Security Risk Analysis of Software Architecture Based on AHP

Abdulaziz Alkussayer  
Department of Computer & Information Programs  
Institute of Public Administration  
Riyadh, Saudi Arabia  
kussayerA@ipa.edu.sa

William H. Allen  
Department of Computer Science  
Florida Institute of Technology  
Melbourne, FL, USA  
wallen@fit.edu

Abstract—Many organizations and companies around the world are currently facing major security risks that threaten assets and valuable information system resources. Numerous security concerns are caused by the lack of sufficient and effective software security risk evaluation processes. Regardless of the technique used in security attacks, which change rapidly, many of these threats can be avoided. This paper presents an effective model for discovering software security risks at an early stage of the software development cycle and reports on the ongoing development of a security trust metrics of software architecture.

Keywords—Software Security, Security Architecture Evaluation, Security Risk Analysis

I. INTRODUCTION

Evaluating the security of a given software under development at an early stage (i.e. design stage) to reveal security risks is crucially important; not only to help ensure that the stakeholder's security objectives have been met, but also-and more importantly-to reveal hidden security design issues that are commonly overlooked. The work proposed in this paper is motivated by the lack of coverage of security concerns caused by relying on a traditional risk analysis approach. The authors have developed a robust security evaluation technique to assess the security support of a software architecture during the design phase called the Security Evaluation Framework (SEF) [1]. The SEF framework yields promising results for evaluating the security of a software architecture.

However, the SEF framework also poses some limitations, especially during the risk-based assessment stage of the evaluation. For instance, the security evaluation of scenarios assesses the risk at the architecture level but neglects the effects of those scenarios on structural components. This, in turn, results in a biased risk assessment that conceals the risk of structural components.

Moreover, the dependability on a shallow severity mapping function proposed by many conventional risk models is sufficiently incapable of providing a deep analysis of the security risk exposure of the system. However, the security risk analysis concerns tracking complex, and often poorly defined, combinations of risks. In fact, security experts [2] acknowledge that "regardless of the risk analysis method used to evaluate risk, it must consider the following properties: 1) The more vulnerabilities exist in an asset and the systems that enable access to it, the greater the risk. 2) The more severe the vulnerabilities, the greater the risk" [2].

In this paper, we propose a new software security risk evaluation model that poses a dramatic improvement while adhering to the security concerns mentioned above. This enhanced technique depends on a hybrid two-stage process that combines a conventional risk analysis practice with the Analytical Hierarchy Process (AHP) [3]. Also, the risk severity mapping function is improved by following the color-coded threat level system implemented by the Homeland Security Advisory System [4].

In addition, an experimental case study is implemented to validate the risk model's efficiency at discovering security threats in the architecture design of a web-based application. The analysis of this case study and the implementation of the security controls are also discussed. Finally, a discussion on the viability to propagate an overall architecture security quantitative indicator as a part of the risk analysis model is also discussed.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the SEF framework [1] and the AHP [3]. Section 3 describes the implemented case study. Section 4 presents the architecture security evaluation of the implemented case study utilizing the proposed risk analysis model. Section 5 presents a discussion on the ongoing development of a security trust indicator of software architecture. Finally, Section 6 contains our conclusion and a discussion of the future work.

II. RISK AND SOFTWARE SECURITY

Nowadays, the use of risk analysis and risk management in software security engineering is a common practice. The idea is to list system assets, vulnerabilities, and threats so that risk can be determined. In this sense, a risk is the probability of a threat compromising a vulnerability to acquire an asset [5]. In the literature, many risk methodologies have been proposed to assess security risks from different angles. Some of them represent the risk as a financial loss and hence assess the
Return on Investment (ROI). Others represent the risk as a qualitative analysis technique that relates risk to a qualitative factor [6], [7], [8], [9], [10].

A. The SEF Framework

The inspiration of the SEF framework [1] comes from recognition of the critical need for assessing the security of a software system through its architectural design to reveal the underlying compliance of the architecture to the stakeholder’s security needs. It strengthens the security of a software architecture by incorporating three distinct factors seamlessly into a cohesive framework.

The SEF framework encompasses six steps as depicted in Figure 1.

B. The Analytical Hierarchy Process (AHP)

The AHP is a decision making technique developed by Thomas Saaty [3]. This method is used in a multifactor decision problem where a choice must be made among several alternatives. The decision problem is formulated in a hierarchical structure.

Typically, the hierarchy is constructed so that elements at the same level are of the same order of magnitude and must be related to some or all elements in the next higher level. Once the hierarchy is constructed, the prioritization process starts to determine the relative importance of the elements in each level of the hierarchy. The AHP uses a pair-wise comparison matrix to calculate the relative objective weight of factors. Elements in each level are pair-wise compared with respect to their importance in making the decision under consideration [3].

A pair-wise comparison matrix is then constructed to determine the relative importance between factors (decision elements). An arbitrary entry in row $i$ and column $j$ of the matrix, labeled $s_{ij}$, indicates how much more (or less) the importance for factor $i$ is than that for factor $j$ [3]. The importance is measured on an integer scale from 1 to 9, with each number having the interpretation shown in Table I.

<table>
<thead>
<tr>
<th>Intensity of Value</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Factor $i$ and $j$ are of equal importance</td>
</tr>
<tr>
<td>3</td>
<td>Factor $i$ has a slightly higher importance than $j$</td>
</tr>
<tr>
<td>5</td>
<td>Factor $i$ has a strongly higher importance than $j$</td>
</tr>
<tr>
<td>7</td>
<td>Factor $i$ has a very strongly higher importance than $j$</td>
</tr>
<tr>
<td>9</td>
<td>Factor $i$ has an absolutely higher importance than $j$</td>
</tr>
<tr>
<td>2, 4, 6, 8</td>
<td>These are intermediate scales between two adjacent judgment</td>
</tr>
<tr>
<td>Reciprocal</td>
<td>If Factor $i$ has a lower importance than $j$</td>
</tr>
</tbody>
</table>

### III. A PRACTICAL CASE STUDY

In this paper, the implemented case study is used as a running example to illustrate the improved security risk evaluation model. The case started by implementing the vulnerable web-based application (i.e. Flowershop system [18]). Security vulnerabilities such as SQL Injection, Cross Site Scripting (XSS), and Weak Session Management were deliberately implemented in the non-secure version of the system. An Apache HTTP server and a MySqL server were configured as a web server and database server, respectively. Then, the existence of these security threats was confirmed by penetrating the system using an open source automated security testing tool (i.e. Paros [19]).

The SEF framework [1] was also used to evaluate the architecture of the non-secure version of the system (i.e. Version-A). In this context, the improved risk evaluation technique was used and incorporated into the SEF framework. This improvement measurably overcame the limitations posed...
Finally, the security controls (e.g. patterns) were gradually implemented as architectural transformations after every evaluation round. The cyclic fashion of this process refers to steps five and six in the SEF framework [1].

Next, an analytical security risks evaluation of this case study was carried out.

IV. THE SECURITY RISK EVALUATION

The first step in the SEF model framework [1] is to determine the goal of the evaluation. In this case, the goal is to evaluate the underlying security support of the Flowershop system. Note that the first-round evaluation process aims to assess the security of the deliberately non-secure version of the system (i.e. Version-A). The second and the third steps are to generate security scenarios and create the security profile. The generation of the security scenarios followed the formal definition of scenarios [1], where

\[
S = \{s_1, s_2, ..., s_n\}
\]

is the set of security scenarios such that \( s_i \) corresponds to the security scenario for the \( i^{th} \) threat where \( n \) is the total number of identified threats. And, \( s_i = (r_i, t_i, p_i) \) is a tuple where \( r_i \in R, t_i \in T, \) and \( p_i \subset P \) such that \( R, T, \) and \( P \) represents security requirements, threats, and security patterns respectively.

However, the threats incorporated in each scenario tuple must be realistically defined based on the threats facing the system under investigation and not based on speculation. Thus, Paros [19] has been used to penetrate Version-A of the Flowershop system.

First, the web site was crawled to traverse all pages looking for vulnerabilities. Next, Paros [19] was set to intercept HTTP requests and process them to compromise these vulnerabilities. As a result, ten imminent security threats were discovered and thus ten scenarios were generated. These security scenarios comprise the security profile.

In fact, the scenario-based evaluation process depends largely on the correctness of the architecture description of the system under investigation and hence great effort has been given to correctly grasp the underlying structure of the Flowershop system. Thus, the system has been functionally decomposed into subsystem/component constituents. The rationale behind such decomposition is to synthesize the functional security risk responsibility higher in the structural hierarchy (i.e. component ->subsystem->scenario). Table II illustrates the functional decomposition of the Flowershop system.

The standard UML static modeling notation is used as the architecture description language in this case. Figure 3 depicts the architecture of the Version-A of the system. The subsystems are denoted by UML package notation and the components are denoted by UML component notation. Now that the architecture is properly described, the evaluation step of the SEF framework can begin. Although the original SEF framework [1] considers three distinct evaluation techniques (i.e. pattern-based, design decision-based, and risk-based evaluation), this paper focuses only on the risk-based evaluation. However, this does not at all mean that the other two evaluation techniques are less important during security risk analysis.

A. The Two-Stage Risk Model

The limitations of conventional risk models, described earlier, are enhanced by properly utilizing the AHP [3]. The hierarchy structure for the Flowershop system is depicted in Figure 2.
framework is to promote simple and effective evaluation steps that can be reproduced easily in a cyclic fashion. However, the components factor plays a critically influential role in evaluating the security risk of the scenarios, as discussed earlier. Thus, a hybrid approach is developed to combine a conventional risk model (e.g. RRM [20]) and the AHP [3]. The technique is explained in two stages:

**Stage 1: Calculate the Elementary Risk**

The components factor is incorporated into the conventional risk equation. The formal definition of the elementary risk equation then follows:

The risk $R_{s_i}$ for the scenario $s_i$ follows the standard risk equation [20].

$$R_{s_i} = S_{s_i} \times O_{s_i}$$

The severity $S_{s_i}$ of a scenario $s_i$ in the Full Scenario Table FST [1] is derived by averaging the technical impact of corresponding security objectives ($j$) given that the threat in the scenario is realized. The formalization of this calculation is given by:

$$S_{s_i} = \frac{I(C_{s_j}) + I(I_{s_j}) + I(A_{s_j}) + I(AC_{s_j})}{\sum_j}$$

Similarly, the vulnerability factor $VF_{s_i}$ of the $s_i$ is estimated by averaging the corresponding vulnerability factors for that scenario. The formalization of this estimation is given by:

$$VF_{s_i} = \frac{EOD_{s_i} + EOE_{s_i} + Pub_{s_i}}{\sum_j}$$

However, the occurrence of the $s_i$ scenario is calculated with respect to the components factor effect as:

$$O_{s_i} = VF_{s_i} \times LPR_{s_i}$$

$$LPR_{s_i} = 1 - Min(\alpha_{s_i,j}) \times \frac{C_{ts_i}}{C_{ts_i}}$$

Where,

- $\alpha_{s_i,j}$ the pattern’s resistance factor,
- $C_{ts_i}$ the total number of components affected by the $s_i$ scenario,
- $C_{ts_i}$ the number of components that has security control implemented, and
- $LPR_{s_i}$ the lack of pattern resistance for the $s_i$ scenario.

For example, the risk calculation for $S_2$ is as follows:

$$R_{s_i} = S_{s_i} \times O_{s_i}$$

$$S_{s_2} = \frac{I(C_{s_2}) + I(I_{s_2}) + I(A_{s_2}) + I(AC_{s_2})}{\sum_j}$$

$$VF_{s_2} = \frac{EOD_{s_2} + EOE_{s_2} + Pub_{s_2}}{\sum_j}$$

$$O_{s_2} = VF_{s_2} \times LPR_{s_2}$$

$$LPR_{s_2} = 1 - Min(\alpha_{s_2,j}) \times \frac{C_{ts_2}}{C_{ts_2}}$$
Stage 2: Risk Analysis Using AHP

The goal is to assign weight for the relative importance of each scenario to the overall system’s risk, considering the elementary risk (stage-1) of individual scenarios. The hierarchy of factors and sub-factors that affect the goal is depicted in Figure 2. However, the components have already been included in the elementary risk evaluation and hence only scenarios pair-wise are evaluated. During the evaluation of the relative importance of each scenario, three aspects are considered:

- The severity of the scenario based on the elementary risk value.
- The number of components affected.
- The existence of patterns to alleviate the vulnerability.

The AHP pair-wise comparison between scenarios is then applied. Because the FST contains ten scenarios as analyzed in the elementary risk evaluation step earlier, the method produces a judgment matrix with 100(10 x 10) cells. Now that the judgment matrix is filled out, the data are averaged over normalized columns, in order to estimate the eigenvector of the matrix, which represents the criterion (i.e. importance) distribution [3]. The Eigenvector of the relative importance of scenarios is depicted in Table III.

The Eigenvector score of each scenario is the percentage that the scenario adds to the scenario’s total importance. In this case, the scenario $S_2$ composes almost 35% of the scenario’s total risk. This result appears to be logically rationalized, considering the severity of the threat posed by $S_2$ (i.e. SQL Injection) and the large number of components vulnerability influenced by this scenario.

However, AHP provides the ability to test for consistency as one of the method’s strengths [3]. The AHP consistency is based on the idea of cardinal transitivity. For example, if scenario $S_x$ is considered to be two times more important than scenario $S_y$, and $S_y$ is considered to be four times more important than scenario $S_z$, then perfect cardinal consistency would imply that $S_z$ be considered eight times more important than $S_x$. Hence, if $S_x$ was judged to be less important than $S_z$, it implies that a judgmental error exists and the matrix is inconsistent [3].

According to Saaty [3] Eigenvalue is obtained from the summation of products between each element of Eigenvector and the sum of columns of the judgment matrix. The Consistency Index (CI) then follows:

$$CI = \frac{\lambda_{\text{max}} - n}{n-1}$$

Where

- $\lambda_{\text{max}}$ the largest Eigenvalue.
- $n$ the size of judgment matrix.

A consistency ratio (CR) is calculated by dividing the Consistency Index (CI) for the set of judgments by the Index for the corresponding Random Matrix (RI). The RI is the average value of the CI, if the entries in the pair-wise comparison matrix were chosen at random. Table IV shows some of the standard RI values [3]. Because the number of scenarios is 10, the RI is 1.51.

<table>
<thead>
<tr>
<th>$n$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RI$</td>
<td>0</td>
<td>0.58</td>
<td>0.90</td>
<td>1.12</td>
<td>1.24</td>
<td>1.32</td>
<td>1.41</td>
<td>1.45</td>
<td>1.51</td>
</tr>
</tbody>
</table>

Saaty [3] suggests that if the consistency ratio exceeds 0.1 the set of judgments may be too inconsistent to be reliable. As
a result, the original assessment of the pair-wise comparison matrix seems to be consistent, and the consistency ratio computed (0.076) supports our observations. Figure 4 depicts a plot chart that summarizes the Version A security scenario’s relative importance considering the risk components.

\[
O_{s_1} = VF_{s_1} \times LPR_{s_1}
\]
\[
VF_{s_2} = \frac{EOD_{s_2} + EOE_{s_2} + Pub_{s_2}}{3} = \frac{10 + 8 + 10}{3} = 9.3
\]
\[
LPR_{s_1} = 1 - \left(\min(\alpha_{s_1}) \times \frac{C_{s_1}}{\sum \alpha_{s_1}}\right)
\]
\[
LPR_{s_2} = 1 - \left[0.8 \times \frac{4}{3}\right] = 0.2
\]
\[
O_{s_2} = VF_{s_2} \times LPR_{s_2} = 9.3 \times 0.2 = 1.86
\]

Thus, the final risk of the scenario \(S_2\) is
\[
S_2 = S_{s_2} \times O_{s_2} = 8.25 \times 1.86 = 15.4 \text{ (Guarded)}.
\]

Clearly three scenarios (\(S_5, S_6, S_8\)) have kept the same risk values. This is due to the fact that no patterns were specified. Thus, secure coding practice is followed to alleviate their risks. Nonetheless, the scenarios’ risk values remain the same. This is understandable as the SEF framework is intended to assess the security risk of a software architecture during the design stage and propagate unrelieved risk values to later stages in the development cycle.

The plot chart in Figure 5 summarizes the security risk exposure of the Flowershop system after implementing the security controls. Although the architecture of Version B reveals an acceptable security risk, a final penetration test is conducted using Paros to confirm the correct implementation of the security controls.

\[
R_{s_1} = S_{s_1} \times O_{s_1}
\]
\[
S_{s_2} = \frac{1}{\sum_1} (C_{s_2} + I(A_{s_2}) + I(A_{s_2}) + I(ACC_{s_2}))
\]
\[
S_{s_2} = \frac{10 + 10 + 5 + 8}{4} = 8.25
\]

The Version B architecture design includes four instances of the Intercepting Validator security pattern implemented at the four corresponding components; hence, the \(C_{s_2} = 4\).

Next, the second evaluation round starts to evaluate the Version B architecture security. The elementary risk is determined using the improved calculation formula. For example, the new risk calculation for \(S_2\) is as follows:

\[
R_{s_2} = S_{s_2} \times O_{s_2}
\]

\[
S_{s_2} = \frac{10 + 10 + 5 + 8}{4} = 8.25
\]

The Version B architecture design includes four instances of the Intercepting Validator security pattern implemented at the four corresponding components; hence, the \(C_{s_2} = 4\).

Thus, the final risk of the scenario \(S_2\) is
\[
S_2 = S_{s_2} \times O_{s_2} = 8.25 \times 1.86 = 15.4 \text{ (Guarded)}.
\]

Clearly three scenarios (\(S_5, S_6, S_8\)) have kept the same risk values. This is due to the fact that no patterns were specified. Thus, secure coding practice is followed to alleviate their risks. Nonetheless, the scenarios’ risk values remain the same. This is understandable as the SEF framework is intended to assess the security risk of a software architecture during the design stage and propagate unrelieved risk values to later stages in the development cycle.

The plot chart in Figure 5 summarizes the security risk exposure of the Flowershop system after implementing the security controls. Although the architecture of Version B reveals an acceptable security risk, a final penetration test is conducted using Paros to confirm the correct implementation of the security controls.

V. LACK OF SECURITY TRUST (LST)

The aim is to equip software engineers with a metric that can quantify the security provided by a software system under development during the design phase. Much of the research in security metrics has resulted in metrics for later development stages (coding, testing, and operation), several of which are investigated in Jaquith’s seminal book "Security Metrics: replacing fear, uncertainty, and doubt" [22]. However, evidence from other security research efforts has questioned the claimed benefit and the validity of these
metrics. Recently, Verendel [23] surveyed the previous work on quantitative representation and analysis of security for the last three decades. He concluded "The result from considering a large part of the proposed methods is that quantified security is a weak hypothesis: for most cases, it is unknown if the methods are valid or not in representing operational security" [23].

Acknowledging the fact that security is a moving target and the threat landscape changes rapidly suggests that a valid, accurate, quantitative security metric seems to be far away on the horizon – especially for systems in the early development stages. However, the best practice in the software security industry suggests the use of the 'good enough' notion to promote security indicators rather than accurate metrics [6], [8]. In this context, the improved risk evaluation model presented earlier in this paper is used to introduce a security indicator. The indicator represents the inverse of a security property of the system. Thus, it measures 'how insecure' the system is rather than 'how secure' it is, a problem that is not yet solved. Thus, we propose that an alternative method for measuring the security, or in this case the relative insecurity, of an architecture is possible - and can best be achieved using security scenarios.

Thus, we present the Lack of Security Trust (LST) indicator which we define as the Root Mean Square (RMS) of the scenario’s risk values. The RMS is a statistical measure of the magnitude of varying quantity [24]. The formal definition then follows:

$$LST = \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2}$$

Where

- $N$ the total number of scenarios.
- $x_i$ the risk value for $i^{th}$ scenario.

For example, the LST indicators for the Flowershop system Version A is:

$$LST_A = \sqrt{\frac{1}{10} \sum_{i=1}^{10} 0x_i^2} = 48.60$$

This results in a (High) overall risk of the architecture. While the LST for Version B of the system is:

$$LST_B = \sqrt{\frac{1}{10} \sum_{i=1}^{10} 0x_i^2} = 15.69$$

And that results in a (Guarded) overall risk.
VI. CONCLUSION AND FUTURE WORK

This paper presents a systematic security risk evaluation model based on a hybrid approach. The model introduces the structural components factor and propagates their risk effects on security to a higher level (i.e., architecture model of the system). Furthermore, we present the security risk evaluation model in this paper, as a part of the SEF framework. However, the model can be used exclusively in security risk analysis situations where the development of scenarios is not foreseen. It provides a practical step-wise method that can be applied frequently in an environment where requirements are changing rapidly.

Moreover, the Flowershop system case study was implemented as a controlled experiment to observe the validity of the enhanced risk model. The experiment’s security controls were effectively implemented gradually after every evaluation round as architectural transformations. This suggests that our technique can easily be integrated with incremental SDLC methodologies. Finally, our observation revealed a security trust indicator that can be evaluated as early as the design phase in the software development cycle.

However, as with all new approaches in software security engineering, further validation is required. In future work, we plan to track observations from industrial case studies to confirm the practicality of the risk model and the viability of the security trust indicator. Another topic of on-going research is the realization of the security components capability in a component-based architecture.

REFERENCES