

Supporting Security Sensitive Architecture Design

Muhammad Ali Babar^{1,2}, Xiaowen Wang², and Ian Gorton¹

¹ Empirical Software Engineering, National ICT Australia, Australian Technology Park,
Alexandria, 1435 Sydney

{malibaba, ian.gorton@nicta.com.au}

² School of Computer Science and Engineering, University of New South Wales, Australia
{xiaowen@cse.unsw.edu.au}

Abstract. Security is an important quality attribute required in many software intensive systems. However, software development methodologies do not provide sufficient support to address security related issues. Furthermore, the majority of the software designers do not have adequate expertise in the security domain. Thus, security is often treated as an add-on to the designed architecture. Such ad-hoc practices to deal with security issues can result in a system that is vulnerable to different types of attacks. The security community has discovered several security sensitive design patterns, which can be used to compose a security sensitive architecture. However, there is little awareness about the relationship between security and software architecture. Our research has identified several security patterns along with the properties that can be achieved through those patterns. This paper presents those patterns and properties in a framework that can provide appropriate support to address security related issues during architecture processes.

1. Introduction

Quality is one of the most important issues in software development. It has been shown that software architecture (SA)¹ greatly influences the achievement of various quality attributes (such as security, performance, maintainability and usability) in a software intensive system [1]. That is why a number of formal and systematic techniques have been developed to ensure that the quality issues are addressed early in the software development lifecycle [2-5]. The principle objective of these techniques is to provide support in order to identify the required quality attributes, help design architectures to achieve the desired quality attributes, assess the potential of the designed architecture to deliver a system capable of fulfilling the required quality requirements, and identify potential risks [6].

¹ We use the definition of software architecture provided by Bass et al. [1] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*. 2 ed. 2003: Addison-Wesley.: “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.”

Security has become one of the most important quality attributes required in networked applications, which support business- and mission-critical processes. The reported cases of security breaches have been growing exponentially over the past decade [7]. Software intensive applications are usually composed of several heterogeneous platforms and components provided by different vendors which have different levels of concerns and considerations for security related issues.

Attackers apply highly sophisticated technologies to wage increasingly malicious attacks, which cause different types of damage to the stakeholders of the application. Examples are the unavailability of required medical records for urgent surgery because of a denial of service attack, or financial losses caused by security compromises on the server holding customers' credit card details. Poor quality software has been found the main reason of such vulnerabilities [8-10]. Despite the increasing importance of the security, many systems are designed and implemented without sufficient considerations for security related issues, which are often not dealt with until late in the development lifecycle [10].

Some security related measures may be easily incorporated into an implemented system. For example, adding a password protected login screen. However, making an application sufficiently secure by implementing appropriate and proven security solutions (e.g. security patterns) requires a much greater degree of modification, which may be prohibitively expensive at a late stage of system development. For instance, once security checks have been implemented in various components separately, it is difficult and expensive to introduce the "check point" pattern² to centralize security policies and algorithms to achieve the "maintainability" security attribute.

One of the major reasons for insufficient attention to security issues during early stages of software development is that many system designers do not have the required security expertise. Moreover, software designers and security engineers have different preferences [11, 12]. Security experts are primarily concerned with security, while software designers need to consider not only security but other quality attributes (e.g. performance, usability, and so on) as well. Furthermore, knowledge about techniques addressing security issues has not been captured and documented in a format that is easily accessible and understandable for designers, which makes engineering for security early in the design process difficult [10, 13].

Security engineering experts have developed and validated several known solutions to recurring security issues in the form of security patterns [10, 14]. These security patterns embody expert "wisdom" and raise security awareness among software practitioners. These patterns facilitate effective and efficient reasoning about security issues at a higher level of abstraction [13]. These security patterns prescribe the mechanics of addressing security issues during the software design phase. However, software engineers often do not realize that certain security patterns have architectural implications and cannot be easily applied after a certain stage of the development lifecycle because architectural decisions are harder and expensive to change [1].

The main contribution of the work reported in this paper is a framework for systematically considering and addressing those security issues that need architectural

Deleted: the

Deleted: several

² A list of architecturally sensitive security patterns used in this research and their brief description is provided in Section 3.3.

support. This framework presents a set of security attributes and properties along with the design patterns known to satisfy them in an integrated format. The proposed framework can help identify those security sensitive solutions, which cannot be cost-effectively retro-fitted into an implemented system - rather such solutions are only applicable during the architecture design stage. This framework is expected to be an important source of architecturally sensitive security knowledge to help understand the relationship between software architecture and security related quality attributes and to improve the architecture design and review processes.

Deleted: to

The remainder of this paper is organized as follows: In the next section, we discuss the concepts and issues that motivate our research. Section 3 presents a framework for supporting reasoning about security related issues during software architecture design and discusses the elements of this framework: security attributes, security properties, and security sensitive patterns and the usage of the framework. Conclusion and future work complete the paper.

Deleted: concludes

2. Theoretical Background and Motivation

A quality attribute is a non-functional requirement of a software system, such as reliability, modifiability, performance, and usability. According to [15], software quality is the degree to which the software possesses a desired combination of attributes. There are a number of classifications of quality attributes. In [16], McCall listed a number of classifications of quality attributes developed by software engineering researchers. A later classification of software quality is provided in [17]. Quality attributes of large software intensive systems are usually determined by the system's software architecture. It has been widely recognized that quality attributes of complex software intensive systems depend more on the overall architecture of such systems than on other factors such as platform and framework selection, language choice, detailed design decisions, algorithms and data structures [1].

Since software architecture plays a vital role in achieving system wide quality attributes, it is important to address the non-functional requirements during the software architecture process. The principle objective of addressing quality related issues at the architecture level is to identify any potential risks early, as it is quicker and less expensive to detect and fix design errors during the initial stages of the software development [1, 6].

A pattern is a known solution to a recurring problem in a particular context. Patterns provide a mechanism for documenting and reusing the design knowledge accumulated by experienced practitioners [18]. Software architectures of complex and large systems are usually developed using many different patterns. The architectures of such systems evolve by successively integrating several patterns that may be described at different levels of abstractions to deal with particular design problems [19]. One of the main goals of using patterns is to design an architecture with known quality attributes [20]. Each pattern either supports or inhibits one or more quality attributes.

Deleted: of

Security patterns capture the security expertise inherent in worked solutions to recurring problems. Security patterns are aimed at documenting the strengths and weak-

nesses of different approaches in a format easily understandable by those who are not security experts [10, 11]. According to Schumacher, “A security pattern describes a practical recurring security problem that arises in a specific security context and presents a well-proven generic scheme for a security solution” [13]. A security pattern’s documentation includes at least four sections: context, problem, solution and security pattern relations.

The security of a system can be characterized in four ways: identity management, transaction security, software security and information security [21]. Each of these focuses on different concerns and requirements. Digital identity management is aimed at defining the users’ capabilities according to their respective roles, tracking their actions, and verifying their identity. Transaction security ensures the secure communication between various parts of a system. Software security protects a system from viruses, software pirates, and certain types of hacking. Information security is responsible for protecting the data and information stored in the backend servers like databases and email servers.

In order to satisfy the requirements for each type of the security, there are different techniques such as encrypting data to prevent it being intercepted, altered, or hijacked. It is usually not necessary to devote equal attention to all categories of the security. A software designer should first assess the risks and come up with an appropriate security policy. The design and implementation strategies should reflect the priorities assigned to different categories of the security in the security policy [21, 22].

Another important issue regarding the design of a secure system is careful consideration of several attributes of security requirements, including authentication, authorization, integrity, confidentiality and auditability. How a smart attacker can compromise the security of a system can be demonstrated by a scenario, for example:

“An unauthorized user logs into an E-commerce application as an administrator after several attempts to guess an admin password. This unauthorized user browses through all the sensitive data such as transaction histories and credit card numbers. Moreover, this user creates a new account for themselves and gains the highest privilege.”

In this single scenario of unauthorized access, all the above-mentioned security aspects have been compromised. The hacker successfully guesses the administrator’s password, which violates authentication and authorization. The intruder reads sensitive data, which infringes confidentiality and integrity. Furthermore, several unsuccessful attempts at guessing the password go unnoticed, which breaks down auditability.

Apart from addressing the above-mentioned security aspects, a secure system must also satisfy other security attributes. For example, it should be easy to modify a security policy (maintainability), and should provide secure operations in all circumstances (reliability).

Architectural decisions made by taking security into consideration can sufficiently address most of the security breaches characterized by the above-mentioned scenario. For example, it is possible to modify or integrate the security policy late in the development and avoid huge code rewriting by using a suitable security sensitive architecture pattern like *check point* [14]. However, security expertise embodied by security patterns and its significance for software architecture is not well understood outside

Deleted: for

Deleted: their

the security community. Moreover, software designers usually do not have access to an integrated set of solutions to security issues that needs to be addressed at the software architecture level [10].

3. An Approach to Support Security Sensitive Architecture Design

We have been developing and assessing various techniques to capture and represent quality attribute sensitive architectural solutions in a format that can be an important source of knowledge during architecture processes [23-25]. Apart from our own work on discovering architecture knowledge from patterns, the idea of developing a framework that can help understand the relationship between software quality attributes and software architecture has been inspired by the work of [26, 27], on linking usability and software architecture.

To support the design and evaluation of security sensitive architectures, we decided to systematically analyze existing security patterns in order to identify those patterns which have architectural implications. We have rigorously studied them to understand the mechanics these patterns provide to achieve security. We have also found that there is no standard definition of the security quality attribute. From the security engineering literature, we identified the most common interpretations of security and grouped them in a small set of security attributes (see section 3.1). In order to establish a relationship between security and software architecture, we analyzed several security patterns (see section 3.3) to study their effect on the identified security attributes.

Since it is extremely hard to draw a direct relationship between quality attributes and software architecture [27], we decomposed the identified security attributes into more detailed elements and properties (see section 3.2), which can be considered a form of high level requirements as we can use general scenarios to characterize them [28]. We have put these identified relationships into a framework that relates the problem and solution domains for the security attribute. This framework consists of three layers: attributes, properties, and patterns. Before presenting the framework and discussing the ways in which it can support architecture design and evaluation activities, we briefly describe each of the layers and its elements.

Formatted: Indent: First line: 0.5 cm

Deleted: is expected to

3.1 Security Attributes

We mentioned in Section 2 that a quality attribute is a non-functional requirement of a software system. Bass et al. describe security as a measure of a system's ability to resist unauthorized usage without effecting the delivery of system's services to legitimate users [1]. Our literature review to identify architecturally sensitive security solutions revealed that the security quality attribute has also been described differently by different researchers and practitioners. For example, Singh et. al. [29] describe a security attribute as security concepts/mechanisms, Schneier [30] calls it security needs, and Proctor and Byrne [22] call it security objectives. However, a commonly

Deleted: have

found concept is that a security quality attribute is a precise and measurable aspect of a system's ability to resist unauthorized usage and different types of attempts to breach security. Having consulted the work of various security experts, we have identified a set of security attributes that need to be sufficiently satisfied by a secure system. We have selected only those attributes, which are most commonly used in the security domain. Following are the security attributes considered in this study.

Deleted: called attacks

Authentication: The identity of a system's clients (users or other systems) should be validated to thwart any unauthorized access.

Authorization: This attribute defines an entities' privileges to the different resources and services of a system and limits interactions with resources according to the assigned privileges.

Integrity: There should be a mechanism to protect the data from unauthorized modification while the data is stored in an organizational repository or being transferred on a network.

Confidentiality: A system should guarantee data and communication privacy from unauthorized access. Resource hiding is an important aspect of confidentiality.

Auditability: This means keeping a log of users' or other systems' interaction with a system. Auditability helps detect potential attacks, find out what happened after assaults, and gather evidence of abnormal activities.

Deleted: It

Maintainability: Facilitates introducing or modifying a security policy easily during later stages of the software development lifecycle.

Availability: Ensures that authorized users can access data and other resources without any obstruction or disturbance. If a disaster occurs, it ensures that a system recovers quickly and completely.

Reliability: This secures the operations of the system in the wake of failure or configuration errors. It also ensures the availability of the system even when a system is being attacked.

Deleted: It

3.2 Security Properties

Software designers apply several design principles and heuristics to achieve different quality attributes [31]. These principles and heuristics are called security properties, which provide a means to link appropriate patterns to a desired quality attribute [27]. In the architecture design stage, architects can consider these properties as requirements, for instance "*the system shall have a robust error handling mechanism*". Since security often conflicts with other quality attributes (such as performance and usability), architects need to decide how and at which levels these properties are implemented using appropriate security patterns.

For instance, when user inputs an incorrect password, a system should provide an informed error message and guide the user to input a correct password. This is a user friendly mechanism for supporting the user login process. But, how can a system distinguish between a user's mistakes and a malicious attack? Should the system execute a strict security strategy or implement a loose security strategy to improve the usability

Deleted: d again, which

Deleted: of

of the system? Architects can use architecturally sensitive security patterns to achieve the required properties with known affects on other quality attributes [14].

The security properties used in our research have been drawn from the work of different authors in the security domain. We have chosen the most commonly cited security properties in [10, 13, 32]. Following are the properties we considered so far.

Error management: A system should provide a robust error management mechanism to support error avoidance, error handling, fallback procedures and failure logging.

Simplicity: A system should encapsulate initialization check processes, ensure security policy and low-level security, manage permissions and share global information. Systems should also be easy to use and keep the user interface consistent.

Access Control: This property requires the system to support user identification, access verification, least privilege and privacy.

Defense in depth: This includes data verification, reduced exposure to attack, data protection, and communication and information protection.

3.3 Security Patterns

Following on the practice of documenting known solution to a recurring problem in a certain contexts in the form of design pattern, security engineering experts have also discovered and validated several known solutions to recurring security issues in the form of security patterns [10, 14]. A security pattern is supposed to document a particular recurring security problem that arises in a certain context and a well-proven solution to address that problem [13]. A catalogue of such patterns, also known as security pattern language, describes proven solutions to some common security problems. They summarize the strengths and weaknesses of different approaches to address known security issues and make security conscious design knowledge accessible to software designers. These patterns also document the knowledge of the tradeoffs that may need to be made in order to use a particular security pattern.

To select the patterns used in our research, we have drawn upon several sources of security engineering knowledge [10, 13, 14, 32], which explain different security patterns with examples and scenarios. The security engineering community has documented many more patterns than we consider here. For example, [14] explains 26 patterns and 3 mini-patterns. However, our research is concerned with only those security patterns that are architecturally sensitive. The list of architecturally sensitive security patterns described in this paper is not exhaustive. We plan to extend this list to identify more patterns, which support security related architectural decisions. These patterns can be organized into an architecturally sensitive security pattern language, which can explicate the relationships between the identified patterns. For example, Yoder and Barcalow [10] describe seven security patterns along with the relationship between them (see Fig. 1). We have found that all these seven patterns are architecturally sensitive and are usually used in tandem to achieve a particular set of security attributes.

Deleted: s

Deleted: ,

Deleted: patterns which have architectural implications and it is very hard and expensive to retro-fitted these patterns into an implemented system - rather these patterns are only applicable during the architecture design stage.

Deleted: And t

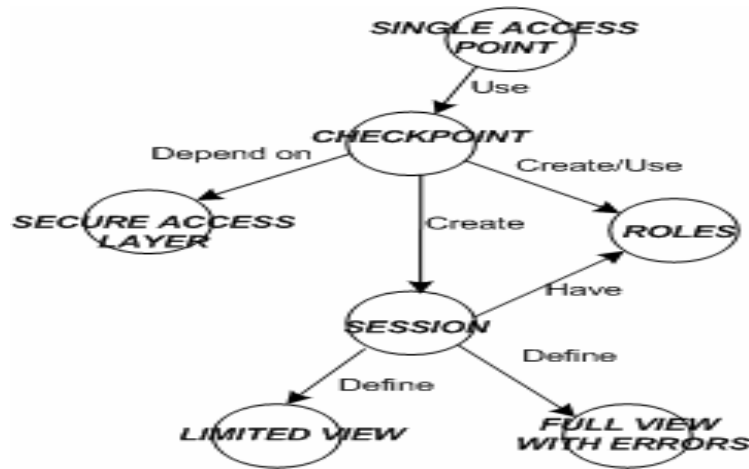


Fig. 1. Architecturally sensitive security pattern system [13].

In the following, we provide a brief description of each of the security patterns considered in this paper:

Single Access Point: This pattern ensures that there is only one entry point to a system. Anyone accessing the system is validated at the entry point. Having only one entry point makes it easy to perform the initial security checks by encapsulating the initialization process. The drawback of this pattern is inflexibility in terms of non-availability of multiple entry points.

Check Point: This pattern centralizes and enforces security policy and encapsulates the algorithm to put the security policy into operation. The algorithm can contain any number of security checks. This pattern can also be used to keep track of the failed security breaches, which helps take appropriate action if the failures are malicious activities.

Roles: This pattern enables the management of the privileges of a system's users at group level. It divides the relationship between a user and their privileges into two new relationships, user-role and role-privilege, which makes the user's privileges easily manageable. However, using the role pattern may result in some complication as implementing roles adds extra complexity for developers.

Session: This creates a session object to store and share variables throughout the system. It is an easy way to share global information among several components of a system. The usage of this pattern needs careful consideration for the propagation of session instance variables and appropriate structure to organize the values stored in the session.

Limited View: This pattern provides a dynamic GUI for different users based on their roles. The users can access content according to their privileges, which prevents unauthorized users from performing illegal operations. However, it can be difficult to realize and training materials for the application must be customized for each set of users.

Deleted: attempts of

Deleted: :

Deleted: It

Full View with Errors: Contrary to *Limited View*, this pattern provides the users of a system with a full view of the GUI. However, it ensures that users can only perform legal operations. In case of illegal operations, an error message is generated to notify the users. This pattern can be easier to implement, however, valid operation can be more difficult to identify as a user can perform any operation. Moreover, frequent error messages usually frustrate a user.

Secure Access Layer: This pattern provides an isolation layer to protect the data and services when integrated with other systems. It encapsulates lower-level security and isolates the developers from any changes made at other levels of security.

Authoritative Source of Data: This pattern is used to verify the validity of data and its origin. It prevents the system from using outdated and incorrect information and reduces the potential risk of processing and propagating fraudulent data.

Layered Security: This pattern is aimed at dividing a system's structure into several layers to improve the security of the system by securing all of the layers. One major drawback of using this pattern is increased complexity at the architecture level.

Formatted: Font: Italic

Deleted: the

3.4 Architecture Sensitive Security Framework

3.4.1 Development Process

In order to support security sensitive architectural decisions, the security attributes, properties and patterns identified in our research have been placed in a framework that we call security framework. The security framework is aimed at explicating and instigating systematic reasoning about the relationship between software architecture and security quality attributes. Figure 2 shows the current form of the security framework that captures and presents the relationships that exist among security sensitive attributes, properties, and patterns. To develop a framework for supporting security sensitive architecture design and evaluation, we used several approaches to identify the relationships between security quality attributes, their properties and appropriate patterns, namely:

- We studied many sources on building secure information system (e.g. [22, 30, 33-35]), and selected the essential quality properties of a system required to satisfy security policies. These are called "security attributes" in this paper.
- We studied several patterns (such as [10-14]) suggested for building and deploying secure systems. We rigorously verified the architectural sensitivity of those patterns for security attributes. We found several patterns are not architecturally sensitive - for example, the "account lockout" and "server sandbox" patterns [14]. The former is a mechanism against a password-guessing attack by limiting the number of incorrect attempts, which are implementation or deployment related decisions depending upon organizational policies. The latter is used to contain any damage by deciding the minimum privileges required to run a web server, which is again an implementation and context dependent decision that can be implemented anytime during the life of an application. This classification exercise

helped us select nine patterns that have architectural implications. However, this list is extensible.

- We organized the identified patterns into a template [23] (see Appendix 1 for an example) to summarize their context, forces, available tactics, affected attributes, and supported general scenarios. We reviewed the architecturally sensitive information captured using the template and identified the security properties that can be characterized by the general scenarios supported by each pattern and also can be considered as requirements to achieve security attributes.
- Like [27], we put the discovered relationships into a framework for linking the software architecture and security quality attributes.

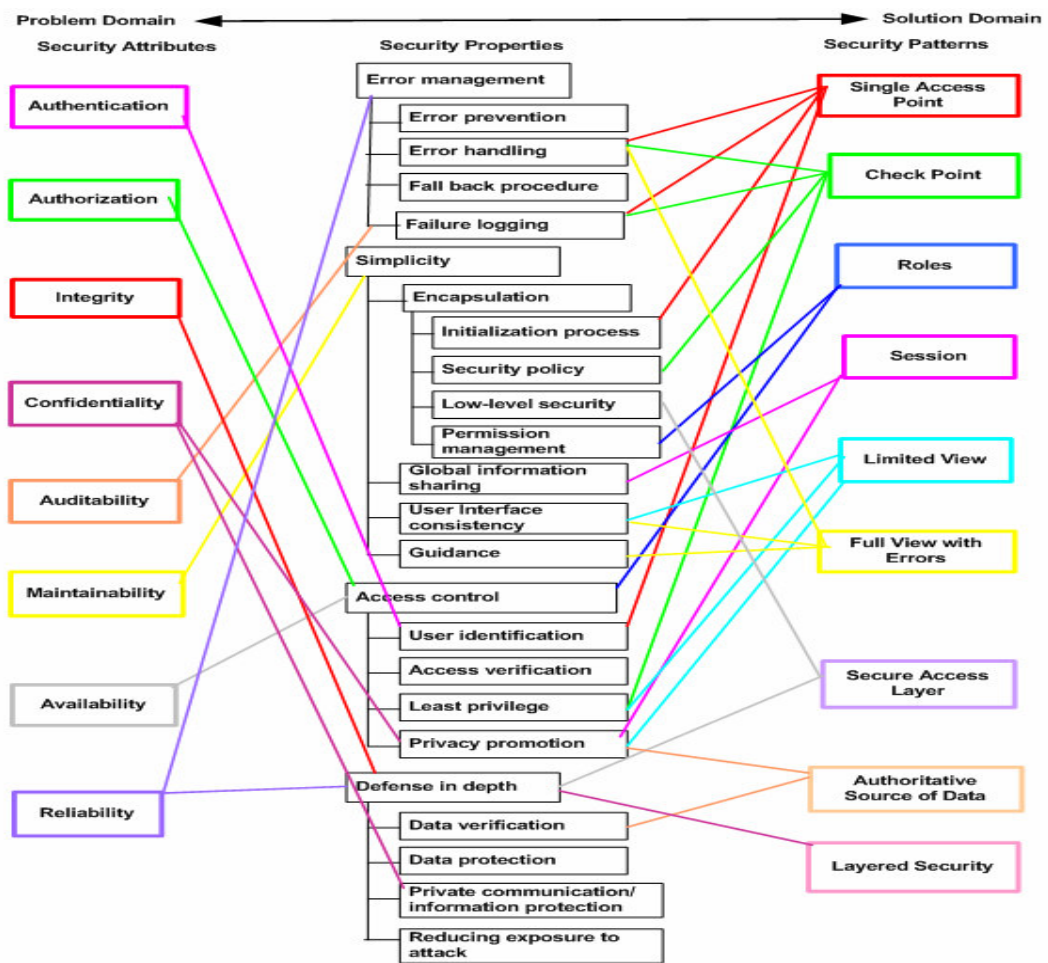


Fig. 2. A Framework for supporting security sensitive architecture design and evaluation

The security framework shown in Fig. 2 presents a collection of security related attributes, properties, and patterns along with the links that form the relationship between software architecture and security quality attribute. To understand one of the ways of using this framework, let us assume that one of the non-functional requirements to be addressed is ease of modifying the security policy, which can be characterized by the “maintainability” quality attribute presented in the left column of the framework. The next step is to identify the security property (or properties) that characterize “maintainability”. According to this framework, the attributes are decomposed into properties, which are placed in the middle column of the framework. This shows that “maintainability” is characterized by the “simplicity” property. This security property itself is decomposed into sub-properties. The security property of interest for this example is “encapsulation of security policy”.

The next step is to identify a pattern that promises to satisfy the desired property. These patterns are presented in the right column of the security framework. For this example, the framework makes it obvious that the “check point” pattern promises to support “security policy”, which in turn supports the ease of modifying security policy requirement.

This framework is expected to be used as a high level guidance to aid in analyzing the architecturally sensitive security issues and their potential solutions. In this way, the security framework connects the security problem domain with the security solution domain. A security attribute is characterized by one or more security properties, which specify security requirements using scenarios, and patterns are used to satisfy those scenarios and in turn properties. Thus, security patterns provide a mechanism to bridge the gap between the problem and solution domains [27].

The framework also demonstrates that the relationships between patterns, properties and attributes are not necessarily binary. Nor are the relationships necessarily positive, however, to keep the diagram uncluttered, only positive relationships have been shown. For example, the “Limited View” pattern has a negative relationship with the “Guidance” property. This is because different types of guidance material need to be provided to different categories of users of the system, which can increase the cognitive load for a user who belongs to many classes of users. Moreover, it is difficult to implement [13].

3.4.2. Framework Usage

With security attributes decomposed into security properties, suitable patterns identified to satisfy those properties, and relationships among them established, we have constructed a mechanism to support security sensitive architecture design and evaluation. This framework can be used by a design team in a several ways. The team may find a certain security property vital for the secure operations of a system. They may realize the importance of a particular property either by looking at the framework, from the stakeholders’ scenarios that characterize a specific property, or from studying a similar system’s properties.

Having realized that a particular security property is important for the system to be designed, the team can use the framework to identify the potential patterns that need to

- Deleted: N
- Deleted: the
- Deleted: ,
- Deleted: which
- Deleted: N

be introduced and which security attributes will be affected (positively or negatively). For example, to develop an online virtual training system various levels of access are required depending upon the status of a user (i.e. trainers, students, coach and others). Having realized the necessity of an appropriate access control mechanism, the team can consult the security framework, which shows that the “Roles” pattern is linked to the “access control” security property, which in turn is linked to four security attributes (i.e. authentication, authorization, integrity, and availability). That means if architect needs to achieve the “access control” security property, they can introduce the “Roles” pattern to support the “access control” property. However, the “Roles” patterns needs to be supported by the “single access point”, “check point”, “session”, “Limited view”, and “authoritative source of data” patterns to achieve all four security attributes linked with the “access control” property.

In another situation, a design team may find that the “auditability” attribute is required. The team can use the security framework to identify the property that is needed to achieve that attribute, namely “Failure logging“ in this case. Having identified the security property that characterizes the desired property “auditability”, the team can use the security framework to find out that the “Single Access point” and “Check point” patterns are needed in order to achieve the “auditability” quality attribute. Moreover, the security framework also helps the team understand why those two patterns need to be used in tandem and which other security properties and attributes are expected to be achieved by using all these patterns in tandem.

Deleted: s

In a third scenario of potential use of the security framework, a software architecture evaluation team may find out that a particular pattern has been used in the architecture being evaluated. They can use the security framework to identify the properties and attributes that are supported by that pattern. Since security properties are non-functional requirements, the security framework can help the evaluation team easily appreciate the security related non-functional requirements that have been considered in the architecture by using a particular security pattern. For example, if the team finds the “Authoritative Source of Data” pattern being used in the architecture being reviewed, they can use the security framework to see that this pattern promotes the “Data Verification” security property, which is positively related to the “Integrity” and “Reliability” security attributes.

4. Conclusion and Future Work

Our main conclusion is that it is important that security issues are sufficiently taken into account during architecture design because certain security sensitive solution need architectural support and it is very difficult and costly to retro-fit security into an implemented system. We observe a gap between security engineering and architecture engineering knowledge. Our objective is to identify and capture architecturally sensitive security knowledge from different sources, and present it in a format that can bridge that knowledge gap. We have identified an initial set of security attributes, properties and patterns, and put them in a security framework that is expected to help

software designers address security issues during software architecture design and evaluation processes.

Our main goal is to improve architectural support for security by providing security knowledge in a format that can support design decisions with an informed knowledge of the consequences of those decisions. More specifically, we intend to raise the awareness about the importance of addressing security related issues during architecture design and review process. Based on our work reported in [23-25], We believe that a systematic approach to identify, capture, and explicitly document the relationships of security attributes, properties, and patterns is an important step towards that goal.

The work reported in this paper cannot be described as complete in its current form as there are a number of things that need to be done before the users - architects/designers/evaluators - of the approach can experience significant benefits in terms of improved realization of security issues during design. These include improved knowledge of the proven security solutions that need architectural support, and ease of identifying and resolving conflicts between security and other quality attributes. In the short term, we plan the following tasks to refine and assess the proposed framework:

- Assess the usefulness and generality of the approach with controlled experiments and case studies.
- Develop a repository to store and access the architecturally sensitive security knowledge.

5. References

- [1] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*. 2 ed. 2003: Addison-Wesley.
- [2] Kazman, R., M. Barbacci, M. Klein, and S.J. Carriere. *Experience with Performing Architecture Tradeoff Analysis*. Proc. of the 21th International Conference on Software Engineering. 1999. New York, USA: ACM Press.
- [3] Kazman, R., L. Bass, G. Abowd, and M. Webb. *SAAM: A Method for Analyzing the Properties of Software Architectures*. Proc. of the 16th ICSE. 1994.
- [4] Bosch, J., *Design & Use of Software Architectures: Adopting and evolving a product-line approach*. 2000: Addison-Wesley.
- [5] Boehm, B. and H. In, *Identifying Quality-Requirement Conflicts*. *IEEE Software*, 1996. **13**(2): p. 25-35.
- [6] Lassing, N., D. Rijssenbrij, and H.v. Vliet. *The goal of software architecture analysis: Confidence building or risk assessment*. Proceedings of First BeNeLux conference on software architecture. 1999.
- [7] CERT. *CERT/CC Statistics 1988-2004*. Last accessed on 26th February 2005, Available from: http://www.cert.org/stats/cert_stats.html.
- [8] Viega, J. and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. 2001: Addison-Wesley.
- [9] Lamsweerde, A.v. *Elaborating Security Requirements by Construction of Intentional Anti-Models*. Proc. of the 26th Int'l. Conf. on Software Eng. (ICSE). 2004. Edinburgh, Scotland.
- [10] Yoder, J. and J. Barcalow. *Architectural Patterns for Enabling Application Security*. Proc. of the 4th Pattern Languages of Programming. 1997. Washington, USA.

- [11] Kienzle, D.M. and M.C. Elder. Final Technical Report: Security Patterns for Web Application Development. Last accessed on 18th February 2005, Available from: <http://www.scrp.net/~celer/securitypatterns/>.
- [12] Kienzle, D.M. and M.C. Elder. Security Patterns: Template and Tutorial. Last accessed on 18th February 2005, Available from: <http://www.scrp.net/~celer/securitypatterns/>.
- [13] Schumacher, M., Security Engineering with Patterns, in Lecture Notes in Computer Science. 2003, Springer-Verlag GmbH.
- [14] Kienzle, D.M., M.C. Elder, D. Tyree, and J. Edwards-Hewitt. Security Patterns Repository - Version 1.0. Last accessed on 18 February 2005, Available from: <http://www.scrp.net/~celer/securitypatterns/>.
- [15] IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology. 1992, New York: Institute of Electrical and Electronic Engineers.
- [16] McCall, J.A., Quality Factors, in Encyclopedia of Software Engineering, J.J. Marciniak, Editor. 1994, John Wiley: New York, U.S.A. p. 958-971.
- [17] ISO/IEC, Information technology - Software product quality: Quality model. ISO/IEC FDIS 9126-1:2000(E).
- [18] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns-Elements of Reusable Object-Oriented Software. 1995, Reading, MA: Addison-Wesley.
- [19] Petersson, K., T. Persson, and B.I. Sanden, Software Architecture as a Combination of Patterns. CrossTalk The Journal of Defense Software Engineering, Oct., 2003.
- [20] Buschmann, F., Pattern-oriented software architecture: a system of patterns. 1996, Chichester; New York: Wiley. xvi, 457 p.
- [21] Hohmann, L., Beyond Software Architecture: Creating and sustaining winning solutions. 2003: Pearson Education, Inc.
- [22] Proctor, P.E. and F.C. Byrnes, The Secured Enterprise: Protecting your information assets. 2002: Prentice Hall PTR.
- [23] Ali-Babar, M. Scenarios, Quality Attributes, and Patterns: Capturing and Using their Synergistic Relationships for Product Line Architectures. Proc. of the Int'l. Workshop on Adopting Product Line Software Engineering. 2004. Busan, South Korea.
- [24] Zhu, L., M. Ali-Babar, and R. Jeffery. Mining Patterns to Support Software Architecture Evaluation. Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture. 2004.
- [25] Ali-Babar, M., B. Kitchenham, P. Maheshwari, and R. Jeffery. Mining Patterns for Improving Architecting Activities - A Research Program and Preliminary Assessment. Proc. of 9th Int'l. conf. on Empirical Assessment in Software Engineering. 2005. Keele, UK.
- [26] Bass, L. and B.E. John, Linking usability to software architecture patterns through general scenarios. Journal of Systems and Software, 2003. **66**(3): p. 187-197.
- [27] Folmer, E., J.v. Gulp, and J. Bosch, A Framework for Capturing the Relationship between Usability and Software Architecture. Software Process Improvement and Practice, 2003. **8**(2): p. 67-87.
- [28] Bass, L., M. Klein, and G. Moreno, Applicability of General Scenarios to the Architecture Tradeoff Analysis Method, Tech Report CMU/SEI-2000-TR-014, Softwar Engineering Institute, Carnegie Mellon University, 2001
- [29] Singh, I., B. Stearns, M. Johnson, and E. Team, Designing Enterprise Applications with the J2EE™ Platform. 2002: Addison Wesley Professional.
- [30] Schneier, B., Secrets and Lies: Digital Security In a networked world. 2000: Wiley Computer Publishing.
- [31] Bass, L., M. Klein, and F. Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method. Proceedings of the 4th International Workshop on Product Family Engineering. 2001. Bilbao, Spain.
- [32] Romanosky, S. Security Design Patterns. Last accessed on 21th February 2005, Available from: <http://www.cgisecurity.com/lib/securityDesignPatterns.pdf>.

- [33] Fegghi, J., J. Fegghi, and P. Williams, Digital Certificates: Applied Internet Security. 1999: Addison Wesley Longman, Inc.
- [34] Juric, M., et al., Patterns Applied to Manage Security, in J2EE Patterns Applied: Real World Development with Pattern Frameworks. 2002, Peer Information.
- [35] Ellison, R.J., A.P. Moore, L. Bass, M. Klein, and F. Bachmann, Security and Survivability Reasoning Frameworks and Architectural Design Tactics, Tech Report CMU/SEI-2004-TR-022, SEI, Carnegie Mellon University, USA, 2004

Appendix 1

Table 1. A template to document and analyse architecturally significant information found in a security pattern.

Pattern Name: The Check Point Pattern		Pattern Type: Security Pattern
Brief description	The Check Point Pattern centralizes and enforces security policy.	
Context	Design an application with a centralized security policy management.	
Problem description	An application needs to be secure from break-in attempts, and the appropriate actions should be taken when such attempts occur.	
Suggested solution	One object should be responsible to encapsulate the algorithm for managing security policy.	
Forces	<ul style="list-style-type: none"> a) It is important to have a centralized mechanism to authenticate and authorized users. b) If users make mistakes, they should receive suitable message and be able to correct them. c) In case of many failed attempts to perform an operation, a suitable action should be taken. d) Error checking code makes it difficult to debug/maintain an application. 	
Available tactics	<ul style="list-style-type: none"> a) Make a security check part of Check Point algorithm, e.g. password checks and time-outs to spoofing. b) For distributed systems Check Point can logically divided into authentication and authorization. c) Consider failure actions, repeatability and deferred checks in designing a Check Point component. d) Failure actions should prompt different actions based on the level of severity. e) Include counters to keep track of the frequency of security violations and parameterize the algorithm. f) Create pluggable security components that can be incorporated in different applications. g) Make security algorithm configurable by tuning on and off some options according to requirements. 	
Affected Attributes		Positively
		Maintainability, Reliability, Auditability
Supported general scenarios		Negatively
		Some checks needs a secondary interface
Supported general scenarios	S1	If a user makes a non-critical mistake, system returns a warning message.
	S2	Security policy shall be easily modified over the life of the application.
	.	
	Sn	
Examples		Ftp server uses <i>Check Point</i> , Xauth uses <i>Check Point</i> to enable X-windows communicate securely with clients.