminated. Thus estimating the probability of security failure of customer agent component is:

$$Cprob(g) = Oprob(t_7) + Oprob(t_8) + Oprob(t_{ia1}) = \frac{3.5}{8} VI$$

Table 5- 5 shows the probability of security failure of customer agent component before adding the secure communication pattern and after adding it. We can conclude that $RF2 < RF1$ after adding this pattern to the architecture assuming that the component still has the same severity.

Component	Probability of security failure before adding security pattern	Probability of security failure after adding security pattern
Customer agent	5/8 VI	(3.5)/8 VI

**Table 5- 5 security risk factors of customer agent component before /after
adding secure communication pattern**

Figure 5- 11: Attack Graph of customer agent after adding a security communication pattern

5.4.2 Example2: Policy Pattern:

Intent:

Isolate policy enforcement to separate components of sensitive data information. Ensure that policy enforcement activities are performed in the proper sequence.

Motivation:

Many software architectures, systems, and components need to enforce policy. In such cases, the policy enforcement must be invoked in the correct sequence at every time there is any access to certain database component. We could use the same policy enforcement to protect more than one database component.

Structure: Figure 5- 12 shows the structure of Policy Pattern.

Figure 5- 12: Structure of Policy Pattern

Participants:

- 1- External actor: request access to certain database component or resource in the architecture
- 2- Authenticator component: Authenticates external actors who may want to have access to certain components in the architecture.
- 3- Guards:
 - a. Collects users,
 - b. request, target and context attributes needed to make access control on certain component
 - c. Request access control decisions from the policy

- d. Rejects or grant requests based on the policy
 - e. Sequences messages operations related to policy enforcement.
- 4- Security Context: Keeps credentials and security information for use in policy decisions
- 5- Policy:
 - a. Makes decisions to grant or deny access to certain database components based on actor attributes, request attributes, context attributes, and policy
 - b. Encapsulates a set of Rules determining which actors can perform which operations on which components.
- 6- Rule: it is a component of the policy expressing the permission for a specified set of Clients to perform a specified set of operations on a specified set of database components

Collaborations:

- 1- The external actor requests access to the Gaurd
- 2- The authenticator causes the user's security attributes to be included in the effective security context
- 3- The Guard determines the request, target, and context attributes
- 4- The Guard requests a policy decision
- 5- The policy checks which of the set of rules matches the security attributes, requested operation, and the targeted resource
- 6- If the Rules indicate that the request should be denied, the Policy returns a notification to the Guard, which then passes a failure notification to the Client

- 7- If the Policy result indicated that the request should be granted, the Guard passes the request to the resource and replays the response back to the Client.

Consequences:

- 1- Ensures that security policy is checked before actor requests for database information access are granted
- 2- Provides a single point of control for policy related activities
- 3- Imposes performance overhead: Separating policy enforcement from request fulfillment will usually introduce network overhead.

In our example, a policy pattern could be implemented in the customer agent component when accessing customer information database component. Rule could be:

- 1- If Administrators credentials are authenticated and remote access to information database component is requested, deny access
- 2- If Administrators credentials are authenticated and local access to information database component is requested, grant access

Table 5- 6 illustrates the difference in severity when adding policy pattern to the architecture. $RF1 > RF2$ and $RF3 > RF4$.

Figure 5- 13: UML sequence diagram of Policy Pattern

Component	Severity of security failure before adding policy pattern	Severity of security failure after adding policy pattern
Customer agent	Catastrophic	Critical
Customer information database	Catastrophic	Critical

Table 5- 6 security risk factors of customer agent and customer information database components before/after adding policy pattern

5.4.3 Integrating Security Risk Assessment with other quality attributes:

In the previous sections, we introduced two design patterns that could be used in the design phase and improve the security attributes and risk of components. This leads to improvements in the quality of software attributes from security standpoint. However, adding such patterns for example could increase the response time of the components in the system to the external actors. This might result in violating the requirements of such a system, and could cause increase in the requirement risk of the components in the system. Therefore, it is important to integrate the software attributes such as requirement, reliability, maintainability, and security all during the early phases of software life cycle as improving the risk of one might cause increase for the others. Integration all these software attributes will improve the quality of software and will help avoid the rule of fix it later. In this research, we introduce our methodology at the early phases where it could be integrated with other risk assessment methodologies to improve the quality of overall software attributes. Similarly, improving the reliability risk assessment of software components could increase the security risk of these

components. Therefore, it is significant to integrate all these risk assessments when developing software at the early phases of software life cycle.

In this section, we show how security risk assessment could be integrated to the other types of risk assessment tool. Software Architecture Risk Assessment (SARA) tool [63] is a tool used to that support architectural level model-based risk assessment, which includes reliability-based risk, requirements-based risk and maintainability-based risk. This tool can be extended to cover the security risk assessment since the security risk assessment is based on UML specifications extracted on the architectural level. In the security risk assessment, we depend on the use cases, scenarios, and sequence diagram to develop the attack graph. We also depend on the reachability matrix that could be extracted from the UML sequence diagram. Access rights and data classifications of components could easily be extracted from the specifications of the components attributes. The Figure 5- 14 shows how security based assessment could be integrated with other assessment using SARA tool. UML software (i.e. RT Rose, Java, Visio) is used to provide the architecture software repository with use cases, sequence diagrams. Then the attack graphs, schedules, Reachability Matrix, access rights, and components specification are extracted from the repository to estimate the probability and severity of components.

Figure 5- 14 The architecture of the Software Architecture Risk Assessment (SARA) Tool

5.5 Microsoft Security Bulletin Analysis

In this section, a correlation between the patches introduced in the Microsoft security bulletins [37] and security risk assessment analysis is established. We prove that these resources are used to validate our technique using Microsoft security Bulletin analysis. Microsoft Security Bulletins [37] describe vulnerabilities and resources (Component, Data, and Connectors) that attacker can use to exploit the software components on the code level. Additionally MSB has its own way of rating the severity. This includes four levels Critical, Important, Moderate, and Low. In our research, there is a rating of severity level for each type of attack. Patching on the code level is the same as adding more secure layers to the components on the architecture level. The following Table 5- 7 compares the MSB on the code level with its equivalent on the architecture level.

Empirical studies are part of the empirical validation to validate security risk assessment. This can be done through several ways. One of these ways is to do empirical and experimental case studies and empirical data, another way is through surveys.

One of the challenges to establish such a validation is to correlate between the patches and our security risk assessment. Meaning not all patches are relevant to the security risk assessment in our scenarios, consequently a patch is correlated to our security risk assessment if we expect that patch to impact our software on the architectural level. Another challenge is to map (convert) the impact and elements of that patch from the code level to the architectural level. Additionally not all relevant patches will reduce the security risk factors. We conclude that building an empirical validation can't be established in short term, it may take years to complete the validation work.

MSB	Severity rating (Impact)	Resources	Affected Software	Likelihood	Protection
Code level	Critical, Important Moderate Low	Software components data, functions	Software components or data (i.e. Windows, IE, Database..)	Microsoft Exploitability Index	Apply patch
Architecture level	Catastrophic, Critical, Major, Minor, Low	Components, connectors, data,	Architectural Components, or data	Probability of Security failure	Adding secure layers to the component /connectors

Table 5- 7 Comparison between MSB on code level and its map on architectural level

The following Table 5- 8 shows the correlations between how patches could improve the security risk factors on code level and on the architectural level.

Level	Risk factors before applying the patch	Risk factors after applying the patch
Code level	Value 1	Value2
Architectural level	Rf1	Rf2

Table 5- 8 Security Risk Factor before and after applying patches

If $\text{value2} < \text{value1}$ and $\text{Rf2} < \text{Rf1}$, we can conclude the validation is established on the architectural level.

In this section, we show experimentally that a correlation between patching on code level and architecture could be built and will help improve the risk factors. [37] is a reference to all Microsoft security bulletins for the last 10 years. In our research, we will be studying some bulletins from 2004 list.

We will study the Patch MS04-012. Table 5- 9 explains the details of this vulnerability security bulletin and its mapping on the architectural level.

Patch #	MSB 04-012	Vulnerability on the Architectural level
Impact of Vulnerability	Remote Code Execution	Change, delete, review data
Maximum severity rating	Critical	Catastrophic
What might an attacker use the vulnerability to do?	Take complete control of an affected system, including installing programs; viewing, changing, or deleting data; or creating new accounts that have full privileges.	Unauthorized review of data, modify data,
How could an attack exploit this vulnerability?	An attacker could log on to the system interactively or by using another program that passes parameters to the vulnerable component (locally or remotely).	Log on to the system remotely, send certain messages to the components
Affected software	Remote Procedure Call (RPC) in Operating system	Customer agent component
Severity before applying patch	Critical	Catastrophic
Mitigation factor	Apply Patch	Apply the policy pattern (No log on remotely with Admin rights)
Severity after applying patch	Low	Major

Table 5- 9: Vulnerability of MSB04-012 and its maps on the architectural level

Figure 5- 15 shows the UML sequence diagram and the policy pattern applied on the customer agent component. The policy pattern will help either approve the log on locally with admin rights or decline the log on remotely with admin rights. In the example we showed that the severity of security failure is reduced by one level as no admin rights are granted for remote users.

Figure 5- 15: UML sequence diagram with MSB04-012 vulnerability and policy pattern

We will study another Patch MS04-023. Table 5- 10 explains the details of this vulnerability security bulletin and its mapping on the architectural level.

Patch #	MSB 04-023	Vulnerability on the Architectural level
Impact of Vulnerability	Remote Code Execution	Change, delete, review data
Maximum severity rating	Critical	Catastrophic or Critical
What might an attacker use the vulnerability to do?	An attacker who successfully exploited this vulnerability could gain the same privileges as the user. Users whose accounts are configured to have fewer privileges on the system would be at less risk than users who operate with administrative privileges.	The hacker has the same privilege as the user logging on.
How could an attack exploit this vulnerability?	To exploit this vulnerability, an attacker would have to host a malicious Web site and then persuade a user to view that Web site. An attacker could also create an HTML	Through the connectors carrying the messages between the outside world and the system,

	e-mail message that contains a specially crafted link, and then persuade a user to view the HTML e-mail message and then click the link.	
Affected software	Internet protocol	Customer agent, customer interface components
Mitigation factor	Apply Patch	Apply the secure communication pattern (on the connectors when a user logon)
Probability after applying patch	Reduced	(3.5/8) VI

Table 5- 10: Vulnerability of MSB04-023 and its map on the architectural level

Figure 5- 16 shows the UML sequence diagram with secure communication pattern. This pattern will confirm that the logon process between the actor and the system is secured and no attacker can intercept this message and exploit this vulnerability. Patch MS04-023 discusses one of the internet protocols that have some functionality where an attacker could exploit to take a complete control of a system. This vulnerability is eliminated on the code level by applying the patch. Since we are assuming in our example that the internet is used to carry the messages between the actors and the components of our system, the vulnerability is existed and could be exploited if no design pattern is placed. In our example we take proactive step by applying the secure

communication pattern. An example of such a pattern is a digital signature between the two parties, or what we call the actor and system proxies.

Figure 5- 16: UML sequence diagram with MSB04-023 vulnerability and secure communication pattern

5.4 Conclusions

Software Engineering processes are changing rapidly, and there are many factors involved in these processes. Varieties of software development techniques make it hard to get statistical sets for empirical studies. However the empirical results obtained from using limited resources are still valuable. Validation could be considered one of research areas that needs a big attention and a huge effort and might take long time to complete.

Chapter 6

Conclusion and Future Work

Several conclusions emerge from this research. We conclude that this work is a promising and significant step in meeting our research objectives: to develop a security risk assessment methodology based on measurable parameters that could be collected and analyzed in the early software design phase based on UML artifacts, Attack Graph, data sensitivity, access rights and reachability matrix.

This research develops an architectural-level security-based risk assessment methodology. In this research we propose a methodology for security risk assessment based on the UML specifications such as uses cases, sequence diagram that can be used in the early phases of software life cycle. The proposed methodology uses attack graph, access rights, data sensitivity, and reachability matrix.

The risk assessment methodology presented in this proposal considers component security risk factors. It is used for calculating the risk factors of various components at the architectural level. We combine the probability of software security element failure with the severity of that failure to estimate the risk factor of software architectural elements.

We estimate the probability of security element failure based on the UML sequence diagrams and attack graphs. We develop the attack graph for each component from a given UML sequence diagram. Then we use probabilistic arguments to estimate the probability of security failure for that component. The proposed metrics could be obtained at early development phases from UML models.

To estimate the severity of security failure we present a new methodology that takes into the account multiple factors when evaluating the worst case consequences. These factors are component classifications based on data sensitivity, access rights, and reachability matrix for elements in the system. This methodology describes an algorithm for estimating severity of each software architectural component. Data sensitivity analysis helps classify database components based on the significance of data that each component uses. Access rights determine whether a message is executed with admin rights or non admin rights. Reachability matrix presents how high or low the components are coupled with each other.

We conducted two studies to validate our proposed methodologies based on the design security patterns. We applied these studies on Microsoft security bulletins.

In summary the proposed methodologies are efficient methods to estimate security risk factors on different levels of software design. They enable us to estimate software components risk factors that enable us to focus on the high risk elements even though they may be rarely used and therefore may not contribute significantly to the overall system risk factor.

Our future work is focused on generalization of the methodology presented in this research. Another future work could be is to automate the process of estimating the security risk assessment of more complex systems since our methodologies are entirely analytical and provide a closed form solution.

In the future, we could also do the following:

- Extend the security risk assessment methodology to handle cloud computing solution as the future businesses are emerging in this area.

- Integrate the current security risk assessment tool with other kinds of risk tools (i.e. maintainability risk, performance risk, requirement risk, reliability risk)
- Extend the mitigation actions required as a result of risk assessment by developing more security patterns and integrate these patterns on the architectural level case studies.
- Extend the testing of validations on the architectural level before the systems go live to implementation or code level.
- For large scale system where big attack graphs are built using an automated tool, we may need to use Bayesian network to calculate the probabilities of intermediate nodes, and goal nodes.
- Extend the proposed algorithm to be applied not only at the architectural or Application Programming Interface (API) level, but also at physical layer, network level, session layer, and data link layer in Open Systems Interconnection model (OSI).

References

- 1- [Aagedal, 2002] J.O. Aagedal, F.D. Braber, T. Dimitrakos, B.A. Gran, D. Raptis, K. Stolen, “*Model based Risk Assessment to Improve Enterprise Security*”, Enterprise Distributed Object Computing Conference, EDOC '02. Proceedings. Sixth International, pp. 51-62, 2002
- 2- [Blakley, 2004] B. Blakley, C. Heath, and Members of the Open Group Security Forum, “*Security Design Patterns: Open Group Technical Guide*”, 2004.
- 3- [Booch, 1999] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", ISBN: 0-201-57168-4, Addison Wesley, est. publication December 1999.
- 4- [Bowles, 2008] J.B. Bowles, W. Hanczaryk, “*Threat Effects Analysis: Applying FMEA to Model Computer System Threats*”, Reliability and Maintainability Symposium, RAMS. Annual. pp. 463 – 468. 2008.
- 5- [Briand, 1993] L. Briand, V. Basili, C. Hetmanski, “*Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components*,” IEEE Transactions on Software Engineering, vol. 19. No. 11, pp. 1028-1044, 1993.
- 6- [Briand] L. Briand, K. El Emam, and S. Morasca. “*Theoretical and empirical validation of software product measure*”. Technical Report number ISERN-95-03, International Software Engineering Research Network, 1995.
- 7- [Buschmann, 1996] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, “*Pattern Oriented Software Architecture: A System of Patterns*”, John Wiley & Sons, 1996.

- 8- [Chen, 2007] Y. Chen, B. Boehm, L. Sheppard, “*Value Driven Security Threat Modeling Based on Attack Path Analysis*”, 40th Annual Hawaii International Conference on System Sciences, pp. 280a, 2007.
- 9- [CNBC1, 2011] <http://www.cnbc.com/id/43335148>
- 10- [CNBC2, 2011] <http://www.cnbc.com/id/43079509>
- 11- [CNN, 2011] http://news.blogs.cnn.com/2011/06/19/customers-personal-information-stolen-from-sega-database/?hpt=hp_t2
- 12- [Dimitrakos, 2002] T. Dimitrakos, B. Ritchie, “*Model based Security Risk Analysis for Web Applications: The CORAS approach*”, Euroweb , The Web and the GRID, from escience to ebusiness, 2002.
- 13- [El-Emam, 1999c] K. El-Emam, S. Benlarbi, N. Goel, and S.N. Rai, “*Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components*”, Journal of Systems and Software, Volume 55 Issue 3, pp. 301 – 320, 2001.
- 14- [Feng, 2008] C. Feng, S. Jin-Shu, ”*A Flexible Approach to Measuring Network Security Using Attack Graphs*”, International Symposium on Electronic Commerce and Security, 2008, ISBN 0769532586, pp. 426 – 431, 2008.
- 15- [Fenton, 1991] N. Fenton, “*Software Metrics: A rigorous Approach*”, Chapman & Hall, Ltd. London, UK, ISBN: 0442313551, 1991.
- 16- [Fenton, 1999] N. Fenton, S. Pfleeger, R. Glass “*Science and Substance: A Challenge to Software Engineers*”, IEEE Software, volume : 11 , Issue:4, pp. 86 – 95, 1994.
- 17- [Folwer, 2004] M. Fowler, “*UML Distilled 3rd edition*”, Addison Wesley, 2004.

- 18- [Hecker, 2008] A. Hecker, “*On System Security Metrics and the Definition Approaches*”, the Second International Conference on Emerging Security Information, Systems and Technologies, pp. 412 – 419, 2008.
- 19- [Heyman, 2008] T. Heyman, R. Christophe, S. Huygens, W. Joosen. “*Using security patterns to combine security metrics*”. Third International Conference on Availability, Reliability and Security, ISBN 0769531024, pp. 1156 - 1163, 2008.
- 20- [Hoglund, 2004] G. Hoglund and G. McGraw, “*Exploiting Software: How to Break Code*”, Addison Wesley, ISBN-10: 0201786958, 2004.
- 21- [Howard, 2002] M. Howard and D. LeBlanc, “*Writing Secure Code*”, Microsoft Press, 2002.
- 22- [Howard, 2003] M. Howard, J. Pincus, and J. Wing, “*Measuring relative attack surfaces*” in Proc. of Workshop on Advanced Developments in Software and Systems Security, 2003.
- 23- [Jacobson, 1992] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard “*Object Oriented Software Engineering*”, Addison-Wesley Professional, ISBN-10: 0201544350, 1992.
- 24- [Jansen, 2009] W. Jansen, “*Directions in Security Metrics Research*” Computer Security Division, National Institute of Standards and Technology (NISTIR), Volume 7564, 2009.
- 25- [Jaquith, 2007] A. Jaquith, “*Security Metrics: Replacing Fear, Uncertainty, and Doubt*” Addison Wesley, ISBN-10: 0321349989, 2007.
- 26- [Jha, 2002] S. Jha, O.S. Sheyner, J. Wing, “*Two Formal Analyses of Attack Graphs*”, Computer Security Foundations Workshop, Proceedings. 15th IEEE, pp. 49 – 63, 2002.

- 27- [Kitchenham] B. Kitchenham, S. L. Pfleeger, and N. Fenton. “*Towards a Framework for Software Measurement Validation*”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, ISSN 0098-5589, Volume 21, Issue 12, pp. 929 – 944, 1995
- 28- [Krautsevich, 2010] L. Krautsevich, F. martinelli, A. Yautsiukhin, “*Formal approach to security metrics, what does “more secure” mean for you?*”, Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, 2010.
- 29- [Liu, 2007] Y. M. Liu, I. Traore, “*Properties for Security Measures of Software Products*”, 2007.
- 30- [Lund, 2004] M.S. Lund, F. D. Braber, K. Stolen, F. Vraalsen, “*A UML profile for identification and analysis of security risks during structured brainstorming*”, SINTEF ICT, ISBN-10: 8214031109, 2004.
- 31- [Madan, 2002] B.B. Madan, K.G. Popstojanova, K. Vaidyanathan, K.S. Trivedi, “*Modeling and Quantification of Security Attributes of Software Systems*”, International Conference on Dependable Systems and Networks, Proceedings, pp. 505 - 514 2002.
- 32- [Manadhata, 2010] P.K. Manadhata, J.M.Wing, “*An Attack Surface Metric*”, IEEE Transactions on Software Engineering, ISSN 0098-5589, Volume 37, Issue 3, pp. 371 – 386, 2011.
- 33- [McGraw, 2004] G. McGraw, “*Software Security*”, IEEE Security & Privacy Magazine, ISSN 1540-7993, Volume 2, Issue 2, pp. 80 - 83, 2004.
- 34- [McGraw, 2006] G. McGraw, Software Security: Building Security In. Addison Wesley, ISBN-10: 0321356705, 2006.

- 35- [Meland, 2008] P.H. Meland, J. Jenson “*Secure Software Design in Practice*”. Third International Conference on Availability, Reliability and Security, ISBN 0769531024, pp. 1164 – 1171, 2008.
- 36- [Morasca,] S. Morasca, L. C. Briand, V. R. Basili, E. J. Weyuker, and M. V. Zelkowitz. Comments on “*towards a framework for software measurement validation*”, IEEE Transactions on Software Engineering, ISSN 0098-5589, Volume 23, Issue 3, pp. 187 – 189, 1997
- 37- [MSB] Microsoft Corporation. Microsoft security bulletin search. <http://www.microsoft.com/technet/security/current.aspx>
- 38- [NASA, 1997] NASA-STD-8719.13A, “*Software Safety NASA Technical Standard*”, 1997.
- 39- [Neuhaus, 2007] S. Neuhaus, T. Zimmermann, C. Holler, A. Zeller, “*Predicting Vulnerable Software Components*”, Proceedings of the 14th ACM conference on Computer and communications security, 2007.
- 40- [OMG] Object Management Group <http://www.omg.org>
- 41- [Payne, 2006] S.C. Payne, “*A Guide to Security Metrics*”, SANS Institute ReadingRoom 2006.
- 42- [Pfleeger, 2010] S. L. Pfleeger, J. M. Atlee, “*Software Engineering , Theory and Practice*” 4th edition Prentice Hall, ISBN-10: 0136061699, 2009.
- 43- [Popstojanova, 2003] K. Goseva-Popstojanova , A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, H. Ammar, A. Mili, “*Architectural-Level Risk Analysis using UML*”, IEEE Trans. Software Engineering, Vol. 29, No.10, October 2003.

- 44- [Rajput, 2000] W.E. Rajput, “*E-Commerce Systems Architecture and Applications*”, Artech House, **ISBN-10:** 1580530850, 2000.
- 45- [Rumbaugh, 1997] J. Rumbaugh, I. Jacobson, and G. Booch, “*Unified Modeling Language Reference Manual*”, Addison Wesley, ISBN: 0-201-30998-X, 1997.
- 46- [Sahinoglu] M. Sahinoglu, “*Security Meter: A Practical Decision-Tree Model to Quantify Risk*”, IEEE Security & Privacy Magazine, ISSN 1540-7993, Volume 3, Issue 3, pp. 18 – 24, 2005.
- 47- [Savola, 2007] R. Savola, “*Towards a Security Metrics Taxonomy for the Information and Communication Technology Industry*”, International Conference on Software Engineering Advances (ICSEA 2007), ISBN 0769529372, p. 60, 2007.
- 48- [SEI, 2005] http://www.sei.cmu.edu/architecture/sw_architecture.html Software Architecture for Software-Intensive Systems, 2005.
- 49- [Sharma, 2006] V.S. Sharma, K.S. Trivedi, “*Quantifying software Performance, reliability and security: an Architecture-based approach*”, JOURNAL OF SYSTEMS AND SOFTWARE, ISSN 0164-1212, Volume 80, Issue 4, pp. 493 – 509, 2006.
- 50- [Shaw, 1996] M. Shaw, D. Garlan, “*Software Architecture: Perspectives on an Emerging Discipline*”, Prentice-Hall, ISBN-10: 0131829572, 1996.
- 51- [Shaw, 1997] M. Shaw, “*Architectural issues in software reuse: It’s not just the functionality, it’s the packaging*”, Proceedings of the 1995 Symposium on Software reusability, Volume 20 Issue SI, pp. 3-6, 1995.
- 52- [Sherer, 1988] S. A. Sherer, “*Methodology for the Assessment of Software Risk*”, Ph.D. Diss., Wharton School, University of Pennsylvania, 1988.

- 53- [Chidamber, 1998] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, “*Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis*”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, ISSN 0098-5589, Volume 24, Issue 8, pp. 629 – 639, 1998.
- 54- [Spinnellis, 2006] D. Spinnellis, “*Code Quality: The Open Source Perspective*” Addison Wesley, ISBN-10: 0321166078, 2006.
- 55- [Taylor, 2010] R.N. Taylor, N. Medvidovic, E.M. Dashofy, “*Software Architecture Foundations, Theory, and Practice*”, Wiley, ISBN-10: 0470167742, 2009.
- 56- [UML, 2005] Unified Modeling Language, OMG, <http://www.uml.org>
- 57- [Vaughn, 2001] R. B. Vaughn, R. Henning, “*Are Measures and Metrics for Trusted Information Systems Possible?*”, Workshop on Information Security System Scoring and Ranking, Williamsburg, VA, USA, 2001.
- 58- [Venturebeat, 2011] <http://venturebeat.com/2008/10/25/hacker-finds-a-security-hole-in-the-google-phone/>
- 59- [Viega, 2001] J. Viega and G. McGraw, “*Building Secure Software: How to Avoid Security Problems the Right Way*”, Addison Wesley, ISBN-10: 020172152X, 2001.
- 60- [Wang, 2009] J. A. Wang, M. Guo, and M. Xia. “*Security Metrics for Software Systems*”, Proceedings of the 47th Annual Southeast Regional Conference, 2009.
- 61- [Weyuker] E.J. Weyuker. “*Evaluating software complexity measures*”. IEEE Transactions on Software Engineering, ISSN 0098-5589, Volume 14, Issue 9, pp. 1357 – 1365, 1988
- 62- [Zuse, 1998] H. Zuse, “*A framework of software measurement*”, Walter De Gruyter Inc, ISBN-10: 3110155877, 1998.

63- [K. Shaik, 2006] K. Shaik, W. Abdelmoez, H. Ammar, “*Software Architecture Risk Assessment (SARA) Tool*”, ISSRE Supplementary Conference Proceedings, 2006

64- [privacy.org] (<http://www.privacyrights.org/data-breach>)