IJCSN - International Journal of Computer Science and Network, Volume 3, Issue 6, December 2014
ISSN (Online): 2277-5420 **www.IJCSN.org**
Impact Factor: 0.274

Security Risk Assessment of Software Architecture, Methodology and Validation

[1] Fadi HajSaid, [2] Yousef Hassouneh, [3] Hany H. Ammar

[1] Microsoft Corporation
New York, NY 10036, USA

[2] Computer Science Department, Birzeit University
Birzeit, West Bank, Palestine

[3] Lane Computer Science and Electrical Engineering Department, West Virginia University
Morgantown, WV 26505, USA

Abstract - Security risk assessment is considered a significant and indispensable process in all phases of software development lifecycles, and most importantly at the early phases. Estimating the security risk should be integrated with the other product developments parts and this will help developers and engineers determine the risky elements in the software system, and reduce the failure consequences in that software. This is done by building models based on the data collected at the early development cycles. These models will help identify the high risky elements. In this paper, we introduce a new methodology used at the early phases based on the Unified Modeling Language (UML), Attack graph, and other factors. We estimate the probability and severity of security failure for each element in software architecture based on UML, attack graph, data sensitivity analysis, access rights, and reachability matrix. Then risk factors are computed and validation studies are conducted. An e-commerce case study is investigated as an example.

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

1. Introduction

Risk assessment involves many activities including risk prioritization in which the risk exposed based on the probability of risk occurrence and its impact on software quality [13]. Performing it at 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 elements that need careful attention in development and throughout all phases of software lifecycles. We are concerned with security-based risk of software architecture. The risk can be defined with two parts, the probability and the severity. In our case, the security risk assessment consists of two parts, probability of security failure and the consequence of such a security failure. According to [4], software security can be defined as "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." Over the last many years, the security attacks on software have grown significantly. The attack can be defined as a means of exploiting a vulnerability, which is defined as an error or weakness in design, implementation, or operation [10]. Security incidents reported to the Computer Emergency Readiness Team Coordination Center (CERT/CC) rose 2,099 percent from 1998 through 2002, an average annual compounded rate of 116 percent. Most of these incidents resulted from software vulnerabilities. Such vulnerabilities can impact critical infrastructure, as well as commerce and security [11]. Therefore, the importance of software security has been manifested by many researchers and practitioners. Furthermore it has been shown that the earlier we incorporate security in the software the better would be in term of effort and cost [5].

In this paper, we develop a new methodology to estimate the security risk assessment at the architectural level based on UML, Attack Graph, data significance, access rights and other factors. First we estimate the probabilities of security failure for each elements in each scenario by building the attack graph. Second we estimate the severity of security failures for each element based on UML and multiple factors (element classification, data significance,

reachability matrix, and access rights). An e-commerce case study will be conducted in our analysis.

2. Related Work

This research is different from previous work on security assessment in many important aspects. First aspect, our methodology is based on UML modeling of the software at the architectural level. Prior work was based on the production and code level, and their assessment was conducted after the system was tested and in production. No mathematical models and probabilistic arguments were used [8]. Our approach adopts the software systems at architectural level. Additionally our approach uses mathematical model to estimate the two parts of risk factors, probabilities and severities. Second aspect, the attack graph was built based on the attacker knowledge of system protocols and method of implementations, and the analysis was conducted on network and in production system. The risk analysis was defined based on only probabilities without estimating the severity work [3]. In contrast, our methodology estimates probabilities and severity for all components and connectors without the need to know the protocols used. Third aspect, much previous work on threat modeling was based on the attacker capabilities in exploiting vulnerabilities and counting all these vulnerabilities quantitatively without really estimating the probabilities and severities of attacks [7][1][6]. In our work, we take the security risk assessment measurement based on system knowledge and not based on the attacker power and behavior and ability to attack the software system.

In Summary, this paper is the first to mathematically and systematically estimate the security risk assessment of software system at the architectural level. Our approach is proactive, where previous works were reactive.

3. Security Risk Assessment Methodology

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. Fig 1 shows the proposed algorithm. 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. Fig. 2 shows the Sequence Diagram for a buy a book scenario. In the following subsection A, we present the methodology of estimating the probability of security failure. Then in subsection B, we present the methodology of estimating the severity of security failure. In subsection C, we estimate components risk factors.

3.1 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 [3] can be represented as a Tuple

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

C_0 is a set of initial nodes. The initial nodes are the initial contact points where the attacker initiates the attack. They are on the side of the actors. T is a set of exploit nodes. They represent the messages between the system and the components or messages between the components. The attacker exploits these messages in order to reach to other elements in the system. C_d is a set of intermediate condition nodes. The Intermediate condition 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. E is a set of edges between nodes (conditions and exploits). Fig 4 shows the attack graph for the customer agent component.

$$C_0 = \{c_1, c_2, \ldots, c_8\}, Cd = \{c_{cci}\}$$
$$T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{ia1}, t_{ia2}\}$$

Eprob(τ) denotes the exploit success probability of τ. It is the probability of exploiting the message τ. Cprob(c) denotes the condition obtained probability of component c. It represents the probability of reaching component c. Oprob(τ) denotes the successful occurrence probability of exploit τ. It represents the probability of successful occurrence of message τ coming from components c1, c2,…,ck

$$Oprob(\tau) = Eprob(\tau).Cprob(c1)\ldots Cprob(ck) \quad (2)$$
$$Cprob(c) = \sum_{j=1}^{p} Oprob(\tau_j) \quad (3)$$

where c is an intermediate component or goal component.

Where $\tau_1, \ldots, \tau_p$ 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. We will pick the customer agent in Fig 2 as a goal node.

2- Extract the initial set $C_0$from the UML sequence diagram. In Fig. 2 $C_0 = \{c_1, c_2, \ldots, c_8\}$, where m=8 represents the number of the total direct messages exchanged between the system and outside world. We assume there is an attack on the system through one of these initial nodes. We assume all initial nodes have equally likely distribution,

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

VI [13] denotes to 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 $\mathbf{T'}$ of exploit nodes set T($\mathbf{T' \subset T}$) $\mathbf{T'} = \{t_1, t_2, \ldots, t_8\}$. $\mathbf{T'}$ represents the direct messages exchanged between the external actors and the system. If we assume all these external messages have equally likely distribution,

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

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

4- Extract a partial set of the intermediate condition nodes. These represent the components in our system connected to the external actors through set $\mathbf{T'}$. In Fig.2, the customer interface component is an intermediate condition node. In Fig. 4, 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 Fig. 2, The internal messagessend(username, password) and buy(book, creditcard) are the messages produced by the component customer interface node. In Fig. 4, these two messages are (t_{ia1}, t_{ia2}) to the customer agent goal node.

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 [12]. 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. Schedule1:Accessmainpage(){1.1}, mainpage(){1.2}. Schedule 2, 3, 4, and 5 can easily be obtained from the UML sequence diagram in Fig. 2.

After developing the attack graph for each component, we use the Eq. (2&3) to estimate the probability of security failure of the goal node.

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

The exploits nodes (t_{ia1}, t_{ia2}) 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 $Eprob(t_{ia2})$ 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. In Fig. 2, we have 6 components, the following MA(6*6) shows messages exchanged in this example.

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

$$Eprob(t_{ia1}) = Eprob(t_{ia2}) = 1/4$$

We can estimate $Oprob(t_{ia1})$& $Oprob(t_{ia2})$ as the following:

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

$$Oprob(t_{ia2}) = Eprob(t_{ia2}) * Cprob(c_{ci}) = \frac{1}{4} * (VI\frac{6}{8})$$

$$Cprob(g) = Oprob(t_7) + Oprob(t_8) + Oprob(t_{ia1}) + Oprob(t_{ia2})$$

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

Cprob(g) is the estimated probability of security failure for customer agent component. Similarly, we can estimate the probabilities of security failures for the other components. Table 1 shows the probabilities of security failures for each component in this specific scenario.

- For each use case
  - For each scenario
    - Identify components, connectors, schedules from UML sequence diagram
    - For each component
      - Identify the messages, data, connectors related to that component
      - Build the attack graph
      - Estimate the probability of security failure from attack graph
      - Estimate severity of security failure based on component classification, Access rights, and Reachability Matrix
      - Calculate risk factor
    - Sort the list of components risk factors

Fig 1. The security risk analysis algorithm.

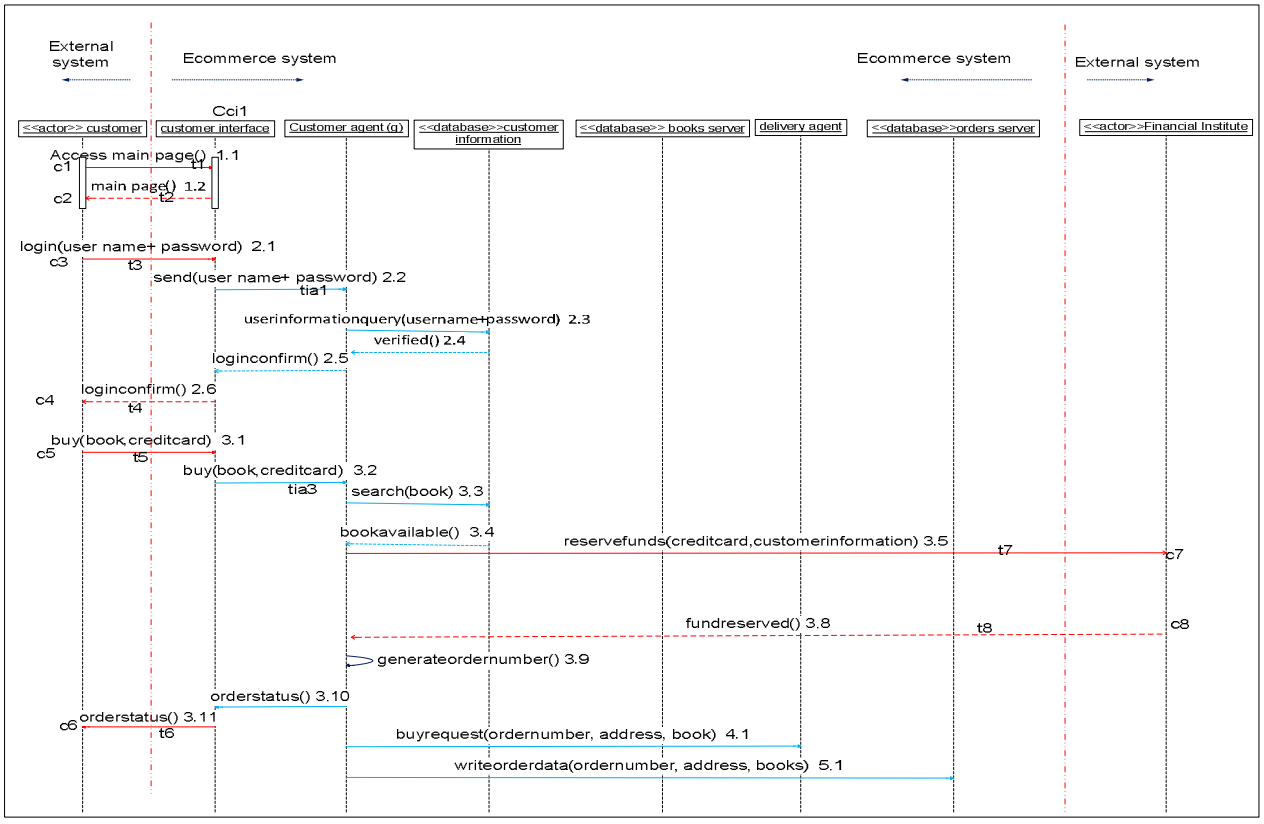

Fig. 2 Sequence diagram of the buy book scenario

.

3.2 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 [14] 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 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. Fig. 3 describes the steps to estimate the impact on the system when security failures occur. In our case study, we estimate the severity of customer information database component. Since this is a database component, we check the data sensitivity. This database stores the customer username and password for customers. Any security breach on this database will lead to security breach to the whole system, therefore the sensitivity is high and severity is Catastrophic (Access right is admin).According to the algorithm, we assign value of 5 to the customer information database component severity. Similarly, customer agent is non-base component; however it has connection with a customer information database. Therefore it has the same severity level. After normalizing the severity, the severities for customer information database and customer agent components become (1).

3.2 Security Risk Factor

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 Prob(c) $\{0 \leq Prob(c) \leq 1\}$ is the probability of security failure of a component in a scenario, and severity (c) $\{0 < severity\ (c) \leq 1\}$ 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. This process is repeated for each scenario in a given use case. However due to the limited space in this paper, we will not discuss the calculations of risk factors for the scenarios, uses cases, and the whole system. Table 1 shows the probabilities, severities, and risk factors of all components in buy a book scenario.

Table. 1. Probabilities, severities, and risk factors of all components in buy book scenario.

Component	Probability	Severity	Risk Factor
Customer Interface	VI(6.5/8)	.6	.48VI
Customer Agent	VI(5/8)	1	.625VI
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

4. Validation Studies

In this section, we describe different techniques to validate the methodology in the previous section. We present two techniques for validation; the first is based on security design patterns, and the second is based on sensitivity analysis. The use of security design patterns should reduce security risk factors. We test this fact using two different security patterns and show that the estimated security risk factors were indeed reduced as expected after

using these patterns. In the second technique, we use sensitivity analysis and show how the estimated risk factors change with the vulnerability index and initial probability distribution of the attack graph.

For each component

- If the component is a database element
  - Check data sensitivity
    - Assign severity based on data sensitivity (Critical =high sensitive, Major= medium sensitive, Minor= low sensitive)
- Else // component is not a database element
  - if the component has a direct connection with a database component
    - Assign database component severity to the non-database component
  - Else// no direct connection with database
    - Assign severity based on reachability (Minor= high reachable, Low = low reachable)
- Check Access rights of the component
  - If (Access rights == admin)
    - Increase severity level by one level
- Normalize the component's severity

Fig. 3. The security severity analysis algorithm.

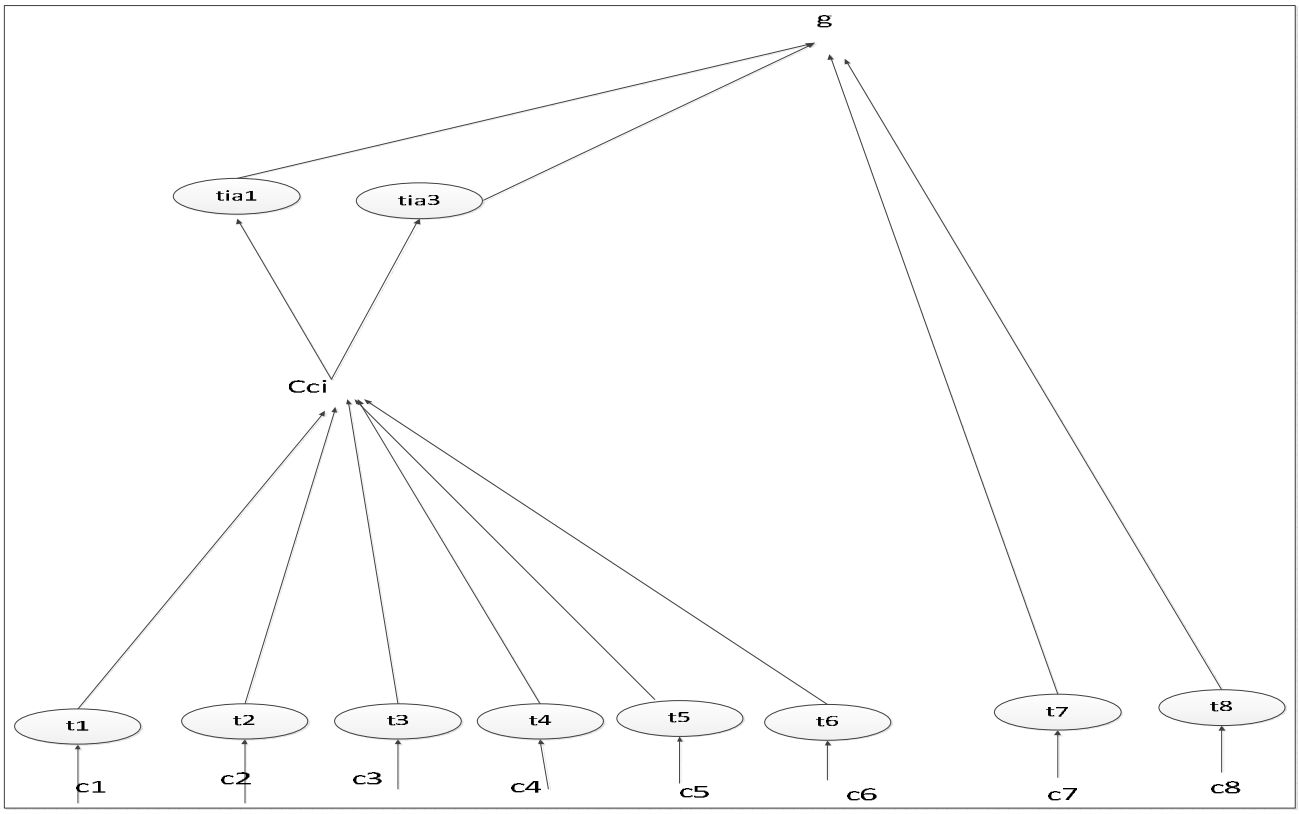

Fig. 4. Attack graph of customer agent component.

4.1 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 solutions to today's problems. Patterns provide information system architects a method for defining reusable solutions to design problems. The following two examples [2] describe two security design patterns, and a methodology for using those patterns to design a secure system. In the first example, we use a secure communication pattern and show how this pattern will help reduce the attacks on the ecommerce example. 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: Unauthorized Disclosure of Traffic, Unauthorized Modification of Traffic, or Impersonation of an actor to the connection.

Fig. 5 shows the structure of Secure Communication Pattern, and Fig. 6 shows the UML sequence diagram of secure communication pattern. This pattern may be considered when sensitive information is exchanged between the actors and the ecommerce system, or when traffic in the connectors may be subject to security attacks. In our example, this an external actor submits a message to its proxy of protection, the actor proxy applies appropriate protection to the message, then the actor proxy uses the connector to transmit the message to the component proxy, and lastly the component proxy will verify the message and decode the message and return the verified message to the receiver. Although using such a pattern will decrease the security risk on a system's component we show, however it may increase communication latency and the associated costs. We choose one message from ecommerce 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. Fig.7 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. 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 2 shows the security risk factor (RF) 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.

Table. 2 security risk factors of customer agent components before and after adding secure communication pattern

Component	Probability	Severity	Risk Factor
Customer Agent without pattern	VI(5/8)	1	.625VI
Customer Agent with secure pattern	VI(3.5/8)	1	.43VI

Fig.7 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.

In the second example, we use a policy pattern to show that this pattern could decrease the severity of security failure for a certain component. As a result, the risk factor of that component will improve. In many software architectures, systems and components, architects 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 a certain database component. We could use the same policy enforcement to protect more than one sensitive component.Fig.8 show the structure of policy pattern. The external actor requests access to certain database or resource in the architecture. The authenticator component does the authentication for specific components in the system. Guards collect user's information (actors), requests attributes needed to make access control on specific system's components, and rejects or grant policy based on the feedback from the policy. Security Context keeps credentials and security information. In the ecommerce case study, a policy pattern could be implemented in the customer agent component when accessing customer information database component. The rule applied would be if administrators credentials are authenticated and remote access to information database component is requested, deny access. If Administrators credentials are authenticated and local access to information database component is requested, grant access.

Table 3 illustrates the risk factors of customer agent component when applying two security patterns (security communication pattern and policy pattern). The risk

factor of customer agent component before adding these two patterns is .625VI, and after adding the two patterns is .35VI. We notice that there is a significant improvement in the risk factors of the components when using such patterns at the architectural level.

Table 3: security risk factor of customer agent component before/after adding two patterns

Component	Probability	Severity	Risk Factor
Customer agent without patterns	VI(5/8)	1	.625VI
Customer agent after adding two patterns	VI(3.5/8)	.8	.35VI

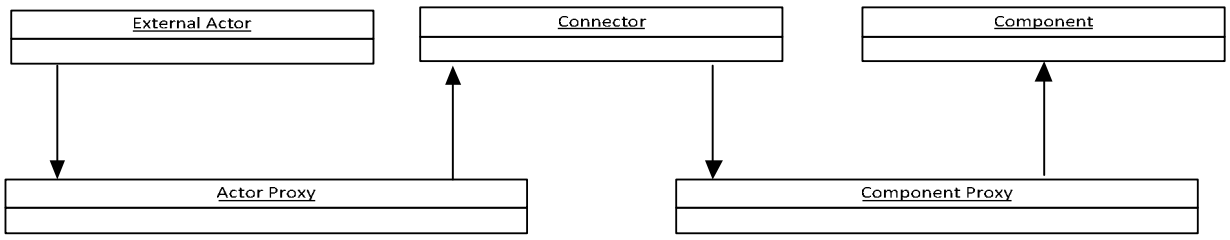

Fig. 5 Structure of Secure Communication Pattern

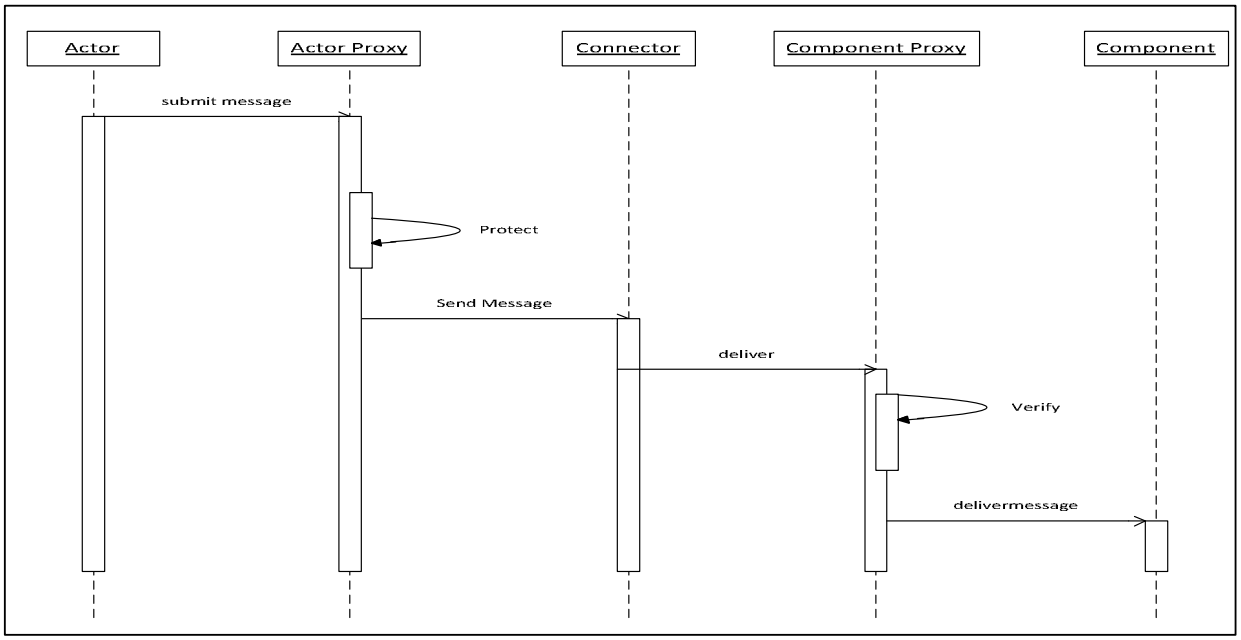

Fig. 6 UML Sequence Diagram of Secure Communication Pattern

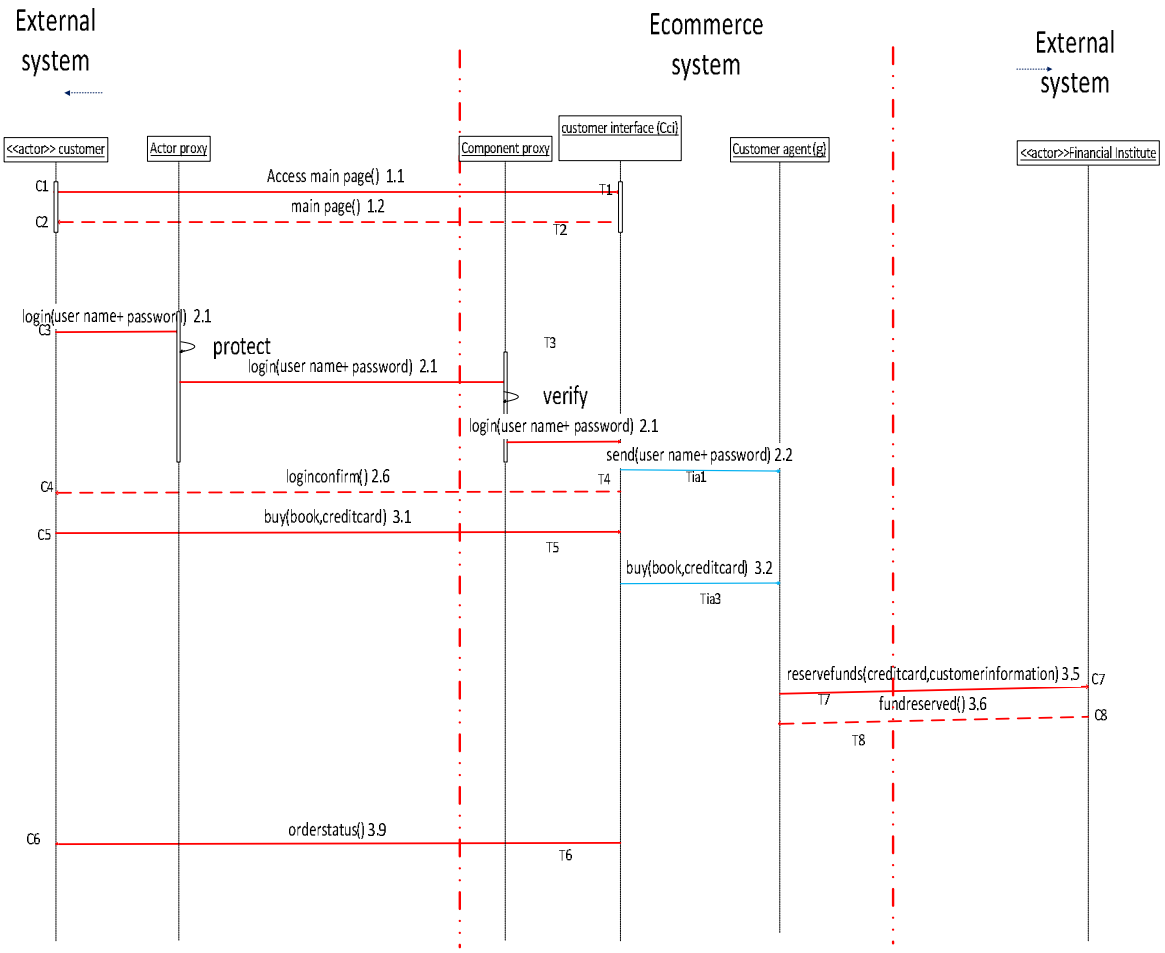

Fig. 7 UML Sequence Diagram of customer agent after adding secure communication pattern

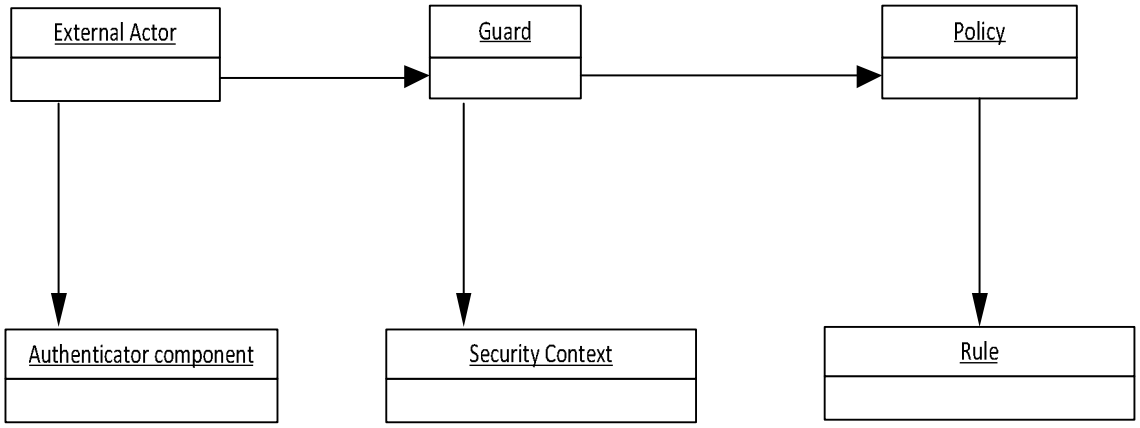

Fig. 8 Structure of Policy Pattern

4.2 Validation through Sensitivity Analysis

In this section, we make some changes to the parameters in the case study 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. First 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. Fig. 9 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. In Fig. 9, we have changed the values of VI by .05. We notice that 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. We can conclude that there are components more sensitive to the system parameters than other components and this could have more impact on value of those components' risk factors. We discuss in this section the severity analysis of the ecommerce system when changing the access rights of each components. We test the severity of e-commerce components when we change the access rights. Fig. 10 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. We can imply that there are components in the system that has high level of severity with or without admin access rights assigned like the customer agent and customer interface. On the other side, there are components that would get a big jump in the severity level when access right is present. From attack standpoint, this will let external actors (hackers) have access to all records comparing to access of one record in the database components. Fig. 11 shows the risk factors of each component with or without admin access rights. We observe from Fig.11 that 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 security attack on this component is difficult. In the previous section 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).We take into the account the values of x1,x2,…x8 are not the same. We consider 9 cases. In each case we estimate the probabilities of these three components with the assumption that VI=.1. Fig. 12 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 through 9: we assume one initial node has a probability of 1 and the rest nodes have zero probabilities.

The following conclusions can be inferred:
The risk factor of customer agent component is high in case 8 and case 9 when the probability of initial nodes x8 is one or x9 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. The risk factor of customer interface component is high in case 2 through case 7 when the probability of initial nodes x1 is one or x2=1,…,x6=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.
The risk factor of book server component is high in case 8 and case 9 when the probability of initial nodes x8 is one or x9 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.

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.

Fig. 13 shows that the distributions of risk factors for each of the three components, customer agent, customer interface, and bookserver database considering 9 cases. 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.

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. 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.

In the previous section, we introduced the design security patterns that could be used on the architectural level and will add more protection layers to the components. These patters will help lower the security risk factors of components. 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

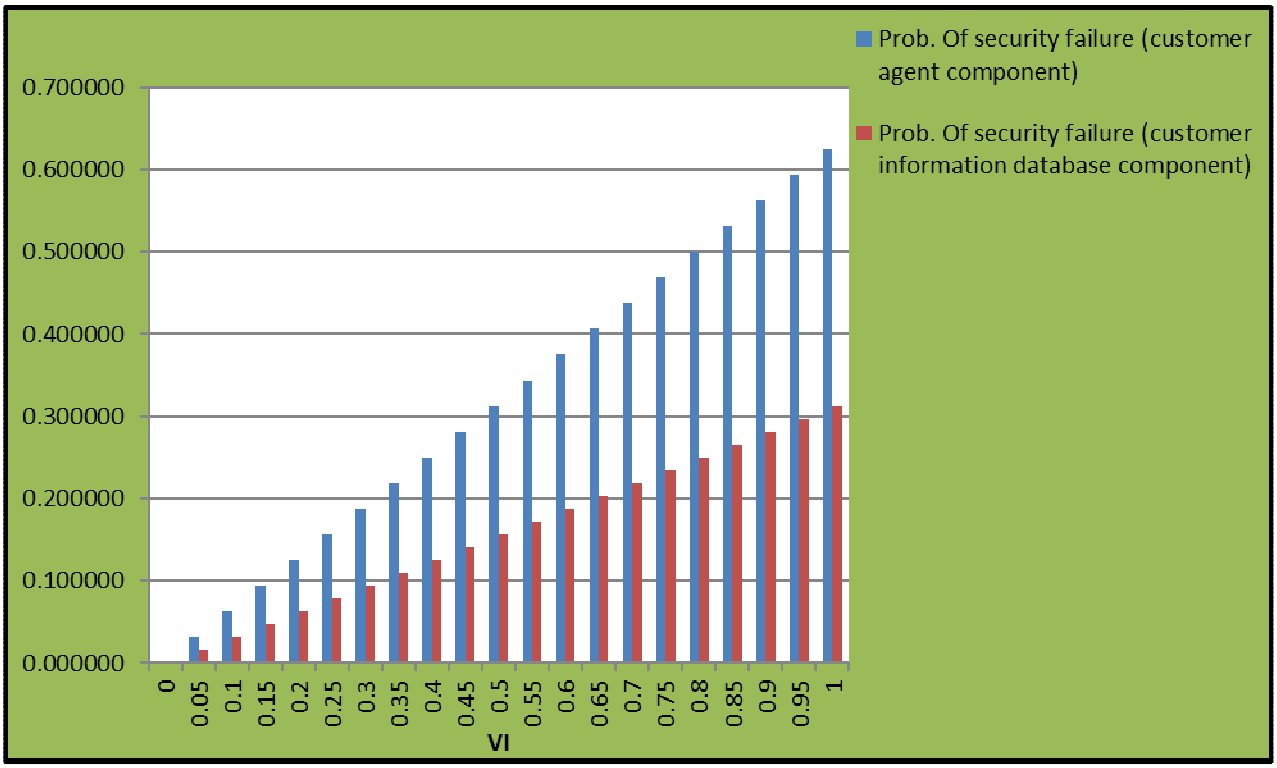

Fig. 9 Prob. of security failures of customer agent and customer information database components when VI changes

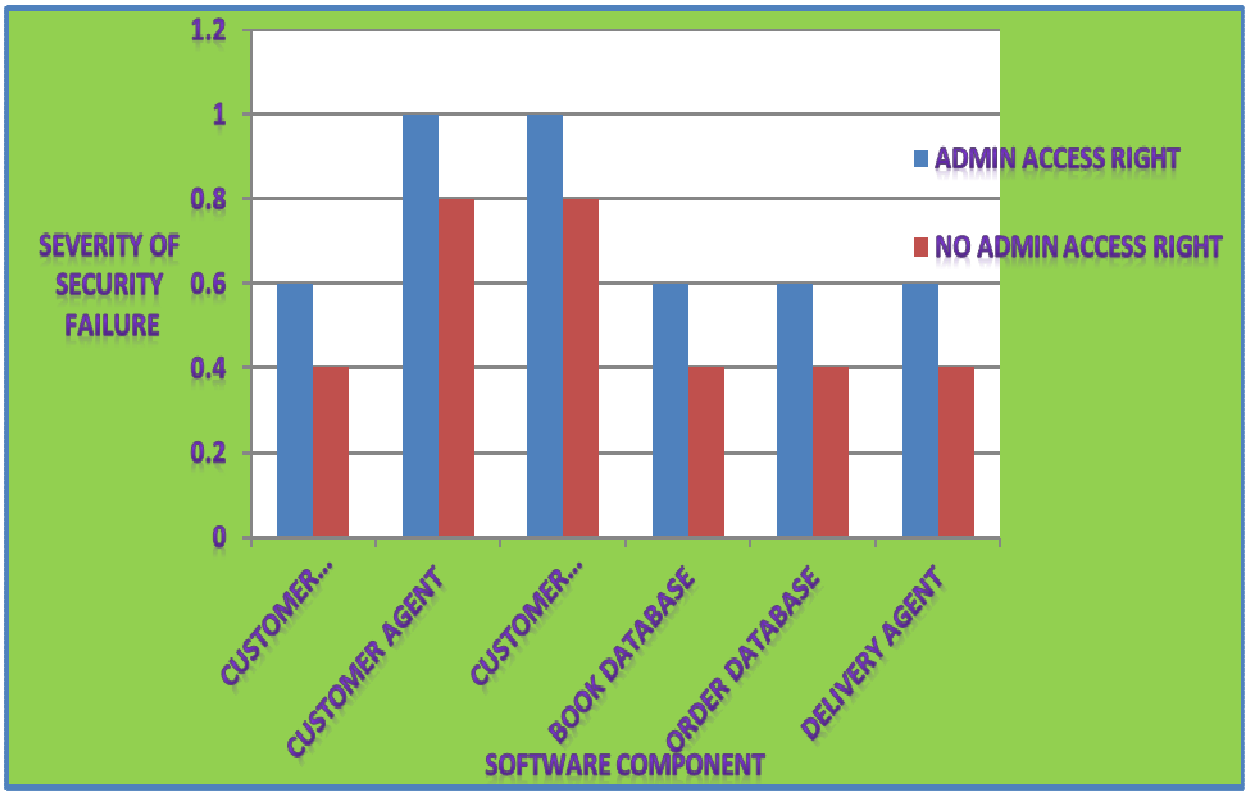

Fig. 10 Severities of security failures of ecommerce components with/without admin rights

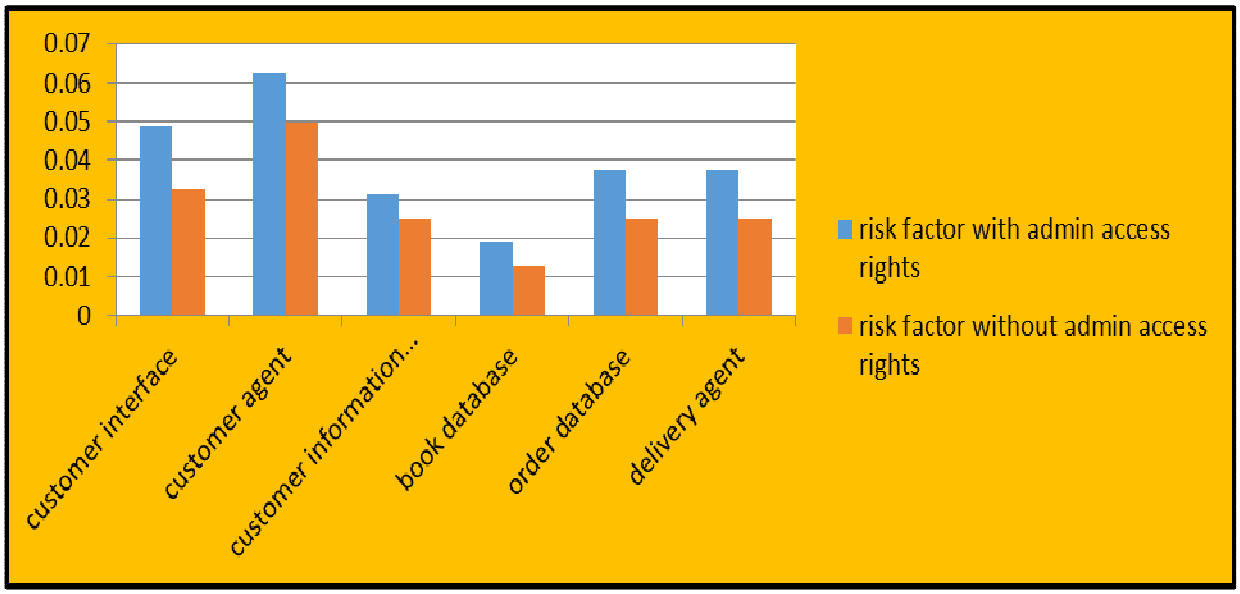

Fig. 11 Risk factors of software components in ecommerce example with admin rights, without admin rights, VI=.1

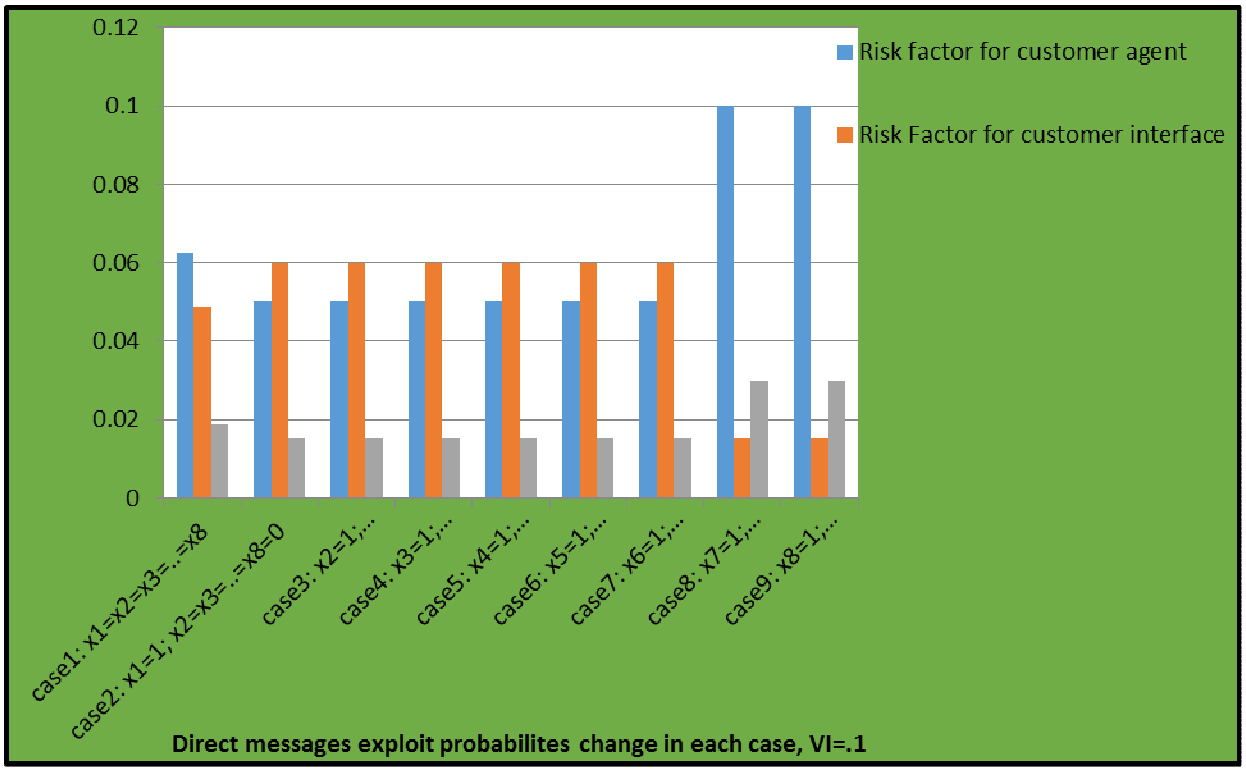

Fig. 12Security Risk Factors for components (customer agent, customer interface, book database) 9 cases, VI==.1

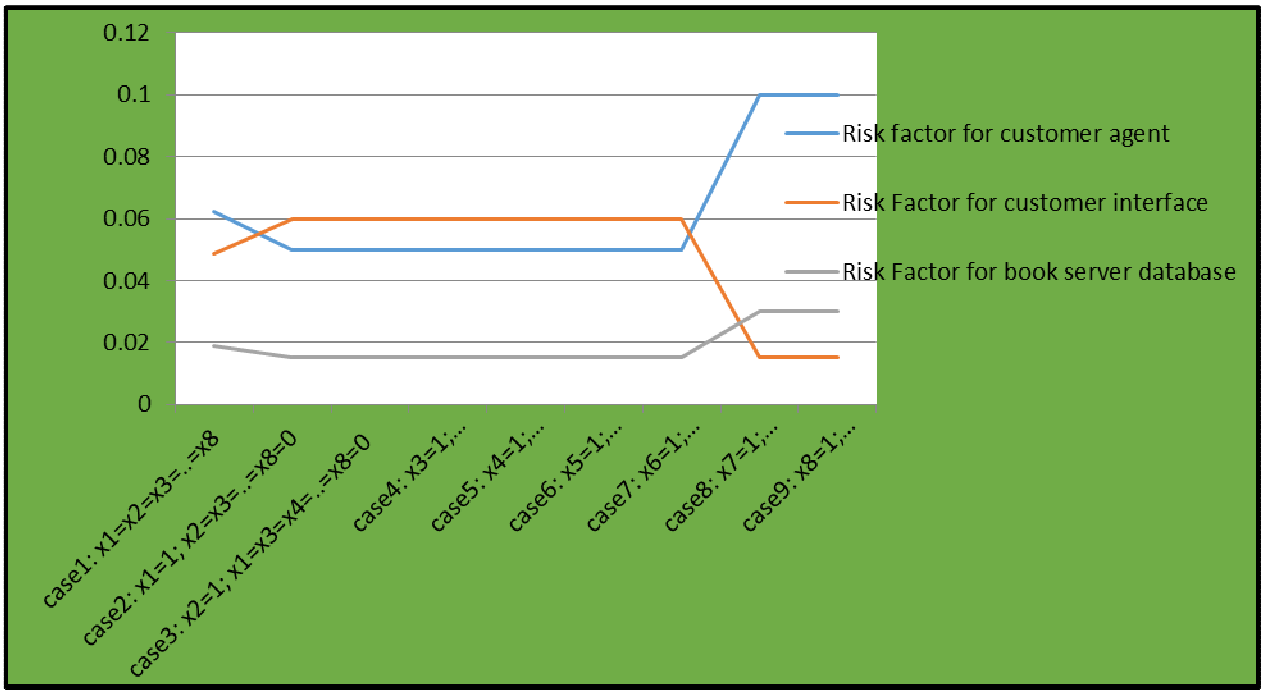

Fig. 13 distributions of risk factors of 3 components (customer agent, customer interface, book database)

5. Conclusions

In this paper, we have proposed a methodology for security risk assessment based on UML specifications, Attack Graph development, database sensitivity, reachability Matrix, and access rights. Furthermore, our estimation is performed at the early phases of software lifecycle. Thus, early security attacks detection will help developers focus on high security risk elements, scenarios and use cases. We conducted two studies to validate our proposed methodologies based on the design security patterns and sensitivity analysis methods. Our assessment is not only beneficial to developers, but also to software companies, industries, governments, and consumers especially most systems are built to be used through internet. Our work can be extended in more than one direction. First, an important extension is to automate the security risk assessment of any system .Second, extend our methodology to assess the security risk in the clouds and hosting systems especially the present and future is growing significantly in these two directions.

Acknowledgments

This research work was funded in part by Qatar National Research Fund (QNRF) under the National Priorities Research Program (NPRP) Grant No.: 7 - 662 - 2 - 247

References

[1] A. Hecker, "On System Security Metrics and the Definition Approaches" IEEEThe Second International Conference on Emerging Security Information, Systems and Technologies, August 2008, p 412-419

[2] B. Blakley, C. Heath, and Members of the Open Group Security Forum, "Security Design Patterns: Open Group Technical Guide", 2004.

[3] C.Feng, and S. Jin-Shu, "A Flexible Approach to Measuring Network Security Using Attack Graphs,"IEEE International Symposium on Electronic Commerce and Security, Computer Society, 2008, p. 426-43

[4] G. McGraw, "Software Security."IEEE Journals, 2004

[5] G. McGraw, "Software Security Building Security In." Addison-Wesel, 2006.

[6] J.A. Wang, H. Wang, M. Guo, M. Xia, "Security metrics for software systems." ACM Proceedings of the 47th Annual Southeast Regional Conference.Article 47, 2009

[7] J.B. Bowles, W. Hanczaryk, "Threat Effects Analysis: Applying FMEA to Model Computer System Threats" IEEE Conference Reliability and Maintainability Symposium, 2008. RAMS 2008. Annual, Jan 2008, p 463 – 468,

[8] J. O. Aagedal, F. D. Braber, T. Dimitrakos, B. A. Gran, D. Raptis, K. Stolen, "Model-based risk assessment to improve enterprise security."Proceedings Sixth International Enterprise Distributed Object Computing, 2002.p. 51-62.

[9] 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.

[10] M. Howard, J. Pincus, and J.M. Wing. "Measuring Relative Attack Surfaces." Workshop on Advanced Developments in Software and Systems Security, 2003.

[11] N.Davis,, et al., "Processes for Producing Secure Software Summary of US National Cybersecurity Summit Subgroup Report." IEEE Security and Privacy, 2004. 2(3), p. 18-25.

[12] P.K.Manadhata, and J.M. Wing, "An Attack Surface Metric." IEEE Transactions on Software Engineering, 2010

[13] R.S. Pressman, "Software Engineering: A Practitioner's Approach." McGraw-Hill Science, 2001 5th ed.

[14] S. L. Pfleeger, and C. P. Pfleeger, "Security in Computing,"4th edition, Prentice Hall, Upper Saddle River, NJ, 2007.

[15] V.Sharma, and K. Trivedi, "Architecture based analysis of performance, reliability and security of software systems."ACM 5th international workshop on Software and performance (WOSP'05), 2005, p. 217- 227.

Fadi HajSaid Bachelor of Electrical Engineering from Damascus University in 1997, Master of Computer Engineering from Stevens Institute of Technology New Jersey in 2000, and Ph.D. of Computer Engineering from West Virginia University in 2011. He has been working in Microsoft Corporation (New York) since 2001 as Technical Account Manager and consultant. His research area is security risk assessment of software architecture.
Biographies should be limited to one paragraph consisting of the following: sequentially ordered list of degrees, including years achieved; sequentially ordered places of employ concluding with current employment; association with any official journals or conferences; major professional and/or academic achievements, i.e., best paper awards, research grants, etc.; any publication information (number of papers and titles of books published); current research interests; association with any professional associations. Do not specify email address here.

Yousef Hassouneh holds a PhD degree in computation from University of Manchester, UK. He has a profound experience in Human Computer Interaction, he designed a collaboration framework and groupware tool to enable Requirements Engineering team collaboration. He is an assistant professor at the computer science department and teaches courses in Software Engineering, Internet programming and programming languages. His research interest are in Software Architecture, Virtual software engineering teams, Software risk assessment and metrics, mining software repositories. He participated in several EU funded projects

Hany H. Ammar BSEE, BSPhysics, MSEE, and PhD EE, is a Professor of Computer Engineering in the Lane Computer Science and Electrical Engineering department at West Virginia University. He has published over 170 articles in prestigious international journals and conference proceedings. Dr. Ammar is currently the Editor in Chief of the Communications of the Arab Computer Society On-Line Magazine. He is serving and has served as the Lead Principal Investigator in the projects funded by the Qatar National Research Fund under the National Priorities Research Program. In

2010 he was awarded a Fulbright Specialist Scholar Award in Information Technology funded by the US State Department - Bureau of Education and Cultural Affairs. He has been the Principal Investigator on a number of research projects on Software Risk Assessment and Software Architecture Metrics funded by NASA and NSF, and projects on Automated Identification Systems funded by NIJ and NSF. He has been teaching in the areas of Software Engineering and Computer Architecture since 1987. In 2004, he co-authored a book entitled Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, Addison-Wesley. In 2006, he co-authored a book entitled Software Engineering: Technical, Organizational and Economic Aspects, an Arabic Textbook.