A Scenario-Based Framework for the Security Evaluation of Software Architecture

Abdulaziz Alkussayer
Department of Computer Science
Florida Institute of Technology
Melbourne, FL, USA
alkussaa@my.fit.edu

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

Abstract— **Software security has become a crucial component of software systems in today's market. However, software security development is still a maturing process. In this paper, we present an approach for assessing software architecture to determine how well it can satisfy intended security requirements. It is important to be able to assess the security of software under development at an early stage (e.g., the design stage). By doing so we are not only reducing the probability that flaws will be introduced and ensuring that stakeholder requirements have been met, but also focusing on a stage where changes will cost just a fraction of what they would cost in later stages (e.g. implementation). This paper reports on the ongoing development of a systematic security evaluation framework that aids in assessing the level of security supported by a given architecture and provides the ability to qualitatively compare multiple architectures with respect to their security support.**

Keywords—**Software Engineering, Secure Software, Software Architecture, Scenario-Based Evaluation**

I. Introduction

It is clear that designing software with security in mind will produce a more secure architectural design and eventually more secure software, yet it is still unclear how to evaluate and conduct this intuitive process. It is equally clear that software architecture is best used to model and outline the system•s structures and inter-components and that a good architectural design is one that performs certain tasks (i.e. functionalities) and exhibits certain properties (e.g. security) [1].

Other quality attributes of software systems, such as modi"-ability, portability, maintainability and usability have matured more rapidly than security. Many of these can be assessed during the design phase using qualitative or quantitative software architecture analysis methods. For example, SAAM [2] and ATAM [3] are well-known scenario-based architecture analysis methods that have been proposed by SEI-CMU and many other scenario-based evaluation approaches have been proposed as well [4], [5], [6], [7]. These techniques have proven useful for assessing quality attributes such as modi"ability, extensibility and "exibility [8].

Intuitively, the security properties of a software architecture could be assessed in a similar fashion. However, dealing with security, because of its nature, is different than dealing with any other quality attributes. Hence, our assessment model incorporates security patterns as a critical factor in the scenario•s assessment. In fact, assessing the architecture of a system to determine the support it provides for security is extremely 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.

In this paper, we present a scenario-based framework for evaluating the security of a software architecture. First, we survey several different architectural evaluation techniques with respect to their applicability to assessing the security properties of a system. Then, we describe the proposed scenario-based framework and how it can be used to evaluate the security of software architecture. Finally, we present a case study to demonstrate the application of our evaluation technique and to illustrate the bene"ts of its use. The rest of this paper is organized as follows. Section 2 provides a brief overview of architectural evaluation techniques and their applicability to security. Section 3 describes the proposed evaluation framework. Section 4 presents a practical case study that illustrates the use of our approach to effectively assess security of software architecture. Section 5 discusses related work, while section 6 contains our conclusions and a discussion of future work.

II. Architectural Evaluation of Security

There are four types of software architectural assessment [9]: mathematical modeling, simulation-based, scenario-based and experience-based assessment. A brief overview of these assessment techniques is presented with respect to their ability to assess security properties of the system architecture.

A. Mathematical Modeling Assessment

A mathematical model is developed that describes the intended structure of the architecture. Once designed, the model is applied to the software architecture to assess whether or not a given architecture meets the prede"ned quality requirement under investigation. This type of assessment may be suitable for •operationalŽ quality attributes where the requirement de"nes the exact intended behavior of the system (e.g. performance). However, it is rather dif"cult (if not impossible - for the time being) to develop a mathematical model to accurately

978-1-4244-5540-9/10/$26.00 ©2010 IEEE

predict the level of security support needed by an architecture. This is due to the following reasons:

1) The nature of software security introduces an inherent limitation. A quality requirement is usually measured by the presence of its positives. The security requirement however is measured by the absence of its negatives. Hence, security requirements pose a dif‘cult challenge.
2) Security metrics do exist for later stages, such as coding, testing and deployment. Examples include the number of ”aws discovered during code review, the number of failed test cases during penetration testing, the number of vulnerabilities exposed after deployment, etc. However, these metrics do not exist before the code is written and hence cannot be employed by a mathematical model at an earlier stage.

B. Simulation-based Evaluation

Simulation of the architecture is usually done using an executable model of the software architecture. This often includes models of the core components of the system. These components are then composed to simulate an overall architecture of the system. This technique allows a better understanding and analysis of different quality attributes, however simulating attacks to the system on a subset of the overall architecture is an incomplete form of penetration test that does not explore the full range of potential attacks. Thus, it is of limited benefit when evaluating the security properties of the system.

C. Scenario-based Evaluation

To assess a particular quality attribute, a set of scenarios is developed that conveys the actual meaning of the requirement. These scenarios are assigned a relative weight and organized into a profile. The profile can then be used to evaluate the software architecture. Scenarios fall into a wide range of categories; for example, *change* scenarios for maintainability and *hazard* scenarios for safety [9]. Similarly, security scenarios describe the permitted, and hence not permitted, architectural security exposure of the system. The scenario-based assessment directly depends on the scenario profile defined for the quality attribute to be assessed. The effectiveness of the technique relies on the accuracy of the representative scenarios comprising the profile [9], [10].

Throughout the literature, a number of scenario-based software architecture evaluation methods are discussed. These approaches are best summarized by Ionita et al. [8]. Many of these approaches are refinements of the Software Architecture Analysis Method (SAAM) [2] and Architecture Tradeoff Analysis Method (ATAM) [3], both developed by SEI-CMU. SAAM uses stakeholder input to derive requirements and scenarios to illustrate system qualities, both of which are then used to indicate how well an architecture addresses a set of system qualities. The SAAM method inspired the development of ATAM. Using these methods, the software architecture is analyzed with respect to its quality attributes (e.g. maintainability, reliability) and tradeoffs between attributes are identified.

Bengtsson et al. [6] present a modifiability analysis method for software architecture which begins by identifying the goal of the analysis. This goal can then be used to predict maintenance costs, perform a risk assessment, or to compare a pair of software architecture candidates. The second step describes the software architecture. The third step elects scenarios relevant to the goal and the fourth step evaluates the scenarios' effect. The final step is to reach conclusions based on an analysis of the results.

Dolan [7] presented a software architecture evaluation technique called the Family Architecture Assessment Method (FAAM), intended for information-system families. The method focuses on extensibility and interoperability. Extensibility is ”the ability of the system to accommodate the addition of new functionality” [7], while interoperability is ”the ability of two or more systems or components to exchange information and use the information that has been exchanged” [7]. FAAM consists of six steps [7]: 1) define assessment, 2) prepare system-quality requirements, 3) prepare software architecture, 4) review and refine artifacts, 5) assess architecture conformance, 6) reporting results and proposals.

Folmer et al. [4] present a scenario-based evaluation method called SAULTA. The method focuses on assessing software architecture usability. SAULTA encompasses the following steps: 1) define the assessment goal, 2) develop change scenarios, 3) create change profiles, 4) describe the architecture, 5) evaluate the scenarios, 6) interpret the results. In the SAULTA method [4] three different types of scenario evaluation techniques are discussed to evaluate usability scenarios; namely: pattern based, design decision based, and use case map based.

D. Experience-based Assessment

In this technique, skilled security architects use their experience and intuition to logically justify certain design decisions. Although this method is more subjective than the others, it can be very helpful if combined with a scenario-based method.

III. The Security Evaluation Framework (SEF)

The inspiration for our evaluation framework 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. The proposed technique strengthens the security of a software architecture by incorporating three distinct factors seamlessly into a cohesive framework.

The initial component of the process is a scenario-based architecture review. An architecture review is an effective way of ensuring design quality and addressing architectural concerns. The key objectives of conducting an architecture review are to evaluate an architecture's ability to deliver a system that fulfills the stakeholders' quality requirements and to identify potential risks [1], [10]. The use of scenarios to evaluate software architecture is maturing process and has proven to be a successful practice [1], [5], [9], [10]. For instance, the ATAM [3] method has gained significant attention from researchers and practitioners because it both provides an assessment of

quality attributes and explores the interactions and interdependencies of those quality attributes, highlighting trade-off mechanisms and opportunities between quality attributes [9]. The scenario-based security evaluation proposed in this paper could be incorporated into an ATAM that represents a broader framework for architectural assessment. Moreover, the stepwise evaluation process of security scenarios in the proposed technique is similar to earlier approaches [4], [6], [7].

The second factor is the incorporation of a risk analysis model. Babar et al. [5] surveyed the state of practice in evaluating software architecture and concluded that 88% of the survey participants conducted architecture review with the goal of identifying potential risks. Thus, the proposed framework augments the use of a well-known risk analysis model for application security, OWASP's Risk Rating Methodology [11].

Finally, the incorporation of security patterns improves the quality of security components in the architecture. Security patterns have proven effective for dealing with security problems in a software system [12], [13], [14], [15]. Therefore, the proposed framework incorporates security patterns not only during the risk analysis, but also as a core component of each of the security scenarios that comprises the security profile.

The proposed evaluation framework encompasses six steps as depicted in Fig. 1. A detailed explanation of these steps follows.

A. Define the Evaluation's Goal

The first step is to determine the goal of the evaluation and declare the expected outcomes of this assessment. In general, the assessment process is usually driven by one of these three goals [9], [10].

- *Quantitative assessment*: To predict the level of security supported by the architecture. Such a prediction is, in many cases, intended to be an indicator rather than an absolute quantitative measure of security (which may not even be feasible).
- *Qualitative assessment*: To compare two or more possible candidate architectures to decide which has the best support for security.
- *Trade-off assessment*: To discover the right granularity of the security support with respect to other quality attributes.

B. Generate Security Scenarios

As we described in Section II-C, security scenarios are appropriate for describing the required security support for a software architecture. In order to generate a coherent security scenario, an architect needs to closely consider the concurrent security engineering activities; namely, threat modeling and security requirements.

Swiderski et al. [16] suggest that threat profiles developed during threat modeling ought to be used during the architecture and coding artifacts. Threats can be well defined and classified according to the STRIDE model [16]. However, it is not clear how an architect would make use of the threat profile as an assessment instrument. Moreover, there is little documented

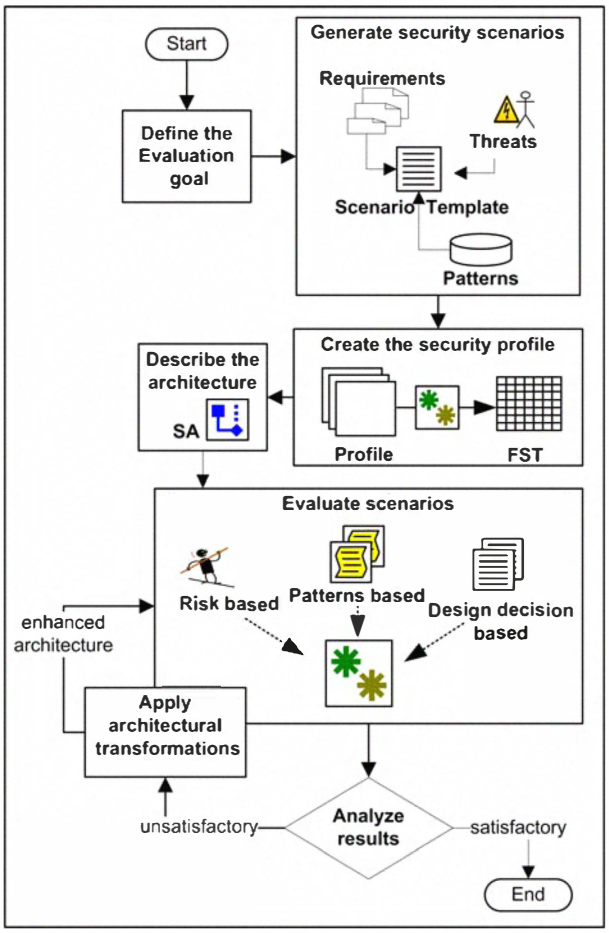

Fig. 1. The Security Evaluation Framework

research regarding the software architectural behavior in cases where a threat is not mitigated and becomes a vulnerability that can be exploited by an attacker.

On the other hand, security requirements by themselves are not enough to construct a security scenario. Despite the fact that abuse cases can be used as a way of explaining security requirements, they are insufficient to construct a security scenario; this is because the term 'abuse case' in its basic sense describes an abnormal (mostly malicious) usage of system functionalities (abuse cases). In other words, it describes a threat to the system in the context of a functional requirement. Even in cases where security requirements have been defined using a sound methodology such as [17], [18], which provides a better way to define and measure security requirements, depending on the security requirements alone to assess the security of software architecture is problematic.

In this context, Rozanski et al. [1] define a quality-based scenario that can be used to capture concerns about different quality attributes. This quality-based scenario is general and hence can be found useful for most of the quality attributes. However, the fact that it doesn't address the potential threats to the system makes it less useful in describing security

concerns. Thus, we believe a useful security scenario is one that increases the architect's attention and explicitly addresses the security threats to the system.

Next, we propose a more systematic approach to generate a coherent security scenario. The formal definition of the security scenario is shown below.

$S = \{s_1, s_2, ...s_n\}$ 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.

$s_i = (r_i, t_i, p_i)$ is a tuple where $r_i \in R$, $t_i \in T$, and $p_i \subset P$ such that R, T, and P represent security requirements, threats, and security patterns respectively.

Fig. 2 shows the structure of a scenario template. The template consists of five elements: requirement, threat, precondition, behavior, and patterns.

- ***requirement*** is the specific security requirement [17] that describes the required security property of the system. It is important to be able to trace a security scenario back to its constituent requirement because in the end the design decision depends on the stakeholders' understanding of different trade-offs. Traceability of requirements aids in reaching that understanding.
- ***threat*** is a description of the threat to the system in which it explicitly (directly) violates the requirement or implicitly (indirectly) leads to a violation. The threat must be imported from the threat profile [16] and not artificially created.
- ***precondition*** is the description of possible system constraint(s) that cause the security scenario to occur.
- ***behavior*** is the expected behavior of the system if a particular scenario is encountered. Note that it is important to focus exclusively on security and avoid describing the system behavior from other quality-attribute perspectives (e.g. performance).
- ***patterns*** that ought to be incorporated in order to mitigate the corresponding threat and safeguard the required behavior. Selecting the right pattern is indeed not an easy task, especially when different pattern catalogs list similar patterns with different names [12], [14]. However, one can make use of well organized repositories such as [19], [20].

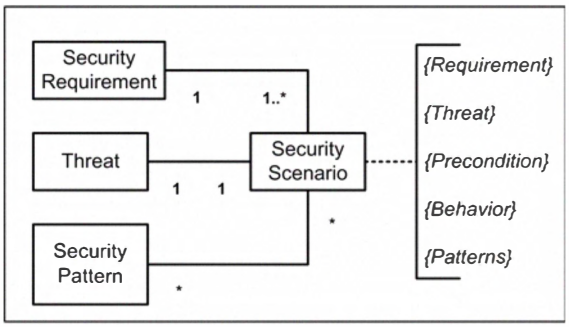

Fig. 2. Structure of a Scenario Template

Clearly, in some cases patterns will be referenced in multiple scenarios (e.g. `Authentication` pattern [13]). In other cases, multiple patterns may be identified in a single scenario because patterns complement each other and "no pattern is an island" [13]. For example, consider a scenario where the requirement states "the product shall retain a protected journal of all transactions". In this scenario the selected patterns would normally include: `Audit Trail` [19] and `Secure Logger` [19] to satisfy the "protected" keyword. Also, if no specific pattern can be identified, then the mitigation strategy of the corresponding threat may be used during the assessment of this scenario.

Note that all of the identified scenarios jointly influence the creation of the security profile in the next step. Including all scenarios in the security profile enriches the design security resource and raises the awareness of security issues at an appropriate stage.

C. Create Security Profile

"The definition of profiles for the quality attributes considered most relevant for the software architectural design allows for concrete and precise description of the meaning of statements about quality attributes" [9]. Consequently, the security scenario profile comprises the identified security scenarios that are going to be used during the assessment analysis of the architecture. There are two factors which influence the creation of the scenario profile [9]: selection criteria and prioritization.

Selection Criteria

A complete selection is defined as one that includes all scenarios that can potentially occur, versus a selection that includes a representative subset of all possible scenarios. Although we believe it is important to include all identified scenarios into the profile, the notion of a complete selection – in the context of security – is vague. This is because security is a moving target and new threats are evolving every day, which in turn changes the threat landscape rapidly. Hence, a good method for selection of representative scenarios depends on the risk associated with each scenario. Next we describe the process of associating risk values with each of the scenarios in the profile. Note that the OWASP Risk Rating Methodology (RRM) [11] follows the standard risk model:

Risk = Likelihood * Impact

The RRM suggested two factors to estimate the risk likelihood (*threat agent factor* and *vulnerability factor*) and two factors to estimate the risk impact (*technical factor* and *business factor*). Each of these factors has a set of options, and each option has a rating from 0 to 9. However in this paper, the threat agent factor and the business impact factor have been excluded to eliminate unnecessary subjectivity and maintain simplicity. Thus, we estimate the likelihood based on the vulnerability factors and the impact based on pure

technical factors. Also, we use the simple numerical values (0-9) temporarily during the estimation process to simplify the analysis process. Later, those numerical values are replaced by the corresponding rating levels. The OWASP's RRM [11] likelihood and impact levels are:

0 to $< 3 \Rightarrow$ Low,
3 to $< 6 \Rightarrow$ Medium,
6 to $9 \Rightarrow$ High.

Using severity levels conveys a simpler ranking of the software risk than associating numerical values [21]. Nevertheless, one can extend the risk estimation to include the business impact factor or any other organizational dependent factors as the RRM is highly customizable and the framework is also adaptable to many other risk estimation methodologies.

Next, we describe the process of extracting scenarios from the security scenario profile using the Full Scenario Table (*FST*). The output is a visually simplified representation of the security scenario profile with a set of scenarios that are ready for the analysis stage. This process is performed by distilling all of the scenarios (S) developed in the previous stage into the *FST*. The *FST* consists of four columns: scenario number, threat, patterns, and security objectives. The security objectives column is then divided into sub-columns that represent the desire objectives under investigation. This study exclusively focuses on the following objects: Confidentiality (C), Integrity (I), Availability (A), and Accountability (Acc). Fig. 3 depicts an example of the *FST*.

Note that it is not unusual to generate some scenarios without associated pattern(s). This could be due to one or both of the following:

i) A lack of patterns for newly emerged threats.

ii) The threat is to be mitigated by following an established best practice.

In any case, it is important to include these scenarios in the *FST* to assure that the target architecture has addressed these threats and their countermeasures.

Scenario Prioritization

A prioritization of the profile aims to highlight the scenarios that pose more risks and need to be considered first. It is natural to prioritize scenarios in the security profile based on the risk identified earlier as a part of the *FST*.

Scenario #	Threat	patterns	Security Objectives				Risk
			IC	II	IA	IACC	
2	V2: SQL Injection	InterceptingValidator	8	9	5	6	Critical
5	V5: Failure to check privileges rights	AuthorizationEnforcer	7	9	5	5	High
10	V10: unencrypted data channel threat	Secure Pipe	9	8	3	4	High
18	V18: Fail to follow chain of trust in validation	AuthenticationEnforcer	8	6	4	5	Medium

Fig. 3. A Snapshot of FST

D. Describe the Software Architecture

The term *architecture description* refers to information about the architecture of the software under development. Although there are several ways to describe an architecture, such as design decision documents [1], [9] and AADL [22], the use of *UML* notations is prominent. The framework uses a *pattern-based* approach to generate scenarios and create the security profile and since most of the security patterns are documented using *UML* notations. Hence, we also use *UML* notations to describe the architecture in our framework.

Once the software architecture description is documented, the security profile is created and the *FST* is prepared, it is time to start the scenario evaluation process.

E. Evaluate The Scenarios

The process of evaluating the security of a software architecture is done by comparing the required security capabilities with the provided security capabilities [9]. Scenarios in the *FST* efficiently convey the required security capabilities of the system. Thus, the security scenarios profile created earlier is used during the evaluation process. For each scenario in the profile, the architecture is analyzed for its support of that scenario.

Folmer et al. [4] proposed three types of scenario evaluation techniques to evaluate usability scenarios of a software architecture; namely: *pattern*-based, *design decision*-based, and *use case map*-based. In contrast, our framework proposes three techniques to evaluate security scenarios. These techniques are: *pattern*-based, *risk*-based and *design decision*-based evaluation [4]. These techniques complement each other, as depicted in Fig. 1.

Pattern-based Evaluation

This requires examining each scenario from the security profile (*FST*) and heuristically evaluating the architecture using the list of identified patterns in that scenario. It is important to note that expert-based identification of any security pattern that influences security in the system is required. However, most security patterns are well structured using UML notations and this structure is usually documented in pattern templates in catalogs. Hence, the process of extracting the pattern from the architecture under investigation can be done by an architect with a lower level of security expertise. This process is repeated for all identified scenarios in the (*FST*) and the results of the analysis are documented.

Risk-based Evaluation

This process estimates and associates risk weights to every scenario (row) in the *FST*. This is done by estimating the impact and likelihood of each scenario. Although this risk estimation requires a certain level of security experience, a good knowledge of previous projects and the organization's platform helps greatly to improve the precision of the estimation. Additionally, OWASP's RRM [11] provides a guide to estimating the technical impact and vulnerability factors, as depicted in Tables I and II respectively.

TABLE I
TECHNICAL IMPACT FACTORS

Impact of Confidentiality (**IC**)	[2]	non-sensitive data disclosed
	[6]	sensitive data disclosed
	[7]	critical data disclosed
	[9]	all data disclosed
Impact of Integrity (**II**)	[1]	minimal corrupt data
	[6]	seriously corrupt data
	[7]	extensive corrupt data
	[9]	all data totally corrupt
Impact of Availability (**IA**)	[1]	minimal services interrupted
	[6]	serious services interrupted
	[7]	extensive services interrupted
	[9]	all services lost
Impact of Accountability (**IACC**)	[1]	fully traceable
	[7]	possibly traceable
	[9]	completely anonymous

TABLE II
VULNERABILITY FACTORS

Ease of Discovery (**EOD**)	[1]	practically impossible
	[3]	difficult
	[7]	easy
	[9]	automated tool available
Ease of Exploit (**EOE**)	[1]	theoretical
	[3]	difficult
	[5]	easy
	[9]	automated tool available
Publicity (**Pub**)	[1]	unknown
	[3]	hidden
	[7]	obvious
	[9]	public knowledge

The technical impact I_{s_i} of a scenario s_i in the *FST* is derived by averaging the impact of corresponding security objectives (j) given that the threat in the scenario is realized. The formalization of this calculation is given by:

$$I_{s_i} = \frac{I(C_{s_i})+I(I_{s_i})+I(A_{s_i})+I(ACC_{s_i})}{\sum^{j} 1}$$

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:

$$VF_{s_i} = \frac{EOD_{s_i}+EOE_{s_i}+Pub_{s_i}}{\sum^{j} 1}$$

If security patterns are identified in a scenario, their remediation effect $(\alpha_{s_{i_1}}, \alpha_{s_{i_2}}, .., \alpha_{s_{i_j}})$ is considered to relax the likelihood estimation of that scenario. However, patterns vary in their anticipated resistance to attacks and hence the Lack of Pattern Resistance LPR is also considered. If multiple patterns are identified in a single scenario s_i, then the LPR_{s_i} follows:

$$LPR_{s_i} = 1 - Min(\alpha_{s_ij})$$

The likelihood of the s_i scenario can then be calculated as:

$$L_{s_i} = VF_{s_i} * LPR_{s_i}$$

The risk R_{s_i} for the scenario s_i follows the standard risk equation [11].

$$R_{s_i} = L_{s_i} * I_{s_i}$$

As discussed, focusing on severity levels to conclude the risk of each scenario conveys a clearer meaning and draws greater attention than numerical values [21]. Thus, the use of RRM [11] severity levels is recommended for this framework. Fig. 4 depicts those RRM [11] severity levels.

Finally, results from the overall analysis of the architecture are summarized. This may reveal some general security indicators. For example, the number of supported security scenarios can be contrasted with the number of scenarios not supported, or the total number of available security patterns can be compared with the number of patterns required by the profile. Also, it is possible to synthesize risk values associated with each threat to the scenario template. As a result, an overall quantitative indicator of the evaluation can be determined. However, such a quantitative metric of the software design is outside the scope of this paper.

The lack of software architecture support to some scenarios may come from an earlier design decision that was a result of trade-off analysis between security and other quality attribute. Hence, we need to investigate the effects of design decisions on the security support of the architecture.

Design decision based evaluation

Analyzing the structural components of the system and their interrelationship based on patterns is very useful. However, the early design decisions of the initial architecture that influenced the current architecture structure are also important. Although these decisions are the input to the evaluation process, they are not always documented during the early design of the architecture [9]. If that is the case, these decisions could be retrieved by interviewing the system architect(s) and the security team.

Every design decision identified is used to evaluate the security support for each security scenario. For each scenario in the security profile we analyze the impact of the design decision and whether this has resulted in sufficient support for that scenario.

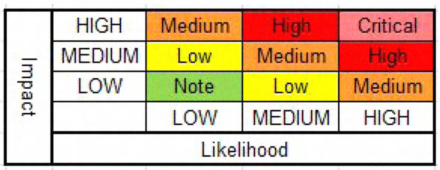

Fig. 4. Risk Severity Levels

F. Interpreting the results

Upon the completion of the scenarios evaluation process, the accumulated results are documented. The interpretation of this document influences the accept/reject decision of the security level provided by the architecture under investigation. If the security level supported is not accepted, then some architectural transformations [9] must be applied to overcome the lack of security support. It is important to note that our approach (i.e., creating scenarios based on patterns) eases the transformation process greatly. In contrast, some design decisions need to be taken in response to the results of the evaluation. These decisions have to be justifiable in case some scenarios are not supported and compromises need to be taken.

Next, we present a case study to illustrate the risk-based evaluation process using our framework.

IV. A Practical Case Study

For illustration purposes, we use the Distributed Architecture case study presented by OWASP-CLASP [23]. However, due to space limitations we will discuss only the *Customer's Participation* use case. This context represents a general form of most of today's online e-commerce systems from the customer's perspective.

The OWASP CLASP project has identified twenty-two specific threats (called vulnerabilities) [23]. These threats convey a simpler form of threat profiling for this case study. Hence, based on these identified threats and on the interaction steps in the use case, we have developed the corresponding security requirements for the target architecture to generate security scenarios as described in Section III-B. Next, we will walk through the framework to evaluate the security risks of the initial (non-secure) design of the architecture. Then, we propose improvements to the initial architecture by integrating security patterns during the architectural transformation stage. After that, the enhanced (secure) design is evaluated. Finally, the risk levels are compared and final results are discussed.

The first step of the framework is to determine the goal of the evaluation. In this case study, the goal is to evaluate the risk of the initial (non-secure) architecture design.

The second step is to create the security scenarios. Hence, a total of twenty-two security scenarios have been created for this example. Following the scenario template explained earlier, each scenario is linked to a specific security requirement, potential threat, and security pattern(s) that may be used to remediate that threat.

The third step in the framework is to create the security scenario profile. In this case, the *FST* distilled from the security scenario profile contains 22 different scenarios. The scenarios within the *FST* are then categorized into security objectives.

The fourth step is to describe the architecture using UML notation.

Now that the software architecture description is documented and the *FST* is prepared, it is time to start the scenario evaluation process.

During the risk-based evaluation, the individual risk of each scenario is estimated and associated with the scenario in the *FST*. Fig. 3 depicts a snapshot of the *FST*. While it is difficult – due to space limitations – to present our risk estimation for all the scenarios in the *FST*, the estimation of the scenario s_2 is presented next. Note that the assignment of technical impacts, vulnerability factors, and pattern resistance is done based on our experience. The threat in the s_2 is SQL Injection [24], one of the most common and effective attacks over the past decade [25]. Thus, the technical impact and likelihood of the attack are both fairly high.

$$I_{s_2} = \frac{I(C_{s_2})+I(I_{s_2})+I(A_{s_2})+I(ACC_{s_2})}{\sum_{1}^{j}}$$

$$I_{s_2} = \frac{8+9+5+6}{4} = 7 \textit{ (High)}$$

The initial (non-secure) design does not include any of the security patterns identified in the FST; hence, the Likelihood of s_2 equals its vulnerability factor.

$$L_{s_i} = VF_{s_i} * LPR_{s_i} = VF_{s_i} * 1$$

$$L_{s_i} = \frac{EOD_{s_2}+EOE_{s_2}+Pub_{s_2}}{3} = \frac{8+7+9}{3} = 8 \text{ (High)}$$

Thus, the final risk of the scenario s_2 is (Critical). Similarly, the estimation of all the scenarios in the *FST* result in :

- 3 Critical risks,
- 7 High risks,
- 7 Medium risks,
- 3 Low risk, and
- 2 Notes (very low risks).

Fig. 5 shows the severity distribution of the 22 scenarios and concludes the first round of the evaluation process. Note that a (*Note*) risk is the result of a scenario that has both a low impact and a low likelihood. Further investigation shows that the two *Note* risks discovered earlier correspond to minimal threats identified in s_{15} and s_{20}. The threats in both scenarios can be effectively remediated following secure coding practices [26].

The first round evaluation results revealed unsatisfactory security capability level of the initial architecture design. Therefore, architectural transformation is applied to the architecture to integrate six security patterns; namely:

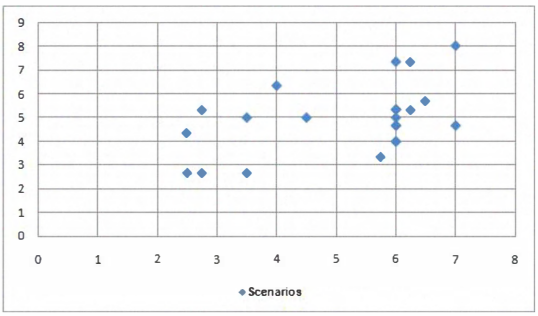

Fig. 5. Scenarios Distribution

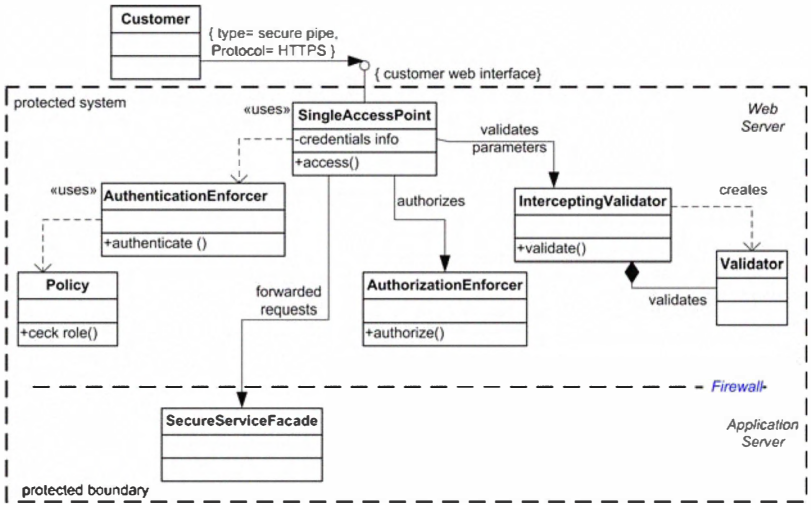

Fig. 6. Enhanced Architecture

`Intercepting Validator` [19], `Authorization Enforcer` [19], `Secure Service Facade` [15], `Authentication Enforcer` [19], `Single Access Point` [13], and `Secure Pipe` [19]. Fig. 6 depicts the enhanced architecture design.

Consequently, once the architectural transformation is completed, a second evaluation process is launched to verify that the enhanced version of the architecture yields a better risk exposure. The result of the second round evaluation are:

- 1 High risks,
- 15 Medium risks,
- 4 Low risk, and
- 2 Notes (very low risks).

The increase in Medium risks is because the RRM [11] severity rating maps a Low likelihood and a High impact into a Medium risk (see Fig. 4). However, a dramatic reduction has been observed in the likelihood of nine scenarios as depicted in Fig. 7. This risk reduction is a result of the weighted integration of patterns. Finally, the findings of the analysis along with the architect's recommendations are documented and communicated to the general body of stakeholders.

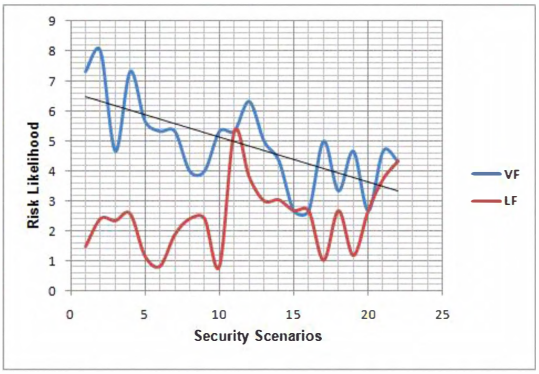

Fig. 7. Likelihood Reduction

V. Related Work

The work by Liu et al. [27] proposed a quantitative assessment of a software architecture's support of security during the design phase. They proposed the USIE model to analyze the security of software architecture and produce an overall quantitative measure of security. However, the USEI model [27] was developed for SOA-based systems and may not apply to other architectures.

Steel et al. [15] presented an application security assessment model as part of the Secure UP methodology. However, this assessment depends on checklists and hence it must be performed by an experienced security consultant or specialized security architect.

Another project, SQUARE [18] proposes a nine-step methodology that generates a final deliverable of categorized and prioritized security requirements. They propose a template used to gather the requirements along with the steps to operate and assess them. Also, they incorporate the methodology into different lifecycle models [28]. However, the focus of the assessment in this methodology is on the requirements themselves, rather than the software architecture design.

Halkidis et al. [29] proposed a methodology for quantifying the risk of a software system using security patterns. They proposed the use of fault trees and fuzzy risk analysis to calculate the risk for each category of STRIDE. Our framework complements this work by introducing the security scenarios to cohesively encapsulate the architectural security knowledge of the system and combine the risk-based evaluation with other architectural evaluation techniques.

Most of the existing studies, to our knowledge, have focused on using scenarios to evaluate the quality attributes of software architecture such as modifiability, portability, maintainability and usability [2], [3], [4], [8], [10] and few attempts have been made to evaluate the security property of software architecture.

VI. Conclusion and Future Work

This paper combines a scenario-based assessment with risk-based analysis to create a framework for software architecture security evaluation. We present a systematic process for generating a security scenario template that simplifies the assessment process. Furthermore, we illustrate the applicability of our framework utilizing a common case study that clearly demonstrates the benefits of evaluating the security of software architecture during the design phase.

Our approach yields two main contributions toward efforts to advance the evaluation of security in software systems. First, the pattern-based scenario templates uniquely generate concrete profiles due to the proven track record of security patterns in practice. Second, the combined evaluation method enables robust architectural stress points evaluation of security and other quality attributes of the system.

However, as with all new approaches to improve software security, further validation is required. In future work, we plan to implement an industrial case study and continue to work towards the full formalization and realization of the framework. Second, the framework lacks a quantitative indicator of the overall architecture security support as a part of the evaluation process. Another topic of on-going research is the realization of security components capability in a component-based architecture.

Acknowledgment

The authors would like to thank the Institute of Public Administration (IPA) in Saudi Arabia for their support of this work.

References

[1] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2005.

[2] R. Kazman, G. Abowd, and M. Webb, "SAAM: A method for analyzing the properties of software architectures," in *16th International Conference on Software Engineering*, 1994.

[3] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," tech. rep., CMU/SEI-2000-TR-004, 2000.

[4] E. Folmer, J. V. Gurp, and J. Bosch, "Scenario-based assessment of software architecture usability," in *Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, ICSE, Portland*, 2003.

[5] M. Babar and I. Gorton, "Software architecture review: The state of practice," *IEEE Computer Society*, vol. 42, no. 7, pp. 26–32, 2009.

[6] P. Bengtsson, N. Lassing, J. Bosch, and H. Vliet, "Analyzing software architectures for modifiability," tech. rep., (HK-R-RES00/11-SE), University of Karlskrona/Ronneby, 2000.

[7] T. Dolan, "Architecture assessment of information-system families: a practical perspective." Ph.D. thesis, Technische Universiteit Eindhoven, 2001.

[8] M. Ionita, D. Hammer, and H. Obbink, "Scenario based software architecture evaluation methods: An overview," in *SARA workshop at ICSE*, 2002.

[9] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. Addison-Wesley, 2000.

[10] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

[11] OWASP, "Risk rating methodology." http://www.owasp.org, 2009.

[12] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress In Informatics*, no. 5, pp. 35–47, 2008.

[13] M. Schumacher, E. Frenandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2006.

[14] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, "An analysis of the security patterns landscape," in *3rd International workshop on Software Engineering for Secure Systems*, 2007.

[15] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall Ptr, 2005.

[16] F. Swiderski and W. Snyder, *Threat Modeling*. Microsoft Press, 2004.

[17] C. Haley, J. Moffett, R. Laney, and B. Nuseibeh, "A framework for security requirements engineering," in *SESS'06*, 2006.

[18] N. R. Mead, E. Hough, and T. Stehney, "Security quality requirements engineering (SQUARE) methodology," tech. rep., CMU/SEI-2005-TR-009, 2005.

[19] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen, "An inventory of security patterns," tech. rep., Katholieke University Leuven, Department of Computer Science, 2006.

[20] M. Hafiz, P. Adamczyk, and R. Johnson, "Organizing security patterns," *IEEE Software*, vol. 24, no. 4, pp. 52–60, 2007.

[21] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.

[22] S. International, "SAE standards: Architecture analysis & design language (AADL)." http://www.sae.org, 2010.

[23] OWASP, "CLASP - Comprehensive, Lightweight Application Security Process." http://www.owasp.org, 2009.

[24] B. Potter and G. McGraw, "Software security testing," *Software Development*, vol. 2, no. 5, pp. 81–85, 2005.

[25] OWASP, "OWASP top ten." http://www.owasp.org, 2009.

[26] M. Howard and D. LeBlanc, *Writing Secure Code*. Microsoft Press, 2003.

[27] Y. Liu, I. Traore, and A. Hoole, "A service-oriented framework for quantitative security analysis of software architectures," in *IEEE Asia-Pacific Services Computing Conference*, 2008.

[28] N. R. Mead, V. Viswanathan, D. Padmanabhan, and A. Raveendran, "Incorporating security quality requirements engineering (SQUARE) into standard life-cycle models," tech. rep., CMU/SEI-2008-TN-006, 2008.

[29] S. T. Halkidis, N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Architectural risk analysis of software systems based on security patterns," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 129–142, 2008.