Picking Battles: the Impact of Trust Assumptions on the Elaboration of Security Requirements

Charles B. Haley[1], Robin C. Laney[1], Jonathan D. Moffett[2], Bashar Nuseibeh[1]

[1] Department of Computing, The Open University,
Walton Hall, Milton Keynes, MK7 6AA, UK
{C.B.Haley, R.C.Laney, B.A.Nuseibeh} [at] open.ac.uk
[2] Department of Computer Science, University of York
Heslington, York, YO10 5DD, UK
jdm [at] cs.york.ac.uk

Abstract. This position paper describes work on *trust assumptions* in the context of security requirements. We show how trust assumptions can affect the scope of the analysis, derivation of security requirements, and in some cases how functionality is realized. An example shows how trust assumptions are used by a requirements engineer to help define and limit the scope of analysis and to document the decisions made during the process.

1 Introduction

Requirements engineering is about determining the *characteristics* of a *system-to-be*, and how well these characteristics fit with the desires of the *stakeholders*. A system-to-be includes all the diverse components needed to achieve its purpose, such as the computers, the people who will use, maintain, and depend on the system and the environment the system exists within. *Stakeholders* are those entities (e.g. people, companies) that have some reason to care about the system's characteristics. A description of these characteristics is the system's *requirements*.

Security requirements are an important component of a system's requirements. They arise because stakeholders assert that some objects, tangible (e.g. cash) or intangible (e.g. information), have direct or indirect value. Such objects are called *assets*, and the stakeholders naturally wish to protect their value. Assets can be harmed, or can be used to cause indirect harm, such as to reputation. Security requirements ensure that these undesirable outcomes cannot take place.

Security requirements often assume the existence of an *attacker*. The goal of an attacker is to cause *harm*. Leaving aside harm caused by accident, if one can show that no attackers exist, then security is irrelevant. An attacker wishes to cause harm by exploiting an asset in some undesirable way. The possibility of such an exploitation is called a *threat*. An *attack* exploits a *vulnerability* in the system to carry out a threat.

It is useful to reason about the attacker as if he or she were a type of stakeholder (e.g. [1; 9; 10]). The attacker would therefore have requirements; he or she wants a system to have characteristics that create vulnerabilities. The requirements engineer

Lecture Notes in Computer Science Vol 2995
http://www.springer.de/comp/lncs/index.html

wants the attacker's requirements to not be met. To accomplish this, one specifies sufficient *constraints* on the behavior of a system to ensure that vulnerabilities are kept to an acceptable minimum [11]. Security requirements specify these constraints.

A system-level analysis is required to obtain security requirements. Without knowledge of a system's components, the requirements engineer is limited to general statements about a system's security needs. Nothing can be said about how the needs are met. To determine security requirements, one must look deeper; we propose to use *problem frames* [8] to accomplish this. In a problem frames analysis, this means looking at and describing the behavior of *domains* within the context of the system.

While reasoning about security, a requirements engineer must make decisions about how much to trust the supplied indicative (observable) properties of domains that make up the system and evaluate the risks associated with being wrong. These decisions are *trust assumptions*, and they can have a fundamental impact on how the system is realized [13]. Trust assumptions can affect which domains must be analyzed, the risk that vulnerabilities exist, and the risk that a system design is stable. During analysis, trust assumptions permit the requirements engineer to pick battles, deciding which domains need further analysis and which do not.

This paper describes combining trust assumptions, problem frames, and threat descriptions in order to aid in derivation of security requirements. Section 2 provides background material on problem frames. Section 3 discusses security requirements. Section 4 describes the role of trust assumptions. Section 5 presents related work, and section 6 concludes.

2 Problem Frames

All problems involve the interaction of domains that exist in the world. The problem frames notation [8] is useful for diagramming the domains involved in a problem and the interconnections (phenomena) between them, and for analyzing their behavior. For example, assume that working with stakeholders produces a requirement "open door when the door-open button is pushed." Figure 1 illustrates satisfying the requirement with a basic automatic door system. The first domain is the door mechanism domain, capable of opening and shutting the door. The second is the domain requesting that the door be opened; including both the 'button' to be pushed and the human pushing the button. The third is the *machine*, the domain being designed to fulfill the requirement that the door open when the button is pushed. The dashed-line oval presents the requirement that the problem is to satisfy. The dashed arrow from the oval indicates which domain is to be constrained by the requirement.

Every domain has *interfaces*, which are defined by the *phenomena* visible to other domains. Descriptions of phenomena of given (existing) domains are indicative; the phenomena and resulting behavior can be observed. Descriptions of

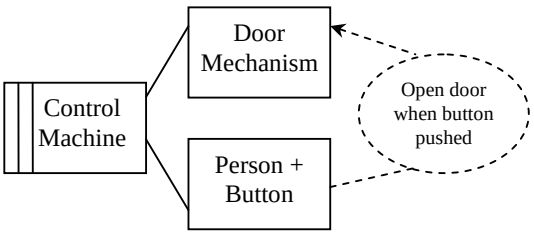

Figure 1 – A basic problem frames diagram

phenomena of designed domains (domains to be built as part of the solution) are optative; one wishes to observe the phenomena in the future. To illustrate the idea of phenomena, consider the person+button domain in Figure 1. The domain might produce the event phenomena ButtonDown and ButtonUp when the button is respectively pushed and released. Alternatively, it might produce the single event OpenDoor, combining the two events into one.

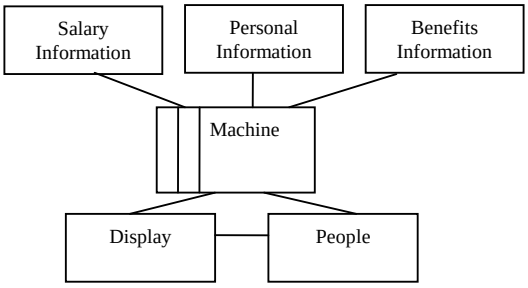

Figure 2 – Example Context Diagram

The two fundamental diagram types in a problem frames analysis are the *context diagram* and the *problem frame diagrams*. The context diagram shows all the domains in a system, and how they are interconnected. The problem frame diagrams each examine a *problem* in the system, showing how a given requirement (problem) is to be satisfied. In systems with only one requirement, the context diagram and the problem frame diagram are almost identical. For most systems, though, the domains in the problem frame diagrams are a projection of the context, showing only the domains or groups of domains of interest to the particular problem.

Figure 2 shows a context diagram for a system that will be used as an example throughout the remainder of this paper. The system is a subset of a Human Resources system. There are two functional requirements, of which we will consider the second.

- Salary, personal, and benefits information shall be able to be entered, changed, and deleted by HR staff. This information is referred to as *payroll information*.
- Users shall have access to kiosks located at convenient locations throughout the building and able to display an 'address list' subset of personal information consisting of any employee's name, office, and work telephone number.

The problem diagram for the second requirement (the 'address list' function) is shown in Figure 3. Phenomena are intentionally omitted. The security requirements will be added in the next section.

3 Security Requirements

Security requirements come into existence to prevent harm by attacks on assets [5; 11]. An asset is something in the context of the system, tangible or not, that is to be protected [7]. A threat is the potential for abuse of an asset that will cause harm in the context of the problem. A vulnerability is a weakness in the system that an attack exploits. Security requirements are constraints on functional requirements,

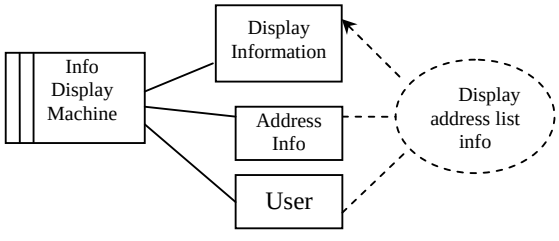

Figure 3 –Address list

intended to reduce the scope of vulnerabilities.

The security community provides general categories for constraints, labeling them using the acronym CIA, and more recently more 'A's [12]:

- Confidentiality: ensure that an asset is visible only to those actors authorized to see it. This is larger than 'read access to a file', as it can include, for example, visibility of a data stream on a network.
- Integrity: ensure that the asset is not corrupted. As above, integrity is larger than 'write access to a file', including operations such as triggering transactions that should not occur.
- Availability: ensure that the asset is readily accessible to actors that need it. Availability is best explained by a counterexample, such as preventing a company from doing business by denying it access to something important.
- Authentication & accountability: ensure that the source of the asset, actor, or action is known. One example is the simple login. More complex examples include mutual authentication (e.g. exchanging cryptography keys) and non-repudiation.

By inverting the sense of these categories, one can construct descriptions of possible threats on assets. These *threat descriptions* are phrases of the form *performing action X on/to asset Y could cause harm Z* [5]. Referring to the example presented above, some possible threat descriptions are:

- Changing salary data could increase salary costs, lowering earnings.
- Exposing addresses (to headhunters) could cause loss of employees, raising costs.

To use the threat descriptions, the requirements engineer examines each problem frame diagram, looking to see if the asset mentioned in the threat is found in the problem. If the asset is found, then the requirements engineer must apply constraints on the problem to ensure that the asset is not vulnerable to being used in the way that the action in the threat description requires. These constraints are security requirements. The security requirements are satisfied by changing the problem in a way that changes the behavior of the domains.

Analysis of Figure 3 shows that there are vulnerabilities that allow the threats to be realized. Attackers can see the data on the network. Nothing prevents an attacker from accessing the system. In order to maintain confidentiality and integrity of the data, the network needs to be protected and employees need to be authenticated. A design decision is made to encrypt data on the network, and appropriate constraints and phenomena are added. Our next problem is employee authentication; we will solve this problem in the next section.

4 Trust Assumptions

A requirements engineer determines how a requirement is satisfied using the characteristics of the domains in the problem. A similar relationship exists between security requirements and trust assumptions; how security requirements are satisfied depends on the trust assumptions made by the requirements engineer.

We use the definition of trust proposed by Grandison & Sloman [4]: "[Trust] is the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context". In our case, the *requirements*

engineer trusts that some domain will participate 'competently and honestly' in the satisfaction of a security requirement in the context of the problem.

Adding trust assumptions serves two purposes. The first is to limit the scope of the analysis to the domains in the context. The second is to document how the requirements engineer chooses to trust other domains that are in the context for some other reason. To illustrate the former, assume a requirement stipulating that the computers operate for up to eight hours in the event of a power failure. The requirements engineer satisfies this requirement by adding backup generators to the system. In most cases, the engineer can trust the manufacturer of the generators to supply equipment without vulnerabilities that permit an attacker to take control of the generators. By making this trust assumption, the requirements engineer does not need to include the supply chain of the generators in the analysis.

Returning to our example, we see that trust assumptions must be added to the diagram to complete the picture. For example, the analysis does not explain why the encrypted networks and authentication are considered secure or how address information is to be protected. The IT organization convinces the requirements engineer that the encryption software and keys built into the system are secure, and that the keys control access to the address information. Choosing to accept the explanations, the engineer adds three trust assumptions (TA1 – TA3) to the problem frame diagram.

There are threats against the name and address information which indicate that confidentiality of the information must be maintained. To counter the threats, the requirements engineer proposes that the information be limited to people having authentication information and able to log in. The IT department refuses on cost grounds. The stakeholders refuse because of ease-of-use.

Further questioning reveals that the front door of the building is protected by a security guard; the guard restricts entrance to authorized personnel. The security manager agrees that the security guard can stand in for authentication. A trust assumption (TA4) is added, having the effect of changing the *people* domain to *employees* by restricting membership to people allowed in by the building security system. Figure 4 shows the resulting problem frames diagram.

The example shows that trust assumptions restrict domain membership. For example, the building security system trust assumption restricts membership of the *people*

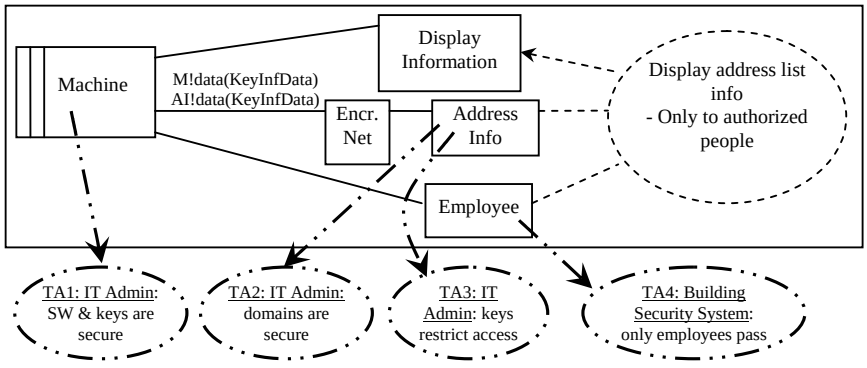

Figure 4 – Address list revisited

domain to people acceptable to the door guard, effectively converting the domain to *employees*.

The *IT Admin: keys restrict access* trust assumption is a special case. The domain being limited is an 'others' domain representing people not permitted to see the data. This domain isn't in the context. Adding the domain and connecting the trust assumption would restrict the domain's membership to null. Rather than adding a null domain, the trust assumption is expressed in terms of its effect and attached to the domain that caused the trust assumption to come into existence.

5 Related Work

We are not aware of other work investigating the capture of a requirements engineer's trust assumptions about the domains that make up the solution to the problem.

Several groups are looking at the role of trust in security requirements engineering. In the *i** framework [14; 16], Yu, Lin, & Mylopoulos take an 'actor, intention, goal' approach where security and trust relationships within the model are modeled as "softgoals": goals that have no quantitative measure for satisfaction. The Tropos project [3] uses the *i** framework, adding wider lifecycle coverage. Gans et al [2] add distrust and "speech acts". Yu and Cysneiros have added privacy to the mix [15]. All of these models are concerned with analyzing trust relations between actors/agents in the running system. As such, an *i** model complements the approach presented here, and in fact can be used to determine the goals and requirements.

He and Antón [6] are concentrating on privacy, working on mechanisms to assist trusting of privacy policies, for example on web sites. They propose a context-based access model. The framework, like *i**, describes run-time properties, not the requirements engineer's assumptions about the domains forming the solution.

6 Conclusions and Future Work

We have described an approach for using trust assumptions while reasoning about security requirements. The approach makes a strong distinction between system requirements and machine specifications, permitting the requirements engineer to choose how to conform to the requirements. The trust assumptions embedded in the domain inform the requirements engineer, better enabling him or her to choose between alternate ways of satisfying the functional requirements while ensuring that vulnerabilities are removed or not created.

Work on trust assumptions is part of a larger context wherein security requirements are determined using the crosscutting properties of threat descriptions [5]. The trust assumptions will play a critical role in analyzing cost and risk. The quantification of the level of trust, not yet used, will be important in this context.

Acknowledgements: The financial support of the Leverhulme Trust is gratefully acknowledged. Thanks also go to Michael Jackson for his many insights about problem frames and requirements, and to the anonymous reviewers for their helpful comments.

References:

1. Crook, R., Ince, D., Lin, L., Nuseibeh, B.: "Security Requirements Engineering: When Anti-Requirements Hit the Fan," In Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02). Essen Germany (2002) 203-205.
2. Gans, G., et al.: "Requirements Modeling for Organization Networks: A (Dis)Trust-Based Approach," In 5th IEEE International Symposium on Requirements Engineering (RE'01). Toronto, Canada: IEEE Computer Society Press (27-31 Aug 2001) 154-165.
3. Giorgini, P., Massacci, F., Mylopoulos, J.: "Requirement Engineering Meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard," In Proceedings of the 22nd International Conference on Conceptual Modeling. Chicago IL USA: Springer-Verlag Heidelberg (13-16 Oct 2003) 263-276.
4. Grandison, T., Sloman, M.: "Trust Management Tools for Internet Applications," In The First International Conference on Trust Management. Heraklion, Crete, Greece: Springer Verlag (28-30 May 2003).
5. Haley, C. B., Laney, R. C., Nuseibeh, B.: "Deriving Security Requirements from Crosscutting Threat Descriptions," In Proceedings of the Fourth International Conference on Aspect-Oriented Software Development (AOSD'04). Lancaster UK: ACM Press (22-26 Mar 2004).
6. He, Q., Antón, A. I.: "A Framework for Modeling Privacy Requirements in Role Engineering" at Ninth International Workshop on Requirements Engineering: Foundation for Software Quality, The 15th Conference on Advanced Information Systems Engineering (CAiSE'03), Klagenfurt/Velden, Austria (16 Jun 2003).
7. ISO/IEC: *Information Technology - Security Techniques - Evaluation Criteria for IT Security - Part 1: Introduction and General Model*. ISO/IEC: Geneva Switzerland, 15408-1 (1 Dec 1999).
8. Jackson, M.: *Problem Frames*. Addison Wesley, 2001.
9. van Lamsweerde, A., Brohez, S., De Landtsheer, R., Janssens, D.: "From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering" at Requirements for High Assurance Systems Workshop (RHAS'03), Eleventh International Requirements Engineering Conference (RE'03), Monterey, CA USA (8 Sep 2003).
10. Lin, L., Nuseibeh, B., Ince, D., Jackson, M., Moffett, J.: "Introducing Abuse Frames for Analyzing Security Requirements," In Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03). Monterey CA USA (8-12 Sep 2003) 371-372.
11. Moffett, J. D., Nuseibeh, B.: *A Framework for Security Requirements Engineering*, Department of Computer Science. University of York, UK, YCS368 (August 2003).
12. Pfleeger, C. P., Pfleeger, S. L.: *Security in Computing*. Prentice Hall, 2002.
13. Viega, J., McGraw, G.: *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley, 2002.
14. Yu, E.: "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," In Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97). Annapolis MD USA (6-10 Jan 1997) 226-235.
15. Yu, E., Cysneiros, L. M.: "Designing for Privacy and Other Competing Requirements," In Second Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, NC USA (15-16 Oct 2002).
16. Yu, E., Liu, L.: "Modelling Trust for System Design Using the i* Strategic Actors Framework," In *Trust in Cyber-societies, Integrating the Human and Artificial Perspectives*, R. Falcone, M. P. Singh, Y.-H. Tan, eds. Springer-Verlag Heidelberg (2001) 175-194.