ELICITING AND REFINING REQUIREMENTS FOR COMPREHENSIBLE SECURITY

Brandon Broadnax,[1] Pascal Birnstill, [2] Jörn Müller-Quade[1] and Jürgen Beyerer[2]

[1] *{brandon.broadnax|mueller-quade}@kit.edu*
Karlsruhe Institute of Technology, Germany

[2] *{pascal.birnstill|juergen.beyerer}@iosb.fraunhofer.de*
Fraunhofer Institute of Optronics, System Technologies and
Image Exploitation IOSB, Karlsruhe, Germany

Abstract

In this work we introduce the principle of comprehensible security, which demands that the security of an IT system is understandable for stakeholders. In particular, all assumptions made for the security mechanisms of an IT system ought to be well-documented. Based on this principle, we propose a conceptual framework that facilitates communication between developers and stakeholders. Our framework uses a goal-oriented approach where requirements are gradually refined. Each refinement corresponds to a specific stage of the development process. In addition, requirements originating from legal constraints are also considered in our framework, because it is indispensable to consider applicable law when developing an IT system. Furthermore, since designing secure IT systems is an interdisciplinary challenge, our framework was also developed to facilitate collaboration between experts of different subfields of computer science. To this end, our framework provides a method for decomposing security requirements into tasks addressed within specific subfields.

Keywords: Information security, security requirements, comprehensible security

1 INTRODUCTION

Security of a system is often blindly trusted - with disastrous consequences. As a result, confidential data of users or entire companies may become disclosed or large networks may suffer DDOS attacks. Typically, it is only stated that the system uses some security mechanisms. However, these much-touted security mechanisms are black boxes to users. They may be outdated and therefore completely insecure. Moreover, the security of a system is generally based on unproven assumptions that are not documented. This is a huge problem as these assumptions may, for instance, hold only for a certain period of time, making the system insecure in the future.

Security that is not comprehensible should not be trusted at all. We therefore put forward the principle of comprehensible security which demands that the security of an IT system is understandable for stakeholders. In particular, all assumptions made during security analysis have to be well-documented and open to public inspection. Consequently, when discussing the security of a software project with stakeholders, one should be able to explain why, how, and under which assumptions the specification, design, and implementation ensure the security-related requirements. In the following we mention various problems that have to be tackled in order to achieve comprehensible security in requirements engineering.

First of all, security-related requirements are introduced on different levels of abstraction. Stakeholder requirements are typically phrased in an informal language in a rather ambiguous manner. This is in parts due to the fact that stakeholders are typically technical laymen that cannot express their needs in a precise way. On a lower level of abstraction, security mechanisms and the guarantees they provide are specified in a very technical language, however. If one wants to be able to explain security to the stakeholders, one must find a way to bridge this gap between the informal requirements of stakeholders and the specification of the final implementation of the system.

Furthermore, security issues have to be tackled in an interdisciplinary context. Naturally, aspects of security are addressed by various subfields of computer science such as cryptography, access and usage control, network security, software engineering, software verification, compiler construction, etc. Overcoming patchwork security however requires that the security mechanisms of these subfields are combined in a reasonable way. In particular, this combination of methods should be done in such a way that security of the system is comprehensible to the developers of the system. Hence, a method that facilitates collaboration between experts of different subfields of computer science is needed. Additionally, all assumptions made by the various domain experts for the security of their mechanisms have to be documented in a way that is comprehensible for stakeholders.

Additional complexity comes from conflicts between different security-related requirements and also between security-related requirements and other functional or non-functional requirements. These conflicts can reveal themselves at different stages of a development process. For instance, although we can already point out during requirements elicitation that there is a trade-off relationship between confidentiality, integrity, and availability, the extent to which we will be able to meet a specific requirement in terms of latency depends on the concrete choices of mechanisms for ensuring confidentiality and integrity during system design or implementation. In order to be able to solve such conflicts along the way, requirements have to be annotated with priorities. In addition, conflicts with legal requirements constitute a special case. As they are non-negotiable, they override conflicting requirements. Handling such cases necessarily requires an additional consultation of requirements engineers and stakeholders.

Our contribution is a conceptual framework, which (i) refines security-related requirements of stakeholders in a way that is understandable for stakeholders (ii) facilitates collaboration between experts of different subfields of computer science (iii) provides means for recognizing and resolving conflicts at each stage of the development process. Throughout this paper, we assume non-security-related requirements to be known, i.e., they have already been elicited.

2 RELATED WORK

Requirements engineering (RE), established in the mid-1970s, is the process of identifying, analyzing, documenting and maintaining requirements. It can be roughly broken down into four activities (cf. [15]): *requirements elicitation*, *requirements* analysis & negotiation, *requirements specification*, and *requirements validation*. Requirements elicitation (also called "‘requirement gathering"’) is the practice of collecting the requirements of a system from stakeholders. The objective of requirements analysis and negotiation is to check requirements and resolve conflicts among the elicited requirements of the system. The result of requirements specification is a description of all the functional and non-functional requirements of a system under development, possibly including a set of use cases. Finally, requirements validation involves checking that the documented requirements actually meet stakeholder needs.

Goal modeling is a modeling type in RE were security requirements are represented in the form of stakeholder goals. A goal-oriented approach in RE can have many advantages (cf. [20]). Eliciting goals naturally leads to the questions "why", "how" and "how else". This way, it is less likely to miss stakeholder requirements or to over-specify them. Furthermore, alternative design options are revealed in the process thus preventing premature design decisions. Modeling requirements as goals can also help to cope with complex requirements as large goals can be decomposed into smaller goals that are easier to realize. Moreover, goals can facilitate identifying conflicts, as typically meeting one goal can complicate or even make it impossible to meet other goals. The various trade-offs between requirements that arise can be resolved more easily with goals. Finally, goal modeling enables to measure requirement completeness by defining requirements to be complete if they fulfill all elicited goals.

Goal modeling has been proposed in literature at various stages of RE. For instance, goals are used for requirements elicitation by the frameworks GOMS [3], ORDIT [5], Ellis'94 [9], i* [21, 22], ISAC [13], F^3 [2]. Goals can also be found in the context of requirements negotiation in the frameworks SIBYL [11], REMAP [17], Duffy'95 [6] or the Reasoning Loop Model [12]. The frameworks KAOS [4], GBRAM [1], and NFR [14] use goals at the requirements specification level. Finally, a goal-oriented approach for requirements validation was adopted for the frameworks GSN [19] and GQM [18]. Goal modeling has been proposed as a strategy for analyzing security trade-offs in [7]. This approach has been developed further in [8] where the authors propose to employ goal modeling to systematically analyze the vulnerabilities of a system design and their effects.

3 FUZZINESS OF SECURITY REQUIREMENTS

We chose the modeling type "goal modeling" for identifying and analyzing security requirements. This model type was chosen for the reason that goals can be expressed by the stakeholders from their individual perspectives. Therefore, goals are a natural representation of security requirements in the context of comprehensible security. Furthermore, we have also adopted the notions "soft goal" and "hard goal" from the i*-framework ([21, 22]) as these notions are suitable tools for decomposing security requirements in a judicious and comprehensible manner.

As stated above, we assume throughout this paper that the functional requirements have already been elicited. We call the class of all users as well as the set of all elicited functional requirements a scenario. Starting from a scenario, the developers elicit the security requirements by interacting with the stakeholders. In general, the requirements that are expressed by the stakeholders are not precise enough for a technical implementation. Therefore, a framework was developed for describing the process from the security requirements of the stakeholders to the technical requirements of the system under development. In this framework, one starts from the assets of a stakeholder. Assets are resources that are of some particular (tangible or intangible) value for a stakeholder. A stakeholder's assets can be potentially harmed by the system under development. This leads to the notion of soft goals. On the level of abstraction of stakeholder requirements, soft goals specify which property of a particular asset must be preserved. For instance, a stakeholder can demand that his personal data are "kept secret", his reputation remains "undamaged" or that his money is "used moderately". The notion of soft goals is used to model the phase of requirements elicitation, where the various security requirements of the stakeholders are iden- tied. Here, it is assumed that stakeholders, being technical laymen, can express their requirements only in the form of soft goals, i.e., in the form

"Property Y of asset X must be preserved!"

Note that soft goals are expressed without referring to any functional requirements of the system under development. As soft goals are expressed using an informal language, it is difficult to directly implement them in a technical system. For that reason, the notion of hard goals was introduced. Hard goals are defined as necessary conditions for fulfilling the elicited soft goals in the system under development. They contribute to the abstraction level of the system's specification. Unlike soft goals, hard goals always refer to some functional requirement that is part of the system under consideration. The syntax for hard goals is expressed along the lines of the following phrase:

“<Functional requirement> must not conflict <soft goal>!”

For example, the requirement “Data must be exchanged confidentially!” is a hard goal with respect to the soft goal “Data must be kept secret!” and with respect to the mechanism “Data transmission” that belongs to the functional requirements of the system under development. Since hard goals are expressed as more concrete requirements for a system under development, they are easier to implement and verify. Moreover, since hard goals are still expressed in a non-technical language and since they are directly related to a specific soft goal, they are comprehensible for a stakeholder. The rationale behind this is that a developer, after refining the various soft goals into hard goals, can present these hard goals to a stakeholder, who can then get an overview of the necessary things that have to be done in to fulfill his requirements.

The aforementioned refinement constitutes only the first step in our framework. Hard goals that were derived from soft goals will be further refined into smaller problems that can be tackled by mechanisms within specific fields of computer science (cf. 4).

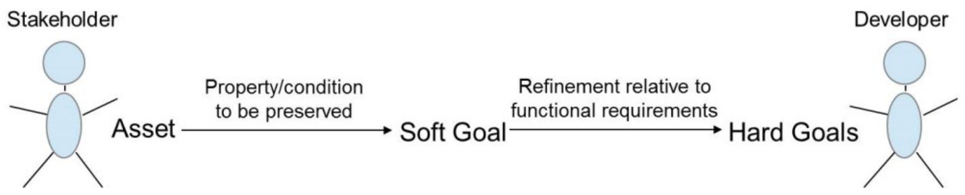
Stakeholder
Developer
Asset
Property/condition to be preserved
Soft Goal
Refinement relative to functional requirements
Hard Goals

Figure 1: Refining soft goals into hard goals

4 INTERDISCIPLINARY DECOMPOSITION

Security of IT systems is an interdisciplinary challenge which requires that mechanisms of various subfields of computer science complement each other. In the following, we show how this can be done in accordance with the principle of comprehensible security. On the level of abstraction of hard goals, we introduce the notion of so-called black box mechanisms. A black box mechanism is a place holder for a class of mechanisms satisfying the considered hard goal. These mechanisms are provided by a particular subfield of computer science. Which specific mechanism is eventually used is a question that is addressed on a lower level of abstraction (cf. 5). Black box mechanisms are chosen on the abstraction level of system design. More specifically, design decisions between black box mechanisms are allowed in case a hard goal can be fulfilled by different black box mechanisms.

The applicability and security of black box mechanisms typically rely on specific abstract assumptions. These assumptions are characteristic of the subfield from which a black box mechanism is provided. In our framework, these assumptions are called separating assumptions. Separating assumptions define the boundaries between subfields. They describe questions which a subfield factors out, passing them to the field of research of another subfield. As separating assumptions rely on the existence and security of mechanisms of other subfields, they can be considered as hard goals addressed by other subfields. In our framework, separating assumptions are expressed in terms of the properties or tasks that are fulfilled for a specific black box mechanism:

“<Property or task> is fulfilled for <black box mechanism>!”

When developing a secure system one should address as many separating assumptions as possible. This leads to an iterative refinement process for hard goals, where the separating assumptions made for the security of a black box mechanism imply hard goals that are fulfilled by other black box mechanisms. In more detail, one proceeds as follows. By rephrasing separating assumptions of a black box mechanism as hard goals, we obtain further hard goals that are tackled by mechanisms of other subfields. These mechanisms give rise to further separating assumptions and so on. This way, one can systematically identify further hard goals until one eventually reaches separating assumptions that one cannot verify, e.g., one has to rely on the correctness of a component or of the operating system because one do not have access to its source code. In our conceptual framework, these unverified assumptions are the trust anchors of the developed system. We call the described iterative process a decomposition of hard goals, which results in a tree structure beneath each hard goal that was derived from a specific soft goal. Usually, the security of a system will depend on several trust anchors. This is because when decomposing a hard goal, one may end up with several separating assumptions that cannot be verified. Furthermore, one normally needs to decompose multiple hard goals, which do not necessarily lead to the same unverifiable separating assumptions. The principle of comprehensible security demands that every separating assumption and trust anchor is explicitly documented in order to make the security of a system as transparent as possible.

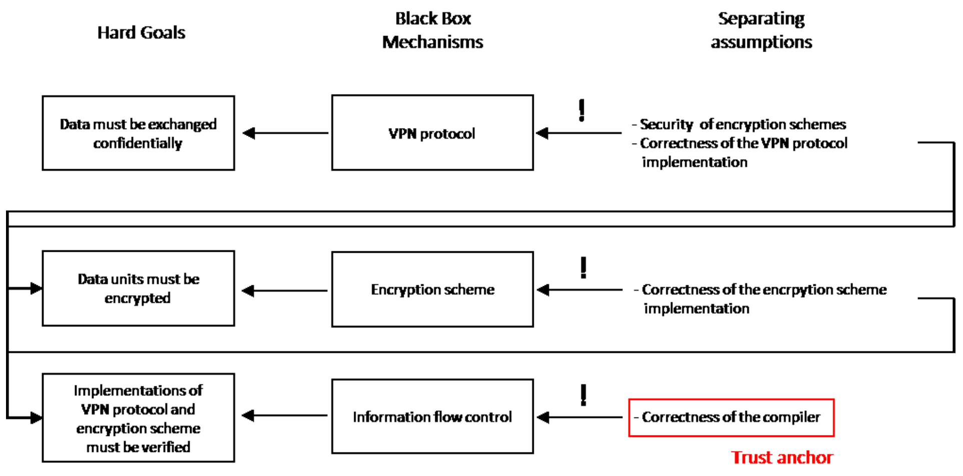
Hard Goals
Black Box Mechanisms
Separating assumptions
Data must be exchanged confidentially
VPN protocol
!
- Security of encryption schemes
- Correctness of the VPN protocol implementation
Data units must be encrypted
Encryption scheme
!
- Correctness of the encrpytion scheme implementation
Implementations of VPN protocol and encryption scheme must be verified
Information flow control
!
- Correctness of the compiler
Trust anchor

Figure 2: Example for the decomposition of a hard goal

Fig. 2 illustrates the decomposition of hard goals through the example of the hard goal *data must be exchanged confidentially*. The black box mechanism *virtual private network (VPN) protocol* fulfills the hard goal under the sep. assump. *security of encryption schemes*, which is addressed by cryptography, and *correctness of the VPN protocol implementation*, which is addressed by software verification. This implies the hard goal *data units must be encrypted*, which leads to the black box mechanism *encryption scheme*. In addition, cryptography also assumes the *correctness of the encryption scheme implementation*. Correctness of the VPN protocol and encryption scheme can be achieved using the black box mechanism information flow control, which assumes the *correctness of the compiler*. Finally, since correctness of the compiler has not been verified in this example, it is a trust anchor of the system.

5 SUBFIELD-SPECIFIC LEVEL

On a more detailed level of abstraction, more precisely when the system design is realized in the system's implementation, black box mechanisms are instantiated by concrete mechanisms, which are called white box mechanisms. White box mechanisms provide the necessary conditions for fulfilling a hard goal. These are called guarantees in our framework (cf. Figure 3).

Guarantees usually cannot be given unconditionally. Rather, specific assumptions have to be made. At the white box level of abstraction, these assumptions are called fundamental assumptions in our framework. Fundamental assumptions comprise subfield-specific foundations. Going back to the example introduced above, assume that we have decided to realize the black box mechanism encryption scheme using (a specific variant of) the RSA cryptosystem. The RSA cryptosystem can be shown to be secure under the fundamental assumption that no efficient algorithm exists to solve the problem of factoring large numbers. Guarantees given by different mechanisms and subfields vary substantially regarding the degree of precision and formalization. Typically, the white box level of abstraction has to be considered when one wants to resolve trade-offs, as concrete mechanisms have to be analyzed for this purpose. We elaborate on this issue in the next section.

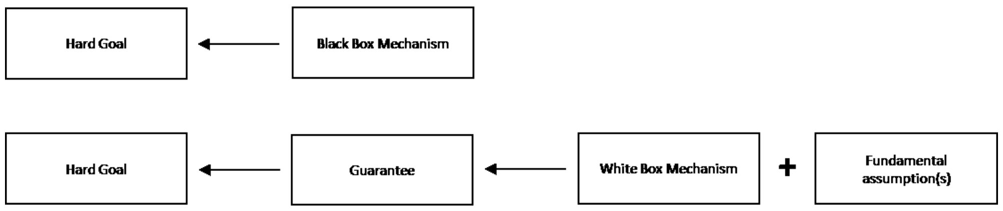

Figure 3: Black box mechanisms are instantiated by white box mechanisms

6 RECOGNIZING AND RESOLVING CONFLICTS

Ambiguity of soft goals isn't the only problem a developer faces during requirements analysis. There can also be conflicts among the various requirements elicited. These conflicts can reveal themselves on various levels of refinement. Conflicts can reveal themselves already during the requirements elicitation phase if there is an inherent trade-off relationship between soft goals (*inherent conflicts*). Conflicts may also emerge as a consequence of using a specific black box mechanism (*derived conflicts*).

The best known inherent conflict is the trade-off relationship between soft goals concerning confidentiality, integrity, and availability, which cannot be maximized at the same time and thus have to be balanced to achieve adequate security. If, for instance, confidentiality is implemented via an encryption scheme and availability is realized via a replication scheme, integrity is hard to preserve since replicates have to be kept synchronized, which introduces additional overhead in terms of encryption, which in turn affects availability.

Although derived conflicts may seem less common, they do emerge when trying to ensure that critical software systems are operated within a secure environment: When realizing a secure environment, we may want to apply black box mechanisms such as deep packet inspection (DPI) on gateways and firewalls for early detection of malware, viruses, or attacks. This requires that payloads of network packets are inspected, which interferes with the goal of exchanging data via confidential communication channels realized via black box mechanisms like VPN protocols that encrypt payloads.

The various trade-offs that might arise through these conflicts have to be identified and discussed with the stakeholder. For this to be possible there has to be a feedback loop to the stakeholders at any level of refinement. In order to cope with these conflicts, soft goals have to be annotated with priorities. These annotations can then help the developer during the development process, e.g., when there are several options for black box mechanisms.

Many conflicts can only be resolved at the white box level of abstraction, as concrete mechanisms are considered at this level of abstraction only. As an example of a conflict to be addressed on the white box level, consider the problem of secure database outsourcing: A database containing confidential data is to be outsourced to a cloud storage provider. Services built on this database have to be able to compute specific queries with a specific upper bound concerning latency. The black box mechanism addressing this problem is confidentiality-preserving indexing (CPI), which can be instantiated by deterministic indexes, order-preserving encryption, or searchable encryption on the white box level. When deciding for a concrete white box mechanism, the type of queries that are to be executed on the database may rule out some candidates for white box mechanisms. In the second step, a white box mechanism with low communication and computation overhead will be chosen in order to minimize latency. In case it turns out that the latency requirements of the stakeholders still cannot be met, latency or security requirements are either relaxed according to given priorities or in consultation with the stakeholders. More details on resolving trade-offs in the context of secure database outsourcing can be found in [10].

6.1 Legal Requirements

Legal requirements are important to consider because even an ideally secure implementation is useless if it does not adhere to applicable law. However, legal requirements are not necessarily implied by the other (non-legal) requirements. This is due to the fact that soft goals are expressed from a stakeholder's individual perspective and stakeholders are typically not only layman with respect to technical issues but also with respect to legal issues. Moreover, legal requirements may even be at odds with other requirements.

Consider the example of video surveillance in subway trains, which help counter acts of violence and vandalism. The large amounts of video data could be pre-evaluated using activity recognition algorithms that automatically alert the police. However, European data protection law prohibits automated individual decisions entailing legal or other adverse consequences for the person(s) affected. Instead, situations have to be assessed by a human operator before any countermeasures are initiated. This legal constraint has to be considered when designing the workflow of the system.

To incorporate requirements originating from legal issues, our framework formally considers the legislator as an additional stakeholder who introduces legal requirements. A common base for soft goals and laws is provided by the notion of assets mentioned above. Like soft goals, laws also originate from specific assets that are to be protected. Unlike soft goals, however, laws cannot be modified so as to resolve conflicts with other soft goals. For this reason, the notion of obligations was introduced. Obligations are non-negotiable requirements that are defined on the same level of abstraction as soft goals. Laws imply obligations. For instance, § 6a of the German Energy Economy Law demands that economically sensitive data such as consumption data of individual households is to be treated as confidential.

7 CONCLUSION

In this paper we have presented a new conceptual framework for requirements engineering. This framework helps a developer during the entire development phase of a system, starting with requirements elicitation. Our framework is tailored in such a way as to make the security of a system under development as comprehensible as possible for a stakeholder. Additionally, our framework facilitates collaboration between developers coming from different fields of computer science by means of an iterative delegation process. Moreover, legal requirements are also considered in our framework because an IT system must adhere to applicable law.

REFERENCES

[1] A. Antón et al. Goal-based requirements analysis. In Requirements Engineering, Proceedings of the 2nd Int. Conf. on, pp. 136-144, IEEE, 1996.

[2] J. Bubenko et al. Objectives driven capture of business rules and of information systems requirements. In Systems, Man and Cybernetics, 'Systems Engineering in the Service of Humans', Proc. of the Int. Conf. on, pp. 670-677, IEEE, 1993.

[3] S. K. Card et al. The psychology of human-computer interaction, 1983.

[4] A. Dardenne et. al. Goal-directed requirements acquisition. Science of computer programming, 20(1):3-50, 1993.

[5] J. Dobson et al. The ordit approach to organisational requirements. In Requirements engineering, pages 87-106. Academic Press Professional, 1994.

[6] D. Duffy et al. A framework for requirements analysis using automated reasoning. In Advanced Information Systems Engineering, pp. 68-81. Springer, 1995.

[7] G. Elahi and E. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In Conceptual Modeling-ER, pp. 375-390, 2007.

[8] G. Elahi et al. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Requirements Engineering 15.1: pp. 41-62, 2010.

[9] C. Ellis and J. Wainer. A conceptual model of groupware. In Proc. of the ACM Conf. on Computer Supported Cooperative Work, pp. 79-88. ACM, 1994.

[10] J. Köhler. Tunable Security for Deployable Data Outsourcing. Karlsruher Institut für Technologie, 2015.

[11] J. Lee and K.-Y. Lai. What's in design rationale? Human-Computer Interaction, 6(3-4):251-280, 1991.

[12] P. Louridas and P. Loucopoulos. A generic model for reflective design. ACM Trans. on Software Engineering and Methodology (TOSEM), 9(2):199-237, 2000.

[13] M. Lundeberg. The isac approach to specification of information systems and its application to the organization of an ifip working conference. In Information systems design methodologies: A comparative review, pp. 173-234. The Netherlands: North-Holland, 1982.

[14] J. Mylopoulos et al. Representing and using nonfunctional requirements: A process-oriented approach. Software Engineering, IEEE Trans. on, 18(6):483-497, 1992.

[15] K. Pohl. Requirements engineering: An overview. RWTH, Fachgruppe Informatik, 1996.

[16] A. Pretschner et al. Distributed usage control. Commun. ACM, 49(9):39-44, 2006.

[17] B. Ramesh and V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. Software Engineering, IEEE Trans. on, 18(6):498-510, 1992.

[18] R. Van Solingen et al. Goal question metric (gqm) approach. Encyclopedia of Software Engineering, 2002.

[19] S. Wilson et al. Safety case development: Current practice, future prospects. In Safety and Reliability of Software Based Systems, pp. 135-156. Springer, 1997.

[20] E. Yu and J. Mylopoulos. Why goal-oriented requirements engineering. In Proc. of the 4th Int. Workshop on Requirements Engineering: Foundations of Software Quality, volume 15, 1998.

[21] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In Requirements Engineering, Proc. of the 3rd IEEE Int. Symposium on, pp. 226-235. IEEE, 1997.

[22] E. S. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In Proc. of the 16th int. conf. on Software engineering, pp. 159-168. IEEE Computer Society Press, 1994.