Selecting Security Patterns that Fulfill Security Requirements

M. Weiss[1], H. Mouratidis[2]
[1]Department of Systems and Computer Engineering, Carleton University, Canada
weiss@sce.carleton.ca
[2]School of Computing and Technology, University of East London, England
haris@uel.ac.uk

Abstract

Over the last few years a large number of security patterns have been proposed. However, this large number of patterns has created a problem in selecting patterns that are appropriate for different security requirements. In this paper, we present a selection approach for security patterns, which allows us to understand in depth the trade-offs involved in the patterns and the implications of a pattern to various security requirements. Moreover, our approach supports the search for a combination of security patterns that will meet given security requirements.

1. Introduction

A considerable effort from the industrial and academic world is focused on the solution of problems related to the security of software systems and it is now generally accepted that security should be treated as part of the software system development process [1] and from the early stages of the software system development process [3], not as an afterthought.

However, an important issue on realizing the above is the lack of security expertise by a large number of software engineers [4]. Towards the solution of this problem, security patterns have been proposed. Security patterns capture design experience and proven solutions to security-related problems in such a way that can be applied by non-security experts.

Over the last few years the number of security patterns has increased considerably [6]. Although this situation has been beneficial for the development of secure software systems, it has created a new problem. It is now difficult to select appropriate security patterns from the large pool of existing patterns that satisfy the security requirements of a system. In fact, representing and selecting security patterns remains largely an empirical task [7]. Using current pattern representations, it is difficult to recognize, under what conditions a pattern should be selected, and understand the consequences of its application, in particular, when choosing between patterns that address the same problem.

Some efforts [6] [12] have been reported in the literature focused on identifying and documenting security patterns. However they have neglected the issue of selecting them. Another line of research [18] [19] [21] has focused on modeling the impact of patterns. That research is important and we consider it complementary to our work. However, such research mostly represents general guidelines and lack formalization that would allow software engineers to verify that the selected patterns actually satisfy the security requirements.

In this paper we present such a formalized approach. Our security patterns selection approach is formalized, on one hand, in terms of the Goal-Oriented Requirements Language (GRL) [8] and on the other hand in terms of Prolog rules. The GRL model shows which contributions a pattern makes on security-related properties such as confidentiality, and the strengths of those contributions. The Prolog rules are used to reason about the evaluation mechanism and to project the effect of combining security patterns. Section 2 briefly introduces the concept of security patterns and introduces our work. Section 3 describes a pattern search engine and Section 4 concludes the paper.

2. Proposed Approach

The goal of our approach is to assist a designer with the selection of security patterns by a) helping them navigate through a possibly large set of patterns or unfamiliar ones through annotations to the patterns (such as impact on NFRs), and b) documenting the rationale for selecting those patterns. The output of pattern selection is a list of patterns, as well the list of requirements that will be met by them and the underlying forces as a way of explaining the selection.

1090-705x/08 $25.00 © 2008 IEEE
DOI 10.1109/RE.2008.32

The concept of patterns originated as an idea in the area of architecture [9] at the late 1970s. According to Alexander et al. [9], a pattern is a three-part rule that defines a relationship between a context, a system of forces that occurs repeatedly in that context, and a solution which allows these forces to resolve themselves. Forces are design trade-offs affected by the pattern.

In our work, the selection of security patterns is formalized in terms of the Goal-oriented Requirements Language (GRL) and Prolog rules. GRL supports reasoning about requirements, and is especially appropriate for dealing with non-functional requirements (NFRs). A GRL model can show the contributions that a pattern makes on security-related NFRs, and the strength of those contributions. The effect of combining patterns can be visualized, but also be reasoned about using an evaluation mechanism. Prolog rules can also be used at this stage: the type and strength of contributions, as well as the evaluation mechanism can be expressed in terms of Prolog rules. However, using Prolog queries we can also use the same pattern representation to search for ways of satisfying a specified level of contributions.

Although a number of pattern representations have been proposed in the literature, for this work, we employ the representation proposed by Araujo and Weiss [21], which was later refined by Mussbacher et al. [22]. The representation is based on GRL, using the notation proposed in [8]. This representation allows us to effectively reason about the forces of each security pattern, and understand the contributions of each security pattern to the various security forces. In our work, we only use a subset of the intentional elements and relationships provided by the Goal-oriented Requirements Language. In particular, we employ task elements (modeled as hexagons) to represent patterns, and softgoal elements (modeled as clouds) to represent the forces of a pattern and NFRs they affect. In addition, decomposition links are employed to model the relationships between patterns and contribution links to represent the contributions (positive or negative) that a pattern makes to specific forces.

A pattern can make these contributions: AND contributions are positive and necessary; OR contributions positive, but not necessary; MAKE contributions are positive and sufficient; HELP indicates that the pattern can positively contribute towards a force, but is not sufficient; BREAK and HURT are the opposites of MAKE and HELP; UNKNOWN indicates that there is a contribution from a pattern to a force, but that the extent and the sense of the contribution is unknown. Consider, for instance, the GRL representation of the Single Access Point pattern [6]. Fig. 1 shows a GRL model of this pattern. It shows the forces, and their impact on security NFRs. So, a Single Access Point provides Accountability (NFR) by ensuring Central Logging (force) of requests to a system.

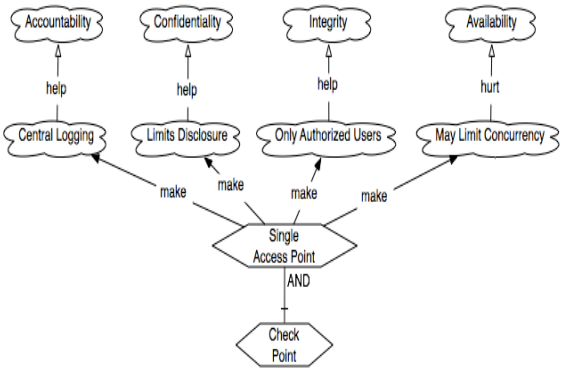

Figure 1. GRL model of Single Access Point

Graphically, closed arrow heads indicate AND, and open arrow heads indicate OR types of contributions. The label on the link indicates the strength of the link. In this model, the selection of forces to describe the security patterns is based, in part, on the analysis of security patterns in terms of their implications on security and non-security related NFRs in [7,23]. This model was defined in the OmniGraffle diagram editor [24], since it was easy to process the XML representation of the model, and extract the structure of the GRL graph, so it could be mapped into Prolog facts. However, other tools such as OME [25] can be used.

In the Prolog representation, the different types of elements are mapped to goals with a satisfaction level and contribution links. Contribution links have strength and a type. In particular, all strengths are represented numerically, with 0.00 (BREAK), 0.25 (HURT), 0.50 (UNKOWN), 0.75 (HELP) and 1.00 (MAKE), and a type of AND, OR or DEPENDS, which indicate in what way the contribution is made. For AND contributions, the weakest contribution decides about the combined effect, and for OR contributions, the strongest contribution. For a DEPENDS link, the satisfaction level of the dependee determines the satisfaction level of the depender. One way of using DEPENDS links here is to model the context in which a pattern is applied, that is, the user's requirements. Patterns are represented by the following predicate:

pattern(Name, FulfilledNFRs, RequiredNFRs).

where *FulfilledNFRs* are the NFRs that positively contribute to security requirements, and *RequiredNFRs* the NFRs that negatively contribute to security requirements. Consider four well-known security patterns for access control: Single Access Point, Check Point, Security Session, and Role-Based Access Control (RBAC) [6]. These patterns can be represented as:

```
pattern('Single Access Point', ['Integrity',
  'Confidentiality', 'Accountability'], ['Availability']).
pattern('Check Point', ['Availability', 'Integrity',
  'Confidentiality'], []).
pattern('Security Session', ['Availability', 'Integrity',
  'Confidentiality', 'Accountability', 'Usability'], []).
pattern('RBAC', ['Manageability', 'Availability',
  'Integrity', 'Confidentiality'], []).
```

We model uses and conflicts relationships between patterns. For the above patterns, we have:

```
uses(and, 'Single Access Point', 'Check Point').
uses(or, 'Check Point', 'Security Session').
uses(or, 'Check Point', 'RBAC').
```

Moreover, satisfaction levels are expressed as a membership function, so that we can use fuzzy logic to evaluate a GRL model. Satisfaction levels can range from 0.00 to 1.00. A satisfaction level of 1.00 (0.00) means that a goal is fully satisfied (denied). Contributions links are modeled using a *contributes* predicate. The first argument indicates the type of contribution (AND, OR, DEPENDS), and the third argument the strength of the contribution. For example, the Single Access Point makes an AND contribution of strength 1.0 toward the goal May Limit Concurrency. Note that there are two sets of contributions: from patterns to forces, and from forces to NFRs. To obtain the satisfaction levels of the top-level goals (Availability etc.), we recursively follow the contribution links. In Prolog, we collect the contributions to each goal, and then run an evaluation method for that goal. Before the value can be returned, we may need to perform these two steps for each contributing subgoal recursively, unless they are leaf nodes of the goal graph. To help collect the contributions, we have defined four rules as shown in Fig. 2.

The first three rules handle OR, AND and DEPENDS, whereas the last rule looks up the satisfaction level of a terminal node. The evaluation rules are defined in Fig. 3. As in fuzzy logic, we evaluate a goal graph by taking the maximum over OR contributions and the minimum of AND contributions. DEPENDS contributions are evaluated to the value of the dependum (dependee), as a depender (dependum) cannot have a higher satisfaction level than the dependum (dependee).

The *propagate* predicate defines how single contributions affect the satisfaction level of a goal. It defines how a satisfaction level VA is mapped by applying a contribution of strength Lambda into a value VB. In the evaluation rules above, we first determine the satisfaction level of a contributing goal, VAT (the "T" represents the tail of the contribution link), then propagate this to a satisfaction level VA. These propagation rules create a fixpoint at 0.5 (UNKNOWN). It should not be possible to "escape" from an UNKNOWN level by some chain of contribution links of HELP or MAKE.

```
% prove a given goal
 prove(G, V) :-
   findall(A/K, contributes(or, A, K, G), L),
   eval(or, L, V).
 prove(G, V) :-
   findall(A/K, contributes(and, A, K, G), L),
   eval(and, L, V).
 prove(G, V) :-
   depends(G, D),
   eval(depends, [G, D], V).
 prove(G, V) :-
   mu(G, V), !.
```

Figure 2. Collecting the contributions to each goal

```
% evaluation  rules
eval(or, [A/K|R], V) :-
  prove(A, VAT), propagate(VAT, K, VA), eval(or,
R, VR), max(VA, VR, V).
eval(or, [A/K], V) :-
  prove(A, VAT), propagate(VAT, K, V).

eval(and, [A/K|R], V) :-
  prove(A, VAT), propagate(VAT, K, VA), eval(and,
R, VR), min(VA, VR, V).
eval(and, [A/K], V) :-
  prove(A, VAT), propagate(VAT, K, V).

eval(depends, [A, B], V) :-
  depends(A, B), prove(B, V).
```

Figure 3. Recursive evaluation of goal graph

3. Pattern Search Engine

To support the selection of security patterns, we have implemented a pattern search engine, which given a set of requirements will find sets of patterns that, together, will satisfy those requirements. A user request indicating security and other non-functional requirements provides input to the search engine. The search engine then attempts to match those requirements against the patterns in a pattern repository. The patterns have been annotated with the information about the NFRs that each pattern fulfills as well as those that it requires, in turn. Relationships between patterns are also represented in those annotations.

At the core of the search engine is an algorithm that matches patterns against user requirements until either there are no more requirements, or no more patterns that can be matched against them. The algorithm has two main parts: an index function that indexes all patterns according to the set of all the requirements they fulfill, and a search function that will search according to the user required security requirements. An output function returns the selected patterns to the users.

Intuitively, the algorithm indexes patterns from a pattern repository $P = \{p_1,\ldots,p_n\}$ according to the set of all the requirements $N = \{n_1,\ldots,n_k\}$ satisfied by patterns in the repository. Users can search the index against their security requirements R ($R \subseteq N$). An output function takes any p_i that fulfills a user requirement and adds it to a set Q ($Q \subseteq P$) that includes the patterns that fulfill user requirements. Since the relationships of the patterns in the repository are annotated, this function also checks for dependencies between patterns. One pattern may require another pattern as a prerequisite (PREREQ). For example, Single Access Point requires Check Point. Therefore, our algorithm also returns all the patterns (p.PREREQ) that are prerequisite for $p \subseteq P$. In addition, the algorithm returns the set $F \subseteq N$ of unfulfilled requirements.

4. Conclusions

In this paper, we introduced a novel approach to find the most suitable security patterns for a given set of security and other non-functional requirements. The proposed pattern search engine also takes into account pattern dependencies, and goodness of fit between the requested requirements (security and others) and those fulfilled by each security pattern.

Our work is based on formalizing security patterns in terms of the Goal-oriented Requirements Language (GRL) and mapping these models into Prolog. As a result, the proposed work demonstrates a number of novel and important contributions: (i) We focus on the explicit consideration of the forces of each pattern for the selection process. This allows us to understand, in depth, the trade-offs involved in the patterns and the implications of this pattern to the various security concerns; (ii) We are concerned with the relationships of the patterns at the pattern language level. In reality, some security patterns can only be applied after certain other security patterns have been already applied. Therefore, to effectively select a set of patterns we must identify and explicitly consider their relationships; (iii) We formalize the representation of patterns. This allows us to effectively and accurately describe how each pattern makes a distinct contribution to a non-functional security requirement. The formalization also allows us to uncover liabilities imposed by a pattern, which are not easy to identify from a textual representation. Finally, formalization allows us to automate parts of the pattern selection process.

References

[1] H. Mouratidis, P. Giorgini, Integrating Security and Software Engineering: Advances and Future Visions, Idea Group Publishing, 2006.

[3] A. van Lamsweerde, Elaborating Security Requirements by Construction of Intentional Anti-Models, in Proceedings of the 26th International Conference on Software Engineering (ICSE), 148-157, IEEE-ACM, 2004.

[4] M. Schumacher, Security Engineering with Patterns: Origins, Theoretical Models, and New Applications, Lecture Notes in Computer Science, 2754, Springer, 2003.

[6] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad, Security Patterns: Integrating Security and Systems Engineering, Willey, 2005

[7] M. Weiss, Modelling Security Patterns Using NFR Analysis, in Integrating Security and Software Engineering: Advances and Future Vision, H. Mouratidis, P. Giorgni (eds), Idea Group Publishing, 2006

[8] GRL, Goal-oriented Requirements Engineering, http://www.cs.toronto.edu/km/GRL (last accessed 23/2/2007)

[9] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language, Oxford University Press, 1979

[12] M. Schumacher, U. Roedig, Security Engineering with Patterns, in the Proceedings of the 8th Conference on Pattern Languages for Programs (PLoP 2001), 2001

[18] L. Chung, K. Cooper, and A. Yi, Developing Adaptable Software Architectures Using Design Patterns: An NFR Approach. Computer Standards & Interfaces, 25, 253-260, 2003

[19] D. Gross, and E. Yu, From Non-Functional Requirements to Design through Patterns. Requirements Engineering, 6(1), 18-36, Springer, 2001

[21] I. Araujo, M. Weiss, Linking Non-Functional Requirements and Patterns, Conference on Pattern Languages of Programs (PLoP), 2002

[22] G. Mussbacher, D. Amyot, and M. Weiss, Formalizing Architectural Patterns with the Goal-Oriented Requirement Language, Nordic Pattern Languages of Programs Conference (VikingPLoP), 2006

[23] R. Wassermann, B. Cheng, Security Patterns, Technical Report, MSU-CSE-03-23, Michigan State University, 2003

[24] OmniGroup, Omnigraffle, http://www.omnigroup.com/applications/omnigraffle/ (Last accessed: 23/2/2007)

[25] Organisation Modelling Environment, OME, http://www.cs.toronto.edu/km/ome/ (Last accessed: 23/2/2007)