A condensed version of this technical report with the same title will appear in Proceedings of the Second International Conference on Trust Management, St. Anne's College, Oxford. A revised and extended version of this report has been submitted for publication.

Technical Report No: 2003/19

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

Charles B. Haley
Robin C. Laney
Jonathan D. Moffett
Bashar Nuseibeh

5th December 2003

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

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. Assumptions made during analysis of the requirements for a system-to-be about the trustworthiness of its various components (including human components) can have a significant effect on the specifications derived from the system's requirements. These *trust assumptions* can affect the scope of the analysis, derivation of security requirements, and in some cases how functionality is realized. This paper presents trust assumptions in the context of analysis of security requirements. A running 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

A requirements engineer is concerned with determining the *characteristics* of a *system-to-be*, and how well these characteristics fit with the desires of the *stakeholders*. The *system-to-be* comprises not only software, but also all the diverse components needed for it to achieve its purpose. For example, a computing system clearly includes the computers, but in addition incorporates the people who will use, maintain, and depend on the system; the environment the system is to exist within; and any systems, computer-based and otherwise, already in place. *Stakeholders* are those entities (e.g. people, companies, governments) that have some reason to care about the *characteristics* of the system-to-be. The characteristics of a system include what the system is to do and how it is to appear in order to satisfy the stakeholders at some agreed-upon level. A description of these characteristics is the system-to-be's *requirements*.

An important component of a system's requirements is its *security requirements*. Security requirements arise because stakeholders assert that some objects, be they tangible (e.g. cash) or intangible (e.g. information), have direct or indirect value. Objects valued in this way are called *assets*, and the stakeholders naturally wish to protect the value of these assets. Tangible assets can be destroyed, stolen, or modified. Information assets can be destroyed, revealed, or modified. Either asset can be used to cause indirect harm, such as to reputation. Security requirements are intended to restrict the number of cases wherein these undesirable outcomes can take place.

Security requirements are unusual in that their derivation requires postulation of the existence of an *attacker*. The goal of an attacker is to cause *harm*. Leaving aside the possibility of harm caused by accident or error, 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*. More precisely, a threat is the potential for abuse of an asset that will cause harm in the context of the system. 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. Recent work has taken this approach, looking at the requirements and goals of the attacker (e.g. [1, 3, 16, 18, 19]). From this point of view, an attacker wants a system to have characteristics that create vulnerabilities. The requirements engineer wants to ensure that the attacker's requirements are not met. One way to do this is to specify sufficient *constraints* on the behavior of a system to ensure that vulnerabilities are kept to an acceptable minimum [20]. Security requirements provide these constraints.

In an ideal world, a requirements engineer would reason about a system's characteristics in the absence of a particular implementation of the system. Under this view, requirements engineering is concerned with enumerating goals for a system under consideration and producing a description of the system's desired behavior [15]. On the other hand, a system is intended to solve a given *problem* in a given *context*. Jackson's *problem frames* [13] analyze the problem in terms of the *context* and the design decisions the context represents. The context contains *domains*, which are the blocks that the *system* (not just software) will be built with and around.

Security requirements require a system-level analysis. Without knowing more about the components of a system, the requirements engineer is limited to general statements of the form *only X is allowed to do Y*. Nothing can be said about how this limit is enforced, or even if it is feasible. To determine security requirements, one must look deeper; in a problem frames analysis, this means looking at and describing the behavior of domains within the context of the system. Descriptions of the *desired behavior* of domains are *optative*. Descriptions of the *actual behavior* of existing domains (their inputs, outputs, and states visible at their interfaces) are *indicative*; they describe an objective truth about the behavior of the domains.

Unfortunately, as in all things human, there are assumptions behind the meaning of *objectively true*. One could say "pushing the button completes the electrical circuit", which is true unless someone has cut the wire to the button. Changing the indicative behavior of a domain could create a vulnerability. While reasoning about security, a requirements engineer must make decisions about how much to trust the supplied indicative 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 [23, 24]. Trust assumptions can affect how a system functions, which domains are in the system and therefore what 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 presents work combining trust assumptions, problem frames, and threat descriptions, to show how the combination aids in elaboration and derivation of security requirements. It expands upon the ideas in [7], and is complemented by [8], which presents derivation of security requirements from threat descriptions. Section 2 provides some 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. Domains are either tangible (e.g. people, equipment, networks) or not (e.g. information). The problem frames notation [13] is useful for diagramming the domains involved in a problem and the interconnections between them, and for analyzing the behavior of these domains within the problem's context. For example, assume that talking to stakeholders produces a requirement "open door when the door-open button is pushed." Figure 1 illustrates some domains that satisfy the requirement; a basic automatic door system with three domains. 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; this domain includes 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 subproblem is to satisfy; by definition the requirement is optative. The dashed arrow from the requirement 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 phenomena of designed domains (domains to be built as part of the solution) are optative; one hopes to observe the phenomena in the future.

To illustrate the idea of phenomena, consider the person+button domain in Figure 1. The domain above 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.

Phenomena are normally shown on a diagram on the interface between two domains. The format is X!Y, where X is an abbreviation of the name of the source domain and Y is some label describing the phenomenon. Using the above example again, the ButtonDown event might be shown as PB!ButtonDown.

The interplay of phenomena between the domains defines *how* the system accomplishes the goal. This interplay is a *specification*, describing how the *requirements* are satisfied [28]. The difference between specification and requirement is important. A specification is an expression of the behavior of phenomena visible at the boundary of the domains, whereas a requirement is a description of the problem to be solved. For example, in the context of a building we might find the requirements 'permit passage from one room to another' and 'physically separate rooms when

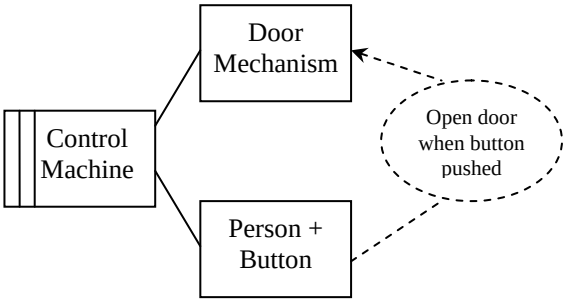

Figure 1 – A basic problem frames diagram

possible'. Clearly the problems involve something like doors. Equally as clearly, they do not specify that doors be used, nor do they specify any phenomena or behavior. It is up to the designer (the architect in this case) to choose the 'door' domain(s) for the system. One might satisfy the requirements with a blanket, an automatic door, a futuristic iris, or a garden maze. Each domain implementation presents different phenomena at its boundary (i.e. they work differently), and the resulting system specification must consider these differences. However, the requirements do not change.

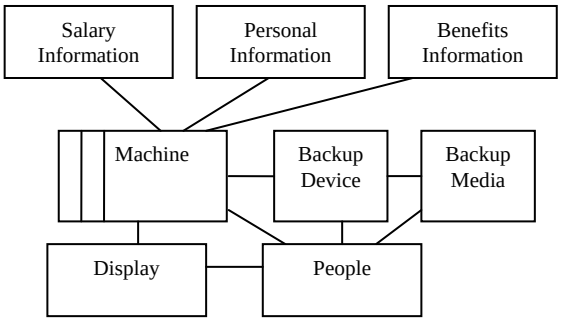

Figure 2 – Example Context Diagram

There are two fundamental diagram types in a problem frames analysis, the *context diagram* and a set of *problem frame diagrams*. The context diagram shows all the domains in a system, and how they are interconnected. Each problem frame diagram examines a *subproblem* in the system, showing how a given requirement (problem) is to be satisfied. In trivial systems with only one requirement, the context diagram and the problem frame diagram are almost identical. For most real 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 subproblem.

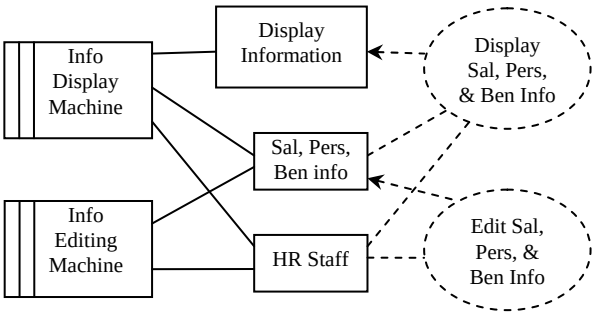

Figure 3 – Payroll data

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 having four functional requirements:

- 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*.
- Each employee shall be able to view a subset of his or her own personal and benefits 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.
- At most 24 hours of modifications to information shall be vulnerable to loss.

This set of requirements would be broken down into four subproblems, one for each requirement. In the interest of brevity, only two of the subproblems are discussed in this paper. Figure 3 shows the subproblem for the first requirement (payroll information), and Figure 4 shows the subproblem for the third requirement (the 'address list' function). Phenomena have been intentionally omitted. The principal differences between the two subproblems are the requirement (what is written in the dotted ovals), the information to be displayed, and the type of person allowed to use the system. In Figure 3, one sees that HR staff can access and change data, while in Figure 4, anyone can access address information. There is no information on the diagrams showing how these access differences will be enforced. These are security requirements, and will be added in the next section.

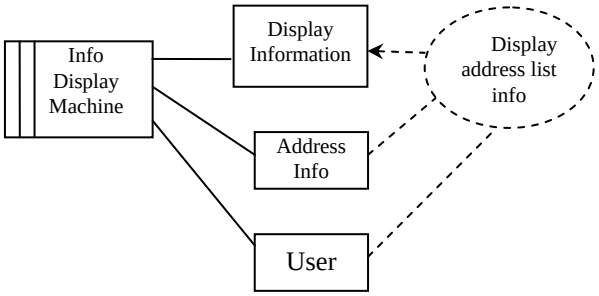

Figure 4 –Address list

3 Security Requirements

Security requirements are generally classified as non-functional requirements (e.g. in [2, 14]), which "can be defined as restrictions or constraints placed on system services" [14]. We are using a slight restatement of this definition: security requirements express constraints on the behavior of a system. The constraints are intended to limit system behavior as much as possible while still satisfying the requirements. For example, a goal for an ATM might be *provide cash to customers*. This goal is obviously overly broad from a security point of view. By providing constraints (security requirements), the circumstances under which cash is to be provided are reduced.

Security requirements come into existence to prevent harm by attacks on assets [8, 20]. An asset is something in the context of the system, tangible or not, that is to be protected [12]. 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, intended to reduce the scope of vulnerabilities. Thus, security requirements stipulate the elimination of vulnerabilities that an attacker can exploit to carry out threats on assets.

The security community provides general categories for constraints, labeling them using the acronym CIA, and more recently another A [21]:

- Confidentiality: ensure that an asset is visible only to those actors authorized to see it. This is larger than 'read access to a file'. It can include visibility of a data stream on a network or visibility of a paper on someone's desk.
- Integrity: ensure that the asset is not corrupted. As above, integrity is larger than 'write access to a file', including triggering transactions that should not occur, changing contents of backup media, making incorrect entries in a paper-based accounting system, or changing a data stream between its source and its sink.
- Availability: ensure that the asset is readily accessible to actors that need it. A counterexample is preventing a company from doing business by denying it access to something important, such as access to its computer systems or its offices.
- Authentication: ensure that the provenance of the asset or actor is known. A common example is the simple login. More complicated examples include mutual authentication (e.g. exchange of cryptography keys), and intellectual property rights management.

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* [8]. Referring to the example presented above, some possible threat descriptions are:

- Exposing salary data could reduce employee morale, lowering productivity.
- 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 to see if the asset involved in the threat is found in the subproblem. To be in a subproblem, the asset must be either a domain or part of a domain. If the asset is found in the subproblem, then the requirements engineer must apply constraints on the subproblem 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, and indicated by adding an inversion of the threat description as an annotation to the requirement, as in *prevent exposure of salary data* or, in the context of the requirements, *only HR staff*. The security requirements are then satisfied by changes and/or additions to the phenomena which change the behavior of the domains.

Without going into the mechanics of how the security requirements were determined (see [8]), analysis of Figure 3 showed that in order to maintain confidentiality and integrity of the data, the network needs to be protected and HR staff needs to be authenticated. A design decision is made to use encryption on the network and a login/password scheme to authenticate HR staff. The resulting problem frame diagram is shown in Figure 5. The security requirement has been added to the oval. Phenomena have been added to support authentication and encryption, and the encrypted network has been made explicit.

Figure 6 shows the address list subproblem with the appropriate constraints added. However, although phenomena have been added to protect against snooping on the network, nothing has been done yet about the *user* domain. The security requirement has not been satisfied. We will solve this problem in the next section.

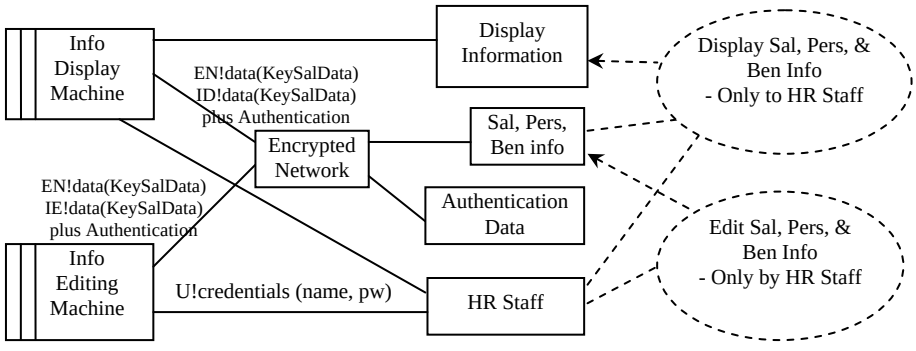

Figure 5 – Payroll data revisited

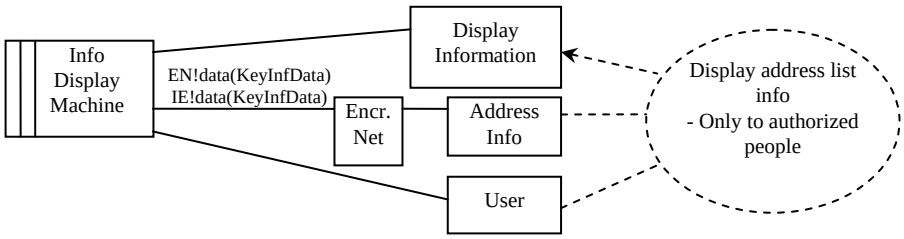

Figure 6 –Address list revisited

4 Trust Assumptions

Recall that a requirement describes what a system is to accomplish. From the point of view of the requirements engineer, how a requirement is satisfied depends on the characteristics of the domains in the problem. An analogous relationship exists between security requirements and trust assumptions; how security requirements are satisfied depends on the trust assumptions made by the requirements engineer.

Trust assumptions are endemic in software and systems development. Viega and McGraw put it very well in [24]:

> *A trust relationship is a relationship involving multiple entities (such as companies, people, or software components). Entities in a relationship trust each other to have or not to have certain properties (the so-called **trust assumptions**). If the trusted entities satisfy these properties, then they are **trustworthy**. Unfortunately, because these properties are seldom explicitly defined, misguided trust relationships in software applications are not uncommon.*
>
> *Software developers have trust relationships during every stage of software development. Before a software project is conceived, there are business and personal trust relationships that developers generally assume will not be abused. For example, many corporations trust that their employees will not attack the information systems of the company.*

We use the definition of trust proposed by Grandison & Sloman [6]: "[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 subproblem.

A trust assumption is an acceptance by a requirements engineer that the specification of a domain can depend on certain stated properties of some other domain up to some stated level, in order to satisfy a security requirement. The requirements engineer *trusts* the *assumption* to be true. The trust assumption results in a relation between the dependent domain (the *trustor*) and the depended-upon domain (the *trustee*), indicating that the trustor depends upon some properties or assertions of the trustee. These properties or assertions act as *domain restrictions;* they restrict the dependent domain in some way. A trust assumption is represented by an arc from the dependent domain to an oval that names the depended-upon domain and properties being depended upon.

Adding a trust assumption serves two purposes. The first and most important is to explicitly limit the scope of the analysis to the domains in the context. The second is to document the ways in which 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 can satisfy this requirement by adding backup generators to the system. Appropriate phenomena would be added to detect the power loss, control the generators, detect going beyond eight hours, etc. In most cases, the requirements engineer can trust the manufacturer of the generators to supply equipment without trapdoors that permit an attacker to take control of the generators. This assumption would be represented by an arc from the generator domain to an oval representing the supplier domain. The domain is thereby restricted to contain generators without trapdoors. 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 find that trust assumptions must be added to each diagram in order 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.

Figure 7 presents the trust assumptions made during analysis of the payroll data problem. The requirements engineer was told by the IT organization that the encryption keys being built into the system are secure because a revocable certificate key exchange system is in use. Furthermore, the certificate tells the data server the access level of the client machine; the server refuses to supply information above the access level indicated by the certificate. Choosing to accept these explanations, the requirements engineer adds two trust assumptions to the

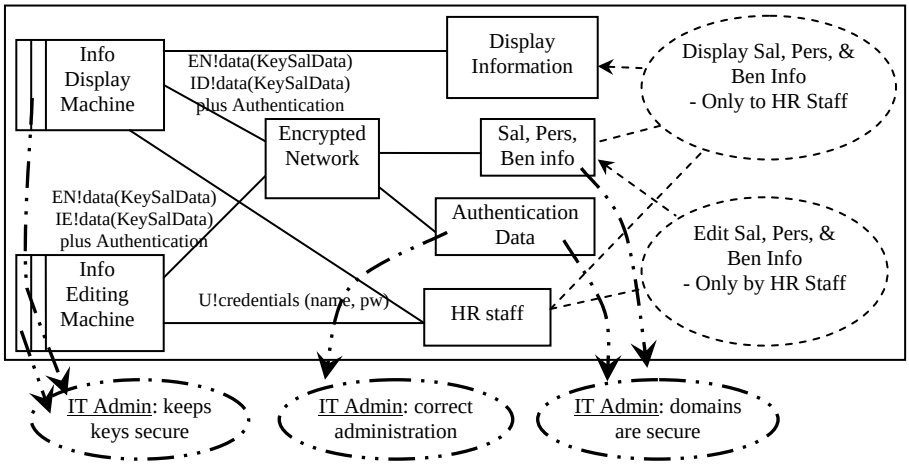

Figure 7 – Payroll data revisited again

problem frame diagram. Finally, the requirements engineer chooses to trust the systems administrators to properly allocate and control the access credentials given to HR department members.

The address subproblem presents the requirements engineer with a more interesting challenge. There are threats against the name and address information which indicate that confidentiality of the information must be maintained. To achieve this, the requirements engineer proposes that the information be limited to people having authentication information and able to log in, thereby proposing that all employees be given authentication information. The IT department refuses to agree, saying that giving all employees authentication information would be too costly. Furthermore, the stakeholders insist that requiring a login would make the system too hard to use. The security officer is equally adamant; the information must be protected somehow. Further questioning leads the requirements engineer to note 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 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 8 shows the resulting problem frames diagram.

The above examples support our position that trust assumptions are domain restrictions. The clearest example is the security system trust assumption; it restricts the membership of the *people* domain to people acceptable to the door guard, effectively converting the domain to *employees*. The other trust assumptions play a similar role. The *IT Admin: correct administration* trust assumption limits the number of people having acceptable credentials. The *IT Admin: … secure* trust assumptions limit the knowledge of the credentials to the machines.

The *IT Admin: keys restrict access* trust assumption is a special case. The domain that it is limiting 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

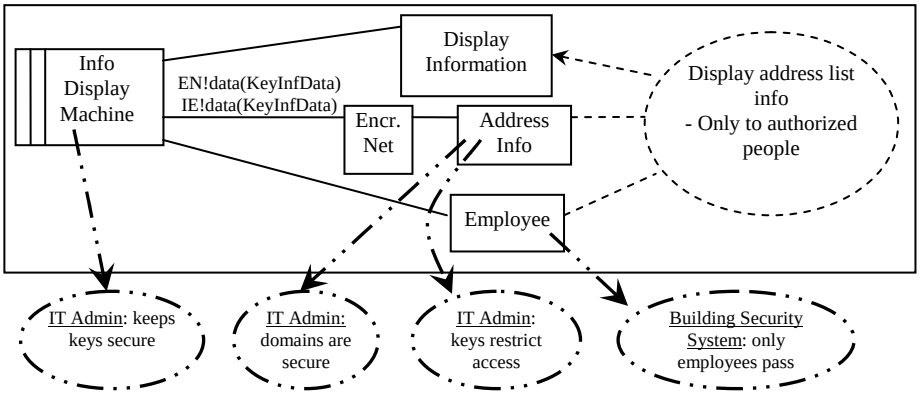

Figure 8 – Address list revisited again

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 [25, 27], 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 [5] uses the *i** framework, adding wider lifecycle coverage. Gans et al [4] add distrust and "speech acts". Yu and Cysneiros have added privacy to the mix [26]. All of these models are concerned with analyzing trust relations between actors/agents in the running system, as opposed to capturing the requirements engineer's assumptions. 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 [9] 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. Context is determined using "purpose" (why is information being accessed), "conditions" (what conditions must be satisfied before access can be granted), and "obligations" (what actions must be taken before access can be granted). The framework, like *i**, describes run-time properties, not the requirements engineer's assumptions about the domains forming the solution.

Another related body of work focuses on security requirements without special emphasis on trust, either in the completed system (as above) or during development (as in this work). van Lamsweerde et al use "obstacles" to analyze security & safety [17] in KAOS, and are developing the notion of anti-goals to discover and close vulnerabilities [16]. Alexander is looking at detecting vulnerabilities using misuse cases [1], as is Sindre et al [22]. McDermott uses 'abuse cases' [19]. Heitmeyer has added security requirements to SCR [10], as have In and Boehm with the WinWin framework [11]. None of the work incorporates explicit capture of how a requirements engineer uses trust when specifying a system.

6 Conclusions and Future Work

We have provided 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 [8]. The trust assumptions will play a critical role in analyzing cost and risk. The *quantification* of the level of trust will be used in this context.

Another future focus will be a tighter coupling of trust assumptions and problem frames. The domains in a subproblem diagram are a projection of the context. The projection can combine domains into single entities, or it can split a domain into its component parts. Having such projections raises the question "to what, exactly, is the trust assumption connected?" The question is important because the trust assumption has an impact on the membership and phenomena of the projected domain, and we must determine how these impacts affect other subproblems that reference [parts of] the projected domain.

Acknowledgements: The financial support of the Leverhulme Trust is gratefully acknowledged. Thanks also go to Michael Jackson for many insights about problem frames and requirements.

References:

1. Alexander, I.: "Initial Industrial Experience of Misuse Cases in Trade-Off Analysis," In Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02). Essen Germany (2002) 61-68.
2. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
3. 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.
4. 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.
5. Giorgini, P., Massacci, F., Mylopoulos, J.: *Requirement Engineering Meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard*, Department of Information and Communication Technology. University of Trento, DIT-03-027 (May 2003).
6. 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).

7. Haley, C. B., Laney, R. C., Moffett, J. D., Nuseibeh, B.: "Using Trust Assumptions in Security Requirements Engineering," Second Internal iTrust Workshop On Trust Management In Dynamic Open Systems, Imperial College, London UK (15-17 Sep 2003).
8. Haley, C. B., Laney, R. C., Nuseibeh, B.: *Deriving Security Requirements from Crosscutting Threat Descriptions*, Department of Computing. The Open University UK, Technical Report 2003/11 (October 2003).
9. 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).
10. Heitmeyer, C. L.: "Applying 'Practical' Formal Methods to the Specification and Analysis of Security Properties," In Proceedings of the International Workshop on Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Computer Security (MMM ACNS 2001). St. Petersburg, Russia: Springer-Verlag Heidelberg (21-23 May 2001) 84-89.
11. In, H., Boehm, B. W.: "Using WinWin Quality Requirements Management Tools: A Case Study," *Annals of Software Engineering (Kluwer)* Vol. 11. no. 1 (Nov 2001) 141-174.
12. 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).
13. Jackson, M.: *Problem Frames*. Addison Wesley, 2001.
14. Kotonya, G., Sommerville, I.: *Requirements Engineering: Processes and Techniques*. United Kingdom: John Wiley & Sons, 1998.
15. van Lamsweerde, A.: "Goal-Oriented Requirements Engineering: A Guided Tour," In 5th IEEE International Symposium on Requirements Engineering (RE'01). Toronto, Canada: IEEE Computer Society Press (27-31 Aug 2001) 249-263.
16. 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).
17. van Lamsweerde, A., Letier, E.: "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering* Vol. 26. no. 10 (Oct 2000) 978-1005.
18. 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.
19. McDermott, J.: "Abuse-Case-Based Assurance Arguments," In Proceedings of the 17th Computer Security Applications Conference (ACSAC'01). New Orleans LA USA: IEEE Computer Society Press (10-14 Dec 2001) 366-374.
20. Moffett, J. D., Nuseibeh, B.: *A Framework for Security Requirements Engineering*, Department of Computer Science. University of York, UK, YCS368 (August 2003).
21. Pfleeger, C. P., Pfleeger, S. L.: *Security in Computing*. Prentice Hall, 2002.
22. Sindre, G., Opdahl, A. L.: "Eliciting Security Requirements by Misuse Cases," In Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Pacific'00). Sydney Australia (20-23 Nov 2000) 120-131.
23. Viega, J., Kohno, T., Potter, B.: "Trust (and Mistrust) in Secure Applications," *Communications of the ACM* Vol. 44. no. 2 (Feb 2001) 31-36.
24. Viega, J., McGraw, G.: *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley, 2002.
25. 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.
26. 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).
27. Yu, E., Liu, L.: "Modelling Trust for System Design Using the i* Strategic Actors Framework," In *Trust in Cybersocieties, Integrating the Human and Artificial Perspectives*, R. Falcone, M. P. Singh, Y.-H. Tan, eds. Springer-Verlag Heidelberg (2001) 175-194.
28. Zave, P., Jackson, M.: "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology* Vol. 6. no. 1 (Jan 1997) 1-30.