Security Risk Assessment of Software Architecture
Fadi HajSaid¹, Yousef Hassouneh², Hany Ammar¹,
¹West Virginia University, USA; ²Birzeit University, Palestine
fhajsaid@yahoo.com, yhassouneh@birzeit.edu, hany.ammar@mail.wvu.edu

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 security risk 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. An e-commerce case study is investigated as an example.

Index Terms — Attack Graph, Probability of security failure, Security risk factor, Severity of security failure, Software Architecture.

I. 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 [11]. 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 [3], 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 [8]. 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 [9]. 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 [4].

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. The rest of the paper is organized as follows. We briefly discuss the related work in Section II. In Section III, we introduce our methodologies to estimate the security risk factors. At the end, we conclude with a discussion of future work in Section IV.

II. 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 [7]. 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 [2]. 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 [6] [1] [5]. 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.

III. 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.

A. 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 [2] can be represented as a Tuple

$$AG = (C_0, T, C_d, E, C_g)$$

(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.

$$Oprob(\tau) = Eprob(\tau).Cprob(c_1) ... Cprob(c_k)$$

(2)

$$Cprob(c) = \sum_{j=1}^{p} Oprob(\tau_j)$$

(3)

Where $c$ is an intermediate component or goal component. Where $\tau_1, ..., \tau_p$ 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 $C_0 = \{c_1, c_2, ..., c_8\}$, 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.

$$Cprob(c_1) = ... = Cprob(c_8) = VI$$

(4)

$VI$ [13] denotes the Vulnerability Index. (VI) is defined as the number of successful attacks on
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.
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 c T’). T’ = \{t_1, t_2, \ldots, t_g\}. T’ represents the direct messages exchanged between the external actors and the system. If we assume all these external messages have equally likely distribution,

\[ Eprob(t_1) = \cdots = Eprob(t_g) = \frac{1}{8} \] (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’. 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 messages send(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_{ia3}) 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 [10]. 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.

Schedule 1: 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 equations (2&3) to estimate the probability of security failure of the goal node.

\[ Cprob(c_{di}) = Oprob(t_{ia1}) + \cdots + Oprob(t_{ia}) = VI \times \left( \frac{6}{8} \right) \]

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:

\[ Eprob(t_{ia1}) = \frac{1}{Z} \] (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.

\[
\begin{bmatrix}
0 & 2 & 0 & 0 & 0 & 0 \\
2 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ MA = \]

\[ Eprob(t_{ia1}) = Eprob(t_{ia3}) = \frac{1}{4} \]

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

\[ Oprob(t_{ia1}) = Eprob(t_{ia1}) \times Cprob(c_{ci}) = \frac{1}{4} \times \left( \frac{6}{8} \right) \]

\[ Oprob(t_{ia3}) = Eprob(t_{ia3}) \times Cprob(c_{ci}) = \frac{1}{4} \times \left( \frac{6}{8} \right) \]

\[ Cprob(g) = Oprob(t_{ia}) + Oprob(t_{ia}) + Oprob(t_{ia1}) + Oprob(t_{ia3}) \]

\[ Cprob(g) = \frac{5}{8} \times VI \]

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.

B. 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 [12] 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-database 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).

### C. 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:

\[
rf(c) = \text{Prob}(c) \times \text{Severity}(c) \tag{7}
\]

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. 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>
<thead>
<tr>
<th>Component</th>
<th>Probability</th>
<th>Severity</th>
<th>Risk Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Interface</td>
<td>VI(6.5/8)</td>
<td>.4</td>
<td>.325VI</td>
</tr>
<tr>
<td>Customer Agent</td>
<td>VI(5/8)</td>
<td>1</td>
<td>.625VI</td>
</tr>
<tr>
<td>Customer information database</td>
<td>VI(3/16)</td>
<td>1</td>
<td>.18VI</td>
</tr>
<tr>
<td>Books database</td>
<td>VI(3/16)</td>
<td>.4</td>
<td>.07VI</td>
</tr>
<tr>
<td>Delivery agent</td>
<td>VI(5/8)</td>
<td>.4</td>
<td>.25VI</td>
</tr>
<tr>
<td>Orders database</td>
<td>VI(5/8)</td>
<td>.4</td>
<td>.25VI</td>
</tr>
</tbody>
</table>

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

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.
IV. CONCLUSION

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. 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 the 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 future is growing significantly in these two directions.

ACKNOWLEDGEMENT

This research work was funded by Qatar National Research Fund (QNRF) under the National Priorities Research Program (NPRP) Grant No: 09-1205-2-470.

REFERENCES


Fig. 4. Attack graph of customer agent component.