A Prototype Tool for Modeling and Analyzing Security Requirements from A Holistic Viewpoint

Tong Li, Jennifer Horkoff, John Mylopoulos

University of Trento, Trento, Italy
{tong.li,horkoff,jm}@disi.unitn.it

Abstract. Security breaches in large socio-technical systems cost billions. Many breaches can be attributed to the piecemeal security design, leaving parts of the system vulnerable while others are over-protected. We advocate holistic security design, and have introduced techniques to support security analysis across multiple layers. This paper presents MUSER, a prototype that assists security requirements analysts dealing with security requirements from a holistic viewpoint based on a three-layer framework. Our prototype analyzes security requirements and related security mechanisms in business layer, application layer, and physical layer. The prototype captures the influences of security mechanisms, which one layer enforces on the other layers, and supports deriving holistic security solutions that tackle security concerns in all layers. We demonstrate the usage of MUSER via a smart grid scenario.
Keyword: Security Requirements · Goal Model · Multilayer · Socio-Technical System · Demo Tool

1 Introduction

Socio-technical systems (STSs) are organizational systems consisting of people, business processes, software applications, and hardware components. As the complexity of STS increases over time, a growing number of security breaches are reported [1].

A common theme for many of these breaches is that security solutions are dealt with in a piecemeal fashion, in which security analysis carried out in one part of the system does not take into account security designs in other parts. For example, when designing an encryption function, a smart meter application can either implement encryption by itself or depend on an external component implemented in a specialized chip. If the system is viewed in a piecemeal fashion, for example, focusing only on the software issues, these two security mechanisms deliver the same function. However, these two alternatives have different influences on requirements of the physical devices, as the latter one requires that the related hardware device should not be accessed by non-authorized people [2], a factor often not accounted for during physical design of the system. As reported in [2], attacks that exploiting this vulnerability have been done by bus-snooping.

In this paper, we present MUSER (MUltilayer SEcurity Requirements analysis tool), a prototype that supports analyzing security requirements of STSs from a holistic point of view, implementing our technique proposed in [3]. Our approach structures STSs into three layers, namely, business layer, application layer, and physical layer. By carrying out analysis both inside one layer and across layers, the approach is intended to generate holistic security solutions for STSs with regard to stakeholder's high-level security needs. To assist the security analysis, our prototype provides the following features: 1) multilayer requirements modeling, 2) hierarchical security requirements refinement, 3) identification of critical security requirements, 4) generation of potential security mechanisms, 5) cross-layer security influence analysis.

In the reminder of this paper, we first describe our analysis approach in Sec. 2, according to which we design our prototype. Next, we introduce the architecture and functionality of our prototype in Sec. 3, and illustrate the utility of the prototype in Sec. 4. Finally, Sec. 5 concludes the paper.

2 Three-Layer Security Requirement Analysis Approach

Our previous work [3] has proposed a goal-oriented three-layer analysis framework, which analyzes security requirements of STSs. This approach adopts concepts from existing well-established goal-oriented modeling languages, such as Techne [4] and i* [5]. To support the three-layer framework, we specialize the concepts *Goal* and *Task* into layer-specific concepts. For example, *Task* is specialized as *Business Process Activity* in the business layer, while it is specialized as *Application Function* in the application layer. In addition, we define *Security Goal* as a special *Softgoal*, and assign it with formal semantics via four dimensions: *Importance*, *Security Attribute*, *Asset*, and *Interval*. For example, a security goal *Medium Integrity[customer information, measure energy consumption]* describes a security requirement "protecting integrity of customer information during the executing period of measure energy consumption to a medium degree".

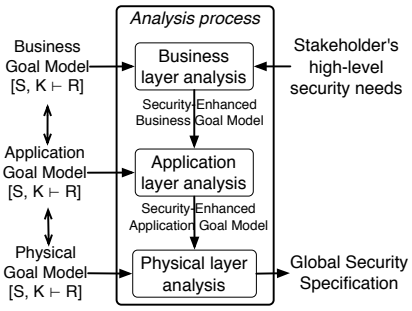


Fig. 1: Overview of the three-layer analysis framework

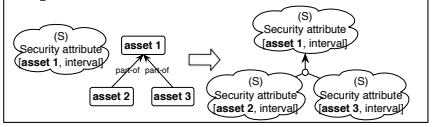
Asset-based Refinement Rule
Content: If an asset consists of sub-assets, a security goal that concerns this asset could be refined to more detailed ones, which focus on the sub-assets.
Datalog Rule: $and\_refined\_sec\_goal(IMP, SA, AS2, INT, SG) :- sec\_goal(SG), importance(IMP), sec\_attribute(SA), asset(AS2), interval(INT), part\_of(AS2, AS1), asset(AS1), has\_properties(SG, IMP, SA, AS1, INT).$
Graphical Transformation Pattern: 

Table 1: Asset-based refinement rule

Our approach analyzes the requirements problem [6] within each of the three layers respectively. In this sense, each layer has its own requirements

R, which are satisfied by its own specifications S, under layer-specific domain assumptions K, i.e. $S, K \vdash R$. In particular, specifications in one layer dictate requirements in lower layers, indicated in Fig. 1. By capturing the cross-layer interactions and analyzing the requirements problem within each individual layer, our approach aims at dealing with security requirements of STSs from a holistic viewpoint. Taking stakeholder's high-level security requirements and layer-specific goal models as input, the approach analyzes detailed security requirements throughout the three layers and finally produces holistic security solutions, which tackle security issues in all three layers (Fig. 1).

The approach includes 23 transformation rules, which guide the corresponding security analysis tasks. Specifically, the rules support refining, simplifying and operationalizing security goals within individual layers, as well as transferring security concerns across layers. All of these rules have been implemented in Datalog [7], which can be automatically inferred with corresponding inference engines, such as DLV [8]. Each rule has been documented in a template, consisting of three sections: *Content*, *Datalog Rule* and *Graphical Transformation Pattern*. Table 1 shows an example regarding the asset-based refinement rule, which refines a security goal via its asset dimension. The full list of transformation rules is available online [1].

3 MUSER Prototype Tool

MUSER (MUltilayer SEcurity Requirements analysis tool) is a prototype, which is designed to support the application of the approach summarized in Sec. 2. The prototype is a Java-based program, which is developed on the top of a specialized and powerful diagramming application *OmniGraffle* [2]. Fig. 2 shows the architecture of the prototype, which consists of four components: *control*, *view*, *model*, and *inference*. We introduce them respectively as below.

Control Component controls the logic of the whole prototype and coordinates other components to deliver inference functions to users. When receiving users' inference requests, it imports related models from the view component, generates a formal model specification in terms of text files, and calls the inference component to carry out corresponding reasoning tasks. According to the reasoning results returned by the inference component, the control component updates related model information and reflects them on the view component.

View Component supports users with graphic modeling, as well as shows graphical inference results to users. The main requirements for this component include: 1) support goal-oriented modeling and allow customized notations; 2) support multilayer modeling, i.e. modeling in different views; 3) be connected with inference component to support reasoning. Although there are a number of available goal modeling tools, such as Open OME, STS-ml [3], none of them can support

[1] http://goo.gl/Pd0TGw
[2] http://www.omnigroup.com/omnigraffle
[3] http://istar.rwth-aachen.de/tiki-index.php?page=i*+Tools

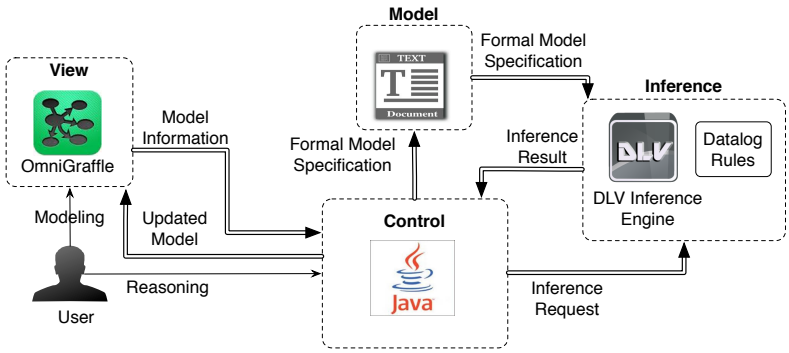


Fig. 2: An overall architecture of the prototype

our multilayer goal modeling and reasoning. We choose a specialized modeling application *OmniGraffle* as the view component of our prototype, meeting all requirements. Particularly, this application has many useful modeling features, such as automatic layout, outline view, and various export formats. As we have defined a number of interfaces for the view component, through which it interacts with the control component, the view component can be replaced by other applications that comply with the interfaces.

Inference Component implements the security analysis rules proposed in our approach and automates corresponding analysis tasks. All the rules are implemented in terms of Datalog rules and facts. Particularly, we leverage DLV inference engine [8] to carry out inference actions. This component receives requests from the control component, and returns inference results back afterwards.

Model Component is responsible for storing the formal models, which are required by the inference component. We use text files for storage, which fit our current needs. If the analysis involves very complex and large models in the future, we will replace the text files with specialized database applications.

3.1 Functionality

Building on the above architecture, we implement a number of functions, which assist the security analysis tasks. As shown in Fig. 3, within one single layer, we first refine and simplify security goals to identify concrete and critical ones. Then, the critical security goals are operationalized into possible security mechanisms and left to security analysts to select. After that, we transfer security concerns downward to its lower layer and iteratively carry out all above security analysis there. The main features of the tool are as follows:

– *Security Goal Refinement:* As a coarse-grained security goal is normally more difficult to analyze and operationalize than a fine-grained one, security analysts need to refine an abstract security goal into sub-goals. The prototype

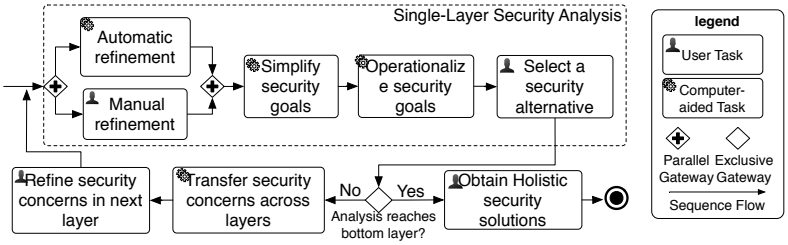


Fig. 3: Detailed security analysis procedure

supports automatically refining a security goal via any of these three dimensions: *security attribute, asset*, and *interval*. Particularly, it provides two ways to automate this analysis. First, it supports a single step of refinement with regard to a particular dimension. Secondly, the prototype could simulate an exhaustive analysis, which explores all possible refinements for the target security goals, to cover all related security goals. Not surprisingly, this simulation could produce a very huge refinement trees, such as shown in Fig. 5. Thus, this analysis has to rely on simplification analysis for filtering non-critical security goals.

- *Security Goal Simplification:* When dealing with a large number of security goals, analysts may not have enough time to go through each of them to determine which is critical and requires further treatments. To release analysts from scrutinizing all security goals, our prototype supports automatic identification of critical security goals, which takes into account the *applicability* and *risk level* of security goals. For example, if a security goal concerns *data confidentiality* of *energy production data* during the execution of *measure energy consumption*, which does not involve with the *energy production data*, then this security goal is not applicable and is further identified as non-critical. In addition, for an applicable security goal, if its risk level is *low* or *medium*, it is identified as non-critical.
- *Security Goal Operationalization:* To effectively achieve critical security goals, our prototype assists analysts in designing security mechanisms based on existing security patterns [9]. A security pattern consists of a security attribute and a security mechanism that is supposed to satisfy that security attribute. When operationalizing a critical security goal, our prototype identifies all matched security patterns with regard to security attributes and generates corresponding security mechanisms for the critical security goal.
- *Cross-Layer Analysis:* To analyze security requirements from a holistic view, the influences of security analysis results in one layer should be propagated to layers below. This analysis takes into account designed security mechanisms and untreated critical security goals in the upper-layer, and generates corresponding security concerns in the lower-layer. For example, an untreated security goal in the business layer, which concerns data confidentiality, is refined into two sub-goals in the application layer that target corresponding applications according to the cross-layer analysis rules [3].

4 Demonstration

We demonstrate the utility of our prototype by holistically analyzing the security requirements of a smart grid example. Particularly, we focus on the real-time pricing scenario.

Real-time Pricing Scenario: *To continuously provide energy to customers, the energy provider needs to balance the load on the power grid to avoid blackouts. To this end, the provider applies a real-time pricing strategy, which adjusts energy price according to current load of the power grid. This strategy requires periodically collecting customer's energy consumption data, based on which a new price is calculated. In this scenario, the energy provider highly relies on the integrity of the energy consumption data.*

According to the detailed analysis procedure shown in Fig. 3, we use our prototype to analyze the security requirements for the above scenario. Our analysis technique will apply to each layer, here we present the security analysis within the business layer, with steps as follows:

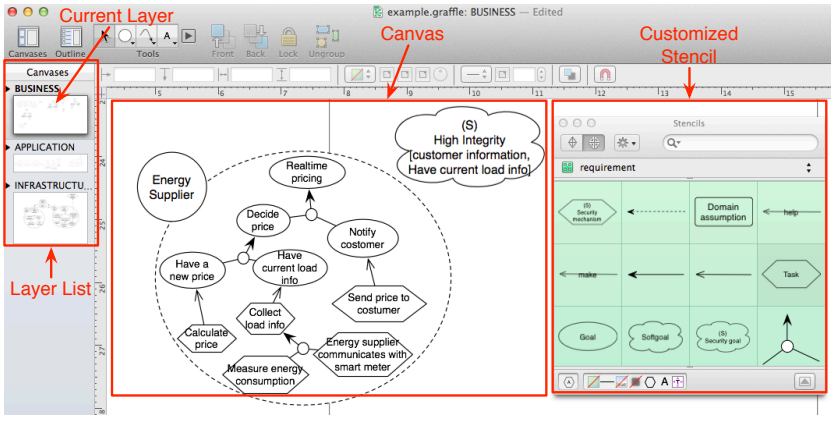


Fig. 4: Graphical modeling interface

1. *Building An Initial Requirements Model:* We first build the requirements model within the business layer, including the high-level security requirements. Fig. 4 shows the graphical modeling interface, which consists of several key components that are highlighted in rectangles. The *Layer List* shows three canvases that are intended for modeling requirements in the three layers. A customized stencil is shown in the bottom part of Fig. 4. Elements in the stencil are used to draw the requirements models, which is presented in the center of the canvas. Due to space limitation Fig. 4 only shows a part of the requirements model.
2. *Refine High-Level Security Goals:* Given the high-level security goal *High Integrity[customer information, Have current load info]* (Fig. 4). We use the prototype to adopt an exhaustive refinement strategy to cover all potential security concerns introduced by this security goal. Not surprisingly, this strategy results in a large refinement tree, which contains 72 potential security goals, shown in Fig. 5. Limited by space, we replace the content of each

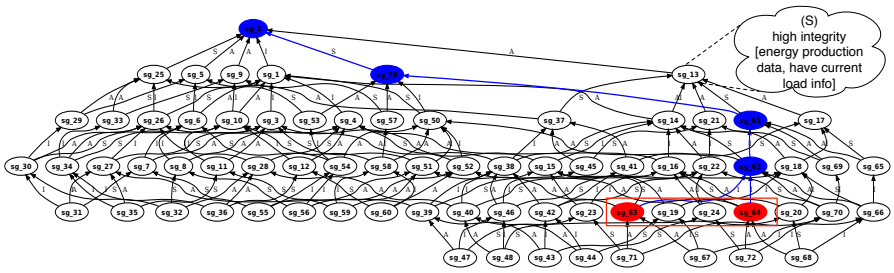


Fig. 5: Example of exhaustive refinement

security goal with its id, and only show the details of one security goal in the upper right corner.

3. *Simplify Security Goals:* As the exhaustive refinement analysis generates a large refinement tree of security goals, which are almost impossible to be analyzed manually, we use our prototype to automatically identify critical security goals. As highlighted by the rectangle in Fig. 5, 2 out of 72 security goals (the filled elements) are identified as critical. Based on these critical security goals, the prototype follows a bottom-up algorithm to calculate the best refinement path, which refines the top-level security goal into the critical ones with minimum steps (the filled elements above the rectangle in Fig. 5). After this simplification analysis, our prototype can generate a simplified security goal model, which contains only the critical security goals and the corresponding best refinement paths.
4. *Operationalize Security Goals:* Facing the two critical security goals, our prototype automatically generates four potential security mechanisms for each of them according to the security patterns [9]. Fig. 6 shows the operationalization of one critical security goal. It is worth noting the security mechanisms suggested by the security patterns may not fit for the security goals in particular system settings, so further judgments from security analysts are required. Furthermore, the prototype calculates all of the alternative security solutions which treat all critical security goals. In our example, the prototype totally generates 25 alternative solutions. Then the security analyst should choose one solution among the alternatives.
5. *Transfer Security Concerns:* After security analysts determine specific security solutions in the business layer, our prototype supports automatically transferring the related security concerns to the application layer, i.e. generating new security requirements with regard to the result of security analysis in this layer. Fig. 7 shows an example of such transfers with regard to the security mechanism *Cryptographic control* selected in the business layer. Note that choosing different security solutions in the business layer will result in different security requirements in the application layer.
6. *Iterative Security Analysis:* After transferring security concerns, we can iteratively carry out security analysis in the application layer and the physical layer, where our prototype automates tasks as in the business layer. Finally,

we will derive holistic security solutions by synthesizing security solutions selected in each individual layer.

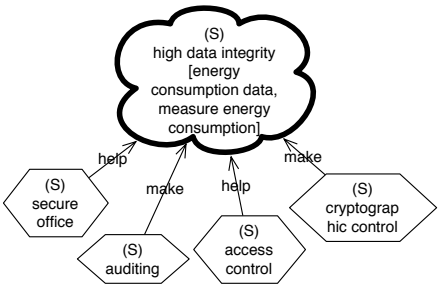


Fig. 6: Example of operationalization

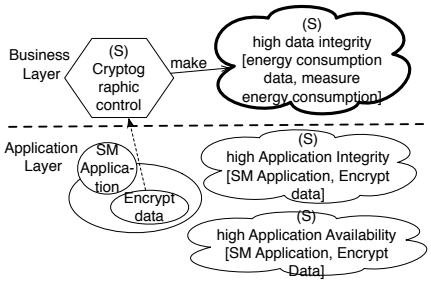


Fig. 7: Example of cross-layer analysis

5 Summary and Future Work

In this paper, we present MUSER, a prototype for analyzing security requirements of STSs from a holistic viewpoint. This prototype is designed to implement our proposed three-layer analysis approach [3], which supports security analysis both within one layer and across layers. In the future, we plan to carry out empirical experiments with our prototype to evaluate its usability and performance, and make further improvements.

Acknowledgements This work is supported by ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution".

References

1. Tony Flick and Justin Morehouse. *Securing the smart grid: next generation power grid security*. Elsevier, 2010.
2. M Carpenter, T Goodspeed, B Singletary, E Skoudis, and J Wright. Advanced metering infrastructure attack methodology. *InGuardians white paper*, 2009.
3. Jennifer Horkoff Tong Li. Dealing with security requirements for socio-technical systems: A holistic approach. In *the 26th International Conference on Advanced Information Systems Engineering (CAiSE'14)*. Accepted, 2014.
4. I.J. Jureta, A. Borgida, N.A. Ernst, and J. Mylopoulos. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proc. of RE'10*, pages 115–124, 2010.
5. Eric Yu. Towards modelling and reasoning support for early-phase requirements engineering. pages 226–235. IEEE Computer Soc. Press, 1997.
6. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.
7. Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418, 1997.
8. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.
9. Riccardo Scandariato, Koen Yskout, Thomas Heyman, and Wouter Joosen. Architecting software with security patterns. Technical report, KU Leuven, 2008.