The Effect 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.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 are frequently made during requirements analysis of a system-to-be about the trustworthiness of its various components (including human components). 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 can be used by a requirements engineer to help define and limit the scope of analysis and to document the decisions made during the process. The paper concludes with a case study examining the impact of trust assumptions on software that uses the Secure Electronic Transaction (SET) specification.

1. Introduction

Requirements engineering is concerned with determining the *characteristics* of a *system-to-be*. 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 also incorporates the people who will use, maintain, and depend on the system; the environment within which the system will exist; and any systems already in place.

An important element 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 and state), have direct or indirect value. Objects valued in this way are called *assets*, and the stakeholders naturally wish to protect these assets from *harm*. For example, tangible assets might be destroyed, stolen, or modified; information assets might be destroyed, revealed, or modified; and state might be modified, revealed, or disputed (this list is not exhaustive). An asset can be used to cause indirect harm, such as to reputation. The requirements engineer uses security requirements to restrict the number of cases wherein these undesirable outcomes can take place. This paper presents how the engineer's derivation, elaboration and analysis of security requirements can be aided through the use of *trust assumptions*, *problem frames*, and *threat descriptions*.

Although not required, derivation of security requirements can be facilitated by the postulation of the existence of an *attacker*. The attacker's goal is to cause *harm*. Ignoring the possibility of harm caused by accident or error, if one can show that no attackers exist, then security is irrelevant. An attacker causes harm by exploiting an asset in some way. The possibility of such an exploitation is called a *threat*. More precisely, a threat is the potential for abuse of an asset in the context of the system that will cause harm. An *attack* exploits a *vulnerability* in the system to carry out a threat.

One can 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, 2, 15, 17, 18, 24]). 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. A way to do this is to specify sufficient *constraints* on the behavior of a system to ensure that the number of vulnerabilities is kept to an acceptable minimum [19]. Security requirements provide these constraints.

One school of thought holds that a requirements engineer should reason about a system's characteristics in the absence of a particular implementation of the system (e.g. [14]). 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. Another view, exemplified by problem frames [12], is that a system is intended to solve a given *problem* in a given *context*, where the context includes design decisions. One uses problem frames to analyze the problem in terms of the *context* and the design decisions

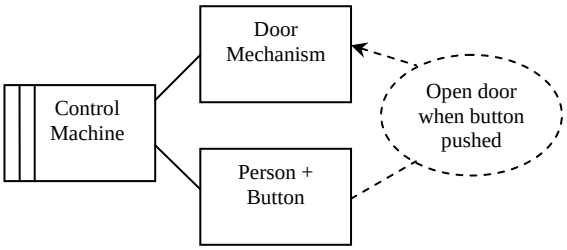

Figure 1 – A basic problem frames diagram

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 demand a system-level analysis [19]. Without knowing more about the components of a system, the requirements engineer is limited to general goals of the form *X must not occur*. Nothing can be said about how such goals are enforced, or even if they are feasible. To determine security requirements, one must look deeper in the system. When using problem frames, one analyzes the behavior of domains within the context of the system to analyze the effects of the security requirements and to show how they are satisfied.

Threat descriptions [7] are useful for reasoning about security requirements in a problem frames environment. Threat descriptions assist with locating potential vulnerabilities, which are closed either by modifying the context or adding *trust assumptions*. A trust assumption is a decision about how much to trust the supplied indicative (objectively true) properties of domains that make up the system and evaluate the risks associated with being wrong. Trust assumptions can have a fundamental impact on how the system is realized [25, 26]. They can affect which domains are included in the analysis, the risk that vulnerabilities exist, and the risk that a system design is not stable or correct. During analysis, trust assumptions permit the requirements engineer to decide which domains need further analysis and which do not.

This paper presents work combining trust assumptions, problem frames, and threat descriptions, showing how the combination aids in derivation, elaboration and analysis of security requirements. The paper is complemented by [7], which describes analyzing security requirements using 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 is a case study, Section 6 presents related work, and section 7 concludes.

2. Problem Frames

In the problem frames universe, all computing problems involve the interaction of domains in the world. Domains are either tangible (e.g. people, equipment, networks) or intangible (e.g. information). The problem frames notation [12] 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.

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.

For example, assume that talking to stakeholders produces a requirement "open the door when the door-open button is pushed." Figure 1 is a problem frames diagram of a solution to satisfy the requirement; an automatic door system composed of 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 designed to fulfill the requirement that the door open when the button is pushed. The dashed-line oval presents the requirement that the problem must satisfy; by definition the requirement is optative. The dashed arrow from the requirement oval indicates which domains are to be constrained by the requirement.

To illustrate the idea of phenomena, consider the person+button (PB) 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.

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. In the example, the ButtonDown event might be shown as PB!ButtonDown.

The interplay of phenomena among the domains defines *how* the system accomplishes the goal. This interplay is a *specification*, describing how the *requirements* are satisfied [30]. 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 possible'. Clearly the problem involves something like doors. Equally as clearly, it does not specify that doors be used, nor does it specify internal 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 requirement with a

blanket, an automatic door, a futuristic iris, or a garden maze. Each domain implementation presents different phenomena at its boundaries (i.e. they work differently), and the resulting system specification must consider these differences. However, the requirement does not change.

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 *problem* in the system, showing how a given requirement is to be satisfied. In small systems, the context diagram and the single problem frame diagram are almost identical and may be combined. For larger systems, the domains in the collection of problem 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 in sections 3 and 4 of this paper. The system is a subset of a Human Resources system having four 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 could be broken down into four subproblems, one for each requirement. In the interest of brevity, only one of the subproblems, the one for the third requirement, is discussed in this paper. Figure 3 shows the problem diagram for this requirement (the 'address list' function). Phenomena have been intentionally omitted. Security requirements will be added in the next section.

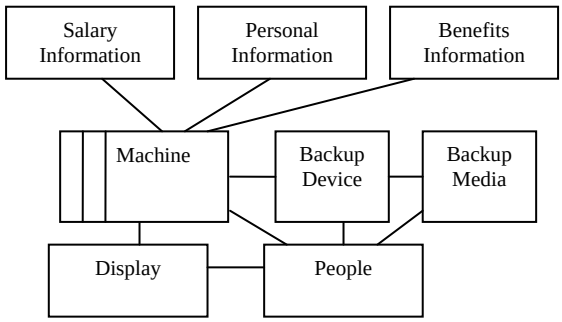

Figure 2 – Example context diagram

3. Security Requirements

Security requirements are often defined as "restrictions or constraints placed on system services" [13]. We slightly restate 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 are added to prevent harm through misuse of assets [7, 19]. An asset is something in the context of the system, tangible or not, that is to be protected [11]. 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 to realize a threat. Security requirements are constraints on functional requirements, intended to reduce the scope of vulnerabilities. Thus, security requirements stipulate the location and elimination of vulnerabilities that an attacker can exploit to carry out threats on assets.

The security community has enumerated some general security goals, labeling them using the acronym CIA, and more recently another A [20]:

- Confidentiality: ensure that an asset is visible only to actors authorized to see it. This is larger than 'read access to a file'. It includes visibility of a data stream on a network or 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 agents 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

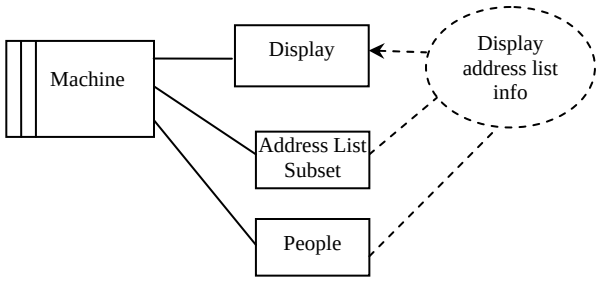

Figure 3 –Address list

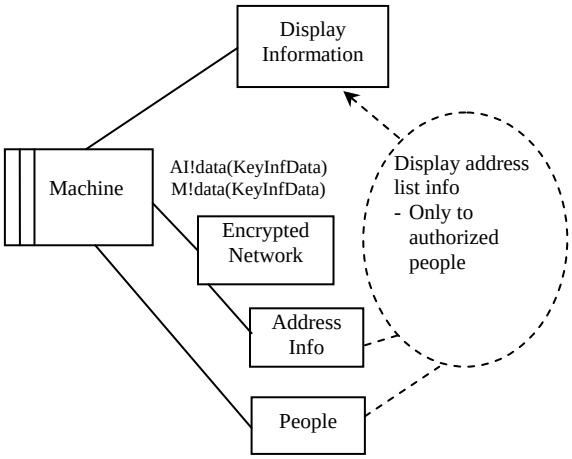

Figure 4 –Address list revisited

login. More complicated examples include mutual authentication (e.g. exchange of cryptography keys), and intellectual property rights management.

By operationalizing these goals (connecting them with specific assets), then inverting the sense of the goals, one can construct descriptions of possible threats on assets. These *threat descriptions* are phrases of the form *performing action X on/to/with asset Y could cause harm Z* [7]. 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 threat descriptions, the requirements engineer examines each (sub)problem diagram to see if the asset involved in the threat is found in the problem. To be in a problem, the asset must be either a domain or part of a domain, or be found in the phenomena. If the asset is found in the problem, 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, and are indicated by adding an inversion of the threat description as an annotation to the requirement, as in *prevent exposure of salary data* or *only by HR staff*. The security requirements are then satisfied by changes and/or additions to the domains or phenomena, changing the behavior of the domains in the context.

Without going into the mechanics of how the security requirements are determined (see [7]), analysis of Figure 3 shows that in order to maintain confidentiality and integrity of the data, the network needs to be protected and users must be authorized. A design decision is made to use encryption on the network. The resulting problem frame diagram is shown in Figure 4. The security requirement has been added to the oval. Phenomena have been added to support encryption, and the encrypted network has been made explicit. How users are authorized is discussed in the next section.

4. Trust Assumptions

When analyzing using problem frames, 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 [26]:

> *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.*

We use the definition of trust proposed by Grandison & Sloman [5]: "[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 some domain to participate 'competently and honestly' in the satisfaction of a security requirement in the context of the problem.

A trust assumption is an acceptance by a requirements engineer that the membership or specification of a domain can depend on certain stated properties, up to some stated level, in order to satisfy a security requirement.[1] The requirements engineer *trusts* the *assumption* to be true. These assumed 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 describing the properties being depended upon.

Adding a trust assumption serves two purposes. The first is to document the ways in which the requirements engineer chooses to trust the behavior of domains that are in the context for some reason. The second, which follows from the first, is to explicitly limit the scope of the analysis to these domains in the context. To illustrate the latter, assume the existence of a requirement stipulating that the computers operate for at least eight hours in the event of a power failure. The requirements engineer can

[1] The "stated level" is a measure of the "quantified belief" in the definition of trust. At the moment, our quantification is binary. In future work, the quantification will be over a finer scale.

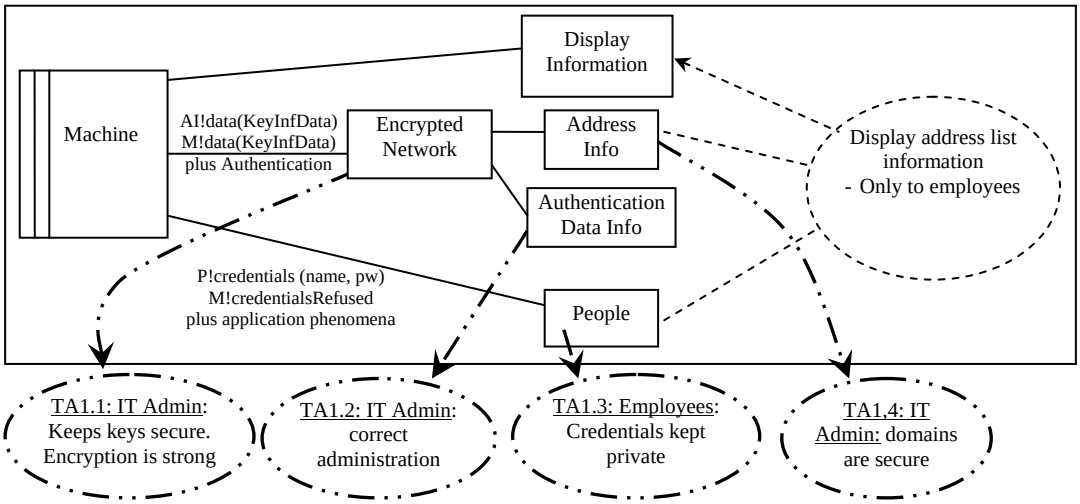

Figure 5 – Address list with authentication

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 situations, 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. thereby restricting the domain 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.

4.1. Example: Using Authentication

Returning to our example, trust assumptions must be added to the diagram in order to complete the picture. For example, the analysis does not explain why the encrypted network is considered secure or how address information is to be protected. We must also determine that the users are authorized to see the information, perhaps using authentication.

Authentication is a system-level problem involving many potentially complex processes. In order to derive requirements for authentication, the requirements engineer must choose how users are authenticated, perhaps based on cost, risk, and ease-of-use factors.

Figure 5 presents a login and password solution, along with some accompanying trust assumptions. The intent is that only those with login credentials can access the information. The requirements engineer is convinced by the IT organization that the encryption system is strong and that the keys being built into the system are secure; the encrypted network connection domain does not require further analysis. Furthermore, the key system tells the data server the access level of the client machine; the behavior of the server is constrained to refuse to supply information above the access level indicated by the keys. Accepting these explanations, the requirements engineer adds trust assumptions TA1.1 and TA1.4 to the problem frame diagram. TA1.2 indicates that the requirements engineer chooses to trust the systems administrators to properly manage access credentials, constraining the domain to contain accurate information. Finally, the engineer assumes that employees will keep their credentials confidential (TA1.3), constraining the 'people' domain to be 'people with their own credentials'.

4.2. Example: Using Building Security

The login/password scheme may be unacceptable to the customer. The IT department may refuse, saying that giving all employees authentication information would be too costly. The stakeholders may refuse, insisting that requiring a login would make the system too hard to use. An alternate solution is to make use of the fact 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.

Figure 6 presents this alternate solution. Authentication is removed and trust assumption TA2.2 is added, having the effect of changing the *People* domain to *Employees* by restricting membership to *people allowed to enter the building by the security system*. The authentication-related trust assumptions are removed. It is up to the customer to decide if the associated risk profile is acceptable.

4.3. Trust Assumptions as Domain Restrictions

The above examples support our position that trust assumptions are domain restrictions. The clearest example is the security system trust assumption (TA2.2 in Figure 6); it restricts the membership of the *People* domain to

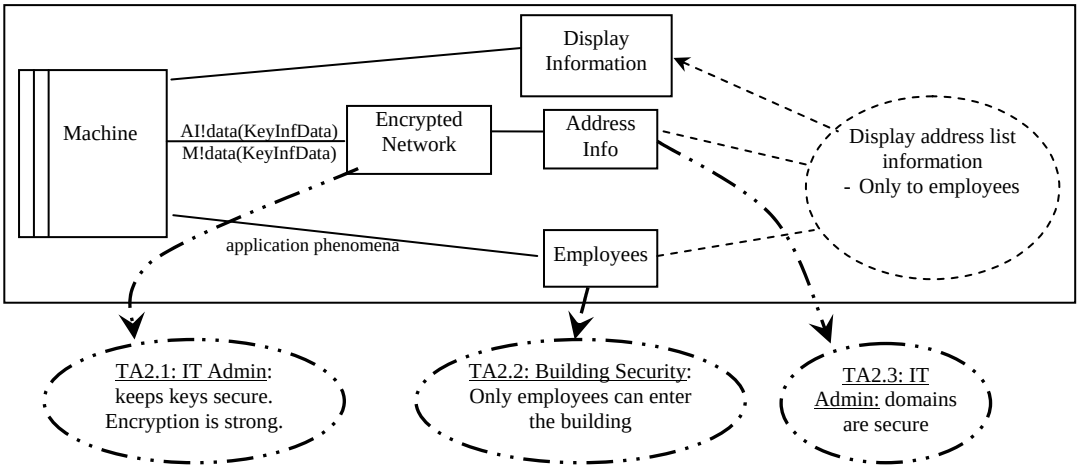

Figure 6 – Address list with door security

people acceptable to the door guard, effectively converting the domain to *employees*. The other trust assumptions play a similar role. For example, TA1.2 (*IT Admin: correct administration*) trust assumption limits the number of people having acceptable credentials.

TA2.1 and TA2.3 (*IT Admin: domains are secure* and the *IT Admin: keeps keys secure)* illustrate restricting behavior (specification) as opposed to membership. In the case of TA2.3, the behavior of the Address Info domain is restricted to supply information only at the level indicated by the key; the assertion is that no other case exists. In the case of TA2.1, the domain is restricted to supplying 'in the clear' information to holders of valid encryption keys; the assertion is that no alternate method to obtain the information exists.

The authorization solutions presented in sections 4.1 and 4.2 clearly have different risk profiles. A future project, adding non-binary *quantification* of the trust level in the trust assumption, will help measure the differences and choose which profile/design is the most acceptable.

5. Case Study

The Secure Electronic Transaction (SET) Specification [21-23] describe a set of mechanisms intended to provide an acceptable level of security for on-line purchasing. This case study looks at incorporating the SET specification into software to support cardholder-side payment authorization. There is one requirement (in the problem frames sense): Complete the Purchase. The study considers one asset, *Customer Account Information (CAI)*, and one derived security goal *Purchases shall be authorized*. Several trust assumptions are derived during the analysis.

To derive the trust assumptions, we first determine the threat descriptions, then negate them to express the security requirements (the constraints). Two threat descriptions are used in this case study: *exposure of cardholder account information could lead to financial loss* (from the goal of confidentiality), and *unauthorized use of cardholder credentials could lead to financial loss* (from the goal of integrity). Appropriate security requirements (constraints) are added to the requirements: *only authorized users may know CAI* and *only authorized individuals may use the cardholder credentials*. The resulting trust assumptions will be listed in a later section.

5.1. SET Overview

SET describes a series of operations between players in an electronic purchase transaction using a credit card. In SET, a *cardholder* requests a cryptographic certificate from a *certificate authority* (CA). The CA verifies that the cardholder has a credit card account with an *issuer*, and then supplies a certificate. The cardholder can subsequently use the certificate to make purchases from a *merchant*. The merchant uses a *payment gateway* to pass the transaction to the *acquirer* (the merchant's bank) for collection. The acquirer normally operates the payment gateway. Figure 7 presents a simplified version of the SET "processing flows" (terminology from [21]), showing the players and the messages they interchange. Several SET messages and fields that do not have a direct bearing on this discussion have been omitted from the diagram, in particular the obtaining of certificates and private keys, and the initial verification of cardholder information. In addition, the diagram shows the merchant using the CAI, which although optional in SET is the technique that the SET specification claims will be the most often used. [22: pg. 14]

5.2. SET-Identified Security Assumptions

The SET specifications make the following security-related assumptions relevant to this case study about the SET environment:

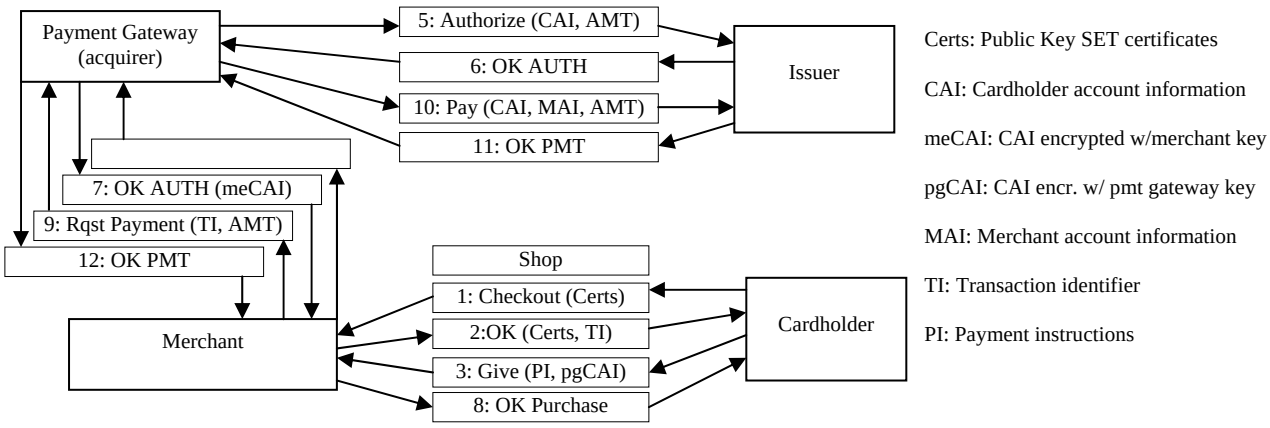

Figure 7 – Simplified SET processing flows

- SA1: The cardholder ensures that no one else has access to his/her private key. [21: pg. 16] In particular, SET software vendors shall "ensure that the certificate and related information is stored in a way to prevent unauthorized access." [21: pg. 46]
- SA2: Cardholder, merchant, and payment gateway machines are free of viruses and trojan horses, and are not susceptible to being hacked. [21: pg. 11]
- SA3: Programming methods and the cryptographic system, and in particular the random number generators, are of the highest quality. [21: pg. 16]
- SA4: The merchant's system stores account information in an encrypted form, and if possible off-line or behind a firewall. [22: pg. 39]

5.3. The Initial Context/Problem Diagram

As there is only one requirement in this case study and therefore only one problem diagram, we are dispensing with separating the context diagram from the problem diagram. In addition, we are not showing any analysis of the 'shopping' process, instead focusing on the point where a purchase is completed. Taking the SET processing flows into consideration, a first-cut problem diagram is shown in Figure 8.

The threat descriptions being used in this study tell us to examine the problem for uses of the information asset *cardholder account information* (CAI), which is made visible by the CAI phenomena in the context diagram, and the asset *cardholder credentials*, stored in the machine.

By tracing the CAI through the problem diagram, we see that it resides in unknown form within the Machine domain. According to the SET specification, the CAI must be encrypted between the machine and the merchant. There is nothing in the diagram that indicates that the user or the merchant can obtain the CAI. We can say the same thing for cardholder credentials. These positions and the security requirements SA1-SA4 lead us to make the following trust assumptions:

- TA1-1: As the credentials are stored on the machine, and as there is no apparent way to limit who can access these credentials, SA1 forces us to assume that the domain *Users* in the problem contains only individuals authorized to use the credentials.
- TA1-2: The storage containing the CAI and the credentials is not readable outside the machine. (SA2)
- TA1-3: The generated symmetric encryption keys are cryptographically secure. (SA3)
- TA1-4: The merchant cannot know the cardholder's private key, and therefore cannot access the CAI that it passes through to the payment gateway. Conflicts with SA4.

The first trust assumption (TA1-1), that the domain *Users* contains only authorized individuals, is clearly very risky. There is no information available to justify the claim. The analyst should add domains and phenomena to the problem to reduce the risk. A similar statement must be made about TA1-2, because nothing is said that allows the engineer to claim that the storage is secure. If the information can be read without supplying a key that is not stored on the machine, then the existence of viruses, spyware, and other programs/users make the trust assumption's claim ludicrous. Vulnerabilities to the threats still exist, and appropriate domains and phenomena must be added to close the vulnerabilities and satisfy the requirement.

Verifying TA1-3 is probably not necessary, assuming

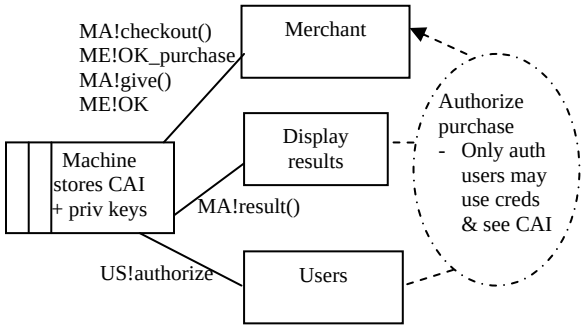

Figure 8 – Purchase problem

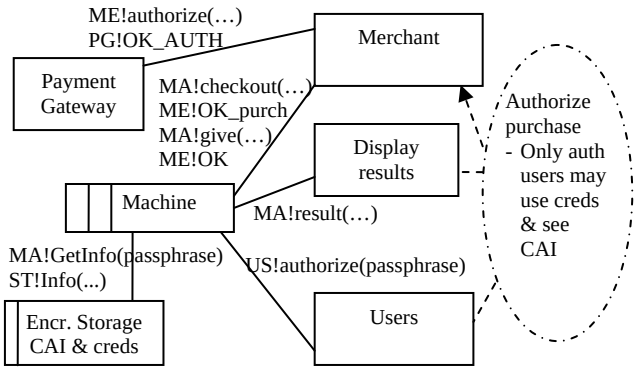

Figure 9 – Purchase problem (again)

that the cryptographic software comes from a company that the requirements engineer believes has verified its applications. If the engineer cannot confirm this belief, then a domain representing the encryption software must be added to the context, and then analyzed appropriately.

TA1-4 serves to limit the scope of the analysis, assuming that nothing on the other side of the merchant can expose CAI to the merchant. Unfortunately, the SET 'processing flows' diagram (step 7) shows that the payment gateway can give the CAI back to the merchant. The trust assumption is invalid and must be removed.

Figure 9 presents the modified problem. Because TA1-1 was rejected, a *passphrase* has been added to verify that the user is authorized. The passphrase is used to encrypt the CAI and certificate storage. In addition, the context has been expanded to include the payment gateway.

The new problem diagram exposes the following trust assumptions:

- TA2-1: Users will not expose the passphrase.
- TA2-2: The merchant implements the SET recommendations and securely stores the CAI. There is no practical way to bypass this security, regardless of storage medium (operational, backup, etc.)
- TA2-3: The merchant's employees authorized to see the CAI will not compromise the information.
- TA2-4: The CAI never appears in the clear on the merchant's internal network.
- The same assumptions that apply to the merchant also apply to the payment gateway.

Figure 10 presents the solution along with the four trust assumptions. To reduce the complexity of the diagram, the phenomena and the trust assumptions applied to the payment gateway are not shown.

The risk presented by TA2-1, that the passphrase will remain confidential, may or may not be acceptable. For example, a French bank decided the risk was too high, and included a smartcard reader in its implementation. The user must both know the passphrase and insert the smartcard into the reader.

The remaining trust assumptions are problematic. There is no practical way for a requirements engineer to examine every merchant and payment gateway company, so the assumptions must be accepted at face value.

The trust assumptions required to fulfill the security requirement might provoke the requirements engineer to reconsider whether a customer-side product based SET is worth constructing. Given that the CAI can be stored on the merchant's machine, the difference between a SET solution and the ubiquitous solution based on SSL (secure sockets layer) is not large. Using SET, it is more difficult for a merchant to change an order, but a dishonest merchant would have no problem creating new non-SET orders charged to the customer. Dishonest merchants and employees could sell the account information. Hackers could steal it. There is nothing the engineer can do to mitigate the problems exposed by these trust assumptions. The customer/stakeholders must decide whether the risks are acceptable.

6. 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 [27, 29], Yu, Lin, & Mylopoulos take an 'actor, intention, goal' approach where security and trust relationships

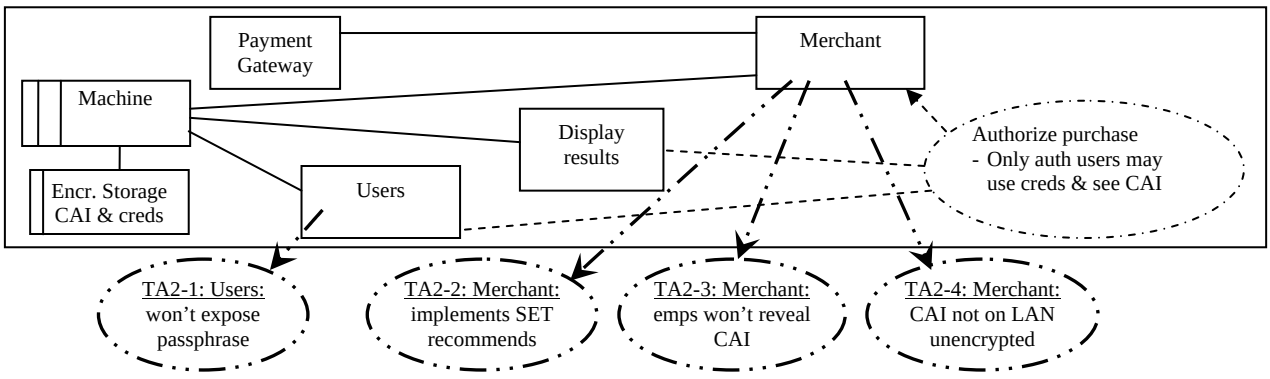

Figure 10 – Purchase problem (again)

within the model are modeled as "softgoals": goals that have no quantitative measure for satisfaction. The Tropos project [4] uses the *i** framework, adding wider lifecycle coverage. Gans et al [3] add distrust and "speech acts". Yu and Cysneiros have added privacy to the mix [28]. 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, a position corroborated by the differences between the analysis in our case study and the *i**-based SET case study found in [4].

He and Antón [8] 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 [16] in KAOS, and are developing the notion of anti-goals to discover and close vulnerabilities [15]. Alexander is looking at detecting vulnerabilities using misuse cases [1], as are Sindre et al [24]. McDermott uses 'abuse cases' [18]. Heitmeyer has added security requirements to SCR [9], as have In and Boehm with the WinWin framework [10]. None of the work incorporates explicit capture of how a requirements engineer uses trust when specifying a system.

7. Conclusions and Future Work

We have provided an approach for using trust assumptions when reasoning about security requirements. The approach uses the strong distinction between system requirements and machine specifications found in problem frames, permitting the requirements engineer to choose how to conform to the requirements. The trust assumptions embedded in the solution inform requirements engineers, better enabling them 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 [7]. In future work, the trust assumptions will play a critical role in analyzing cost and risk. A non-binary *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. When a larger problem is decomposed, the domains in the problem diagrams 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 trust assumptions have impacts on the membership and phenomena of the projected domain, and we must determine how these impacts affect other problems that reference any part 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, and to our colleagues Luncheng Lin and Debra Haley for their careful review of this paper. This paper is a revised and extended version of [6].

References:

[1] I. Alexander, "Modelling the Interplay of Conflicting Goals with Use and Misuse Cases," *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02)*. Essen, Germany, 9-10 Sep 2002, pp. 145-152.

[2] R. Crook, D. Ince, L. Lin, and B. Nuseibeh, "Security Requirements Engineering: When Anti-Requirements Hit the Fan," *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*. Essen Germany, 2002, pp. 203-205.

[3] G. Gans, M. Jarke, S. Kethers, et al., "Requirements Modeling for Organization Networks: A (Dis)Trust-Based Approach," *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01)*. Toronto, Canada, IEEE Computer Society Press, 27-31 Aug 2001, pp. 154-165.

[4] P. Giorgini, F. Massacci, and J. Mylopoulos, "Requirement Engineering Meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard," *Proceedings of the 22nd International Conference on Conceptual Modeling*. Chicago IL USA, Springer-Verlag Heidelberg, 13-16 Oct 2003, pp. 263-276.

[5] T. Grandison and M. Sloman, "Trust Management Tools for Internet Applications," *Proceedings of the The First International Conference on Trust Management*, vol. 2692. Heraklion, Crete, Greece, Springer Verlag, 28-30 May 2003.

[6] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "Picking Battles: The Impact of Trust Assumptions on the Elaboration of Security Requirements," *Proceedings of the Second International Conference on Trust Management*

(iTrust'04), vol. 2995. St Anne's College, Oxford UK, Lecture Notes in Computer Science (Springer-Verlag), 29 Mar - 1 Apr 2004, pp. 347-354.

[7] C.B. Haley, R.C. Laney, and B. Nuseibeh, "Deriving Security Requirements from Crosscutting Threat Descriptions," *Proceedings of the Third International Conference on Aspect-Oriented Software Development (AOSD'04)*, K. Lieberherr, Ed. Lancaster UK, ACM Press, 22-26 Mar 2004, pp. 112-121.

[8] Q. He and A.I. Antón, "A Framework for Modeling Privacy Requirements in Role Engineering," *Proceedings of the 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.

[9] C.L. Heitmeyer, "Applying 'Practical' Formal Methods to the Specification and Analysis of Security Properties," *Proceedings of the International Workshop on Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Computer Security (MMM ACNS 2001)*, vol. 2052. St. Petersburg, Russia, Springer-Verlag Heidelberg, 21-23 May 2001, pp. 84-89.

[10] H. In and B.W. Boehm, "Using WinWin Quality Requirements Management Tools: A case study," *Annals of Software Engineering (Kluwer)*, vol. 11 no. 1, Nov 2001, pp. 141-174.

[11] ISO/IEC, *Information Technology - Security Techniques - Evaluation Criteria for IT Security - Part 1: Introduction and General Model*, International Standard 15408-1, ISO/IEC, Geneva Switzerland, 1 Dec 1999.

[12] M. Jackson, *Problem Frames*, Addison Wesley, 2001.

[13] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, United Kingdom: John Wiley & Sons, 1998.

[14] A. van Lamsweerde, "Goal-oriented Requirements Engineering: A Guided Tour," *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01)*. Toronto, Canada, IEEE Computer Society Press, 27-31 Aug 2001, pp. 249-263.

[15] A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. Edinburgh Scotland, 26-28 May 2004, pp. 148-157.

[16] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-oriented Requirements Engineering," *Transactions on Software Engineering (IEEE)*, vol. 26 no. 10, Oct 2000, pp. 978-1005.

[17] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, "Introducing Abuse Frames for Analyzing Security Requirements," *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*. Monterey CA USA, 8-12 Sep 2003, pp. 371-372.

[18] J. McDermott, "Abuse-Case-Based Assurance Arguments," *Proceedings of the 17th Computer Security Applications Conference (ACSAC'01)*. New Orleans LA USA, IEEE Computer Society Press, 10-14 Dec 2001, pp. 366-374.

[19] J.D. Moffett and B. Nuseibeh, *A Framework for Security Requirements Engineering*, Technical Report YCS368, Department of Computer Science, University of York, York UK, Aug 2003.

[20] C.P. Pfleeger and S.L. Pfleeger, *Security in Computing*, Prentice Hall, 2002.

[21] Secure Electronic Transaction LLC, *SET Secure Electronic Transaction Specification Book 1: Business Description, Version 1.0*, Purchase NY, 31 May 1997.

[22] Secure Electronic Transaction LLC, *SET Secure Electronic Transaction Specification Book 2: Programmer's Guide, Version 1.0*, Purchase NY, 31 May 1997.

[23] Secure Electronic Transaction LLC, *SET Secure Electronic Transaction Specification Book 3: Formal Protocol Definition, Version 1.0*, Purchase NY, 31 May 1997.

[24] G. Sindre and A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases," *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Pacific'00)*. Sydney Australia, 20-23 Nov 2000, pp. 120-131.

[25] J. Viega, T. Kohno, and B. Potter, "Trust (and Mistrust) in Secure Applications," *Communications of the ACM*, vol. 44 no. 2, Feb 2001, pp. 31-36.

[26] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison Wesley, 2002.

[27] E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," *Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97)*. Annapolis MD USA, 6-10 Jan 1997, pp. 226-235.

[28] E. Yu and L.M. Cysneiros, "Designing for Privacy and Other Competing Requirements," *Second Symposium on Requirements Engineering for Information Security (SREIS'02)*. Raleigh, NC USA, 15-16 Oct 2002.

[29] E. Yu and L. Liu, "Modelling Trust for System Design using the i* Strategic Actors Framework," *Trust in Cyber-societies, Integrating the Human and Artificial Perspectives*, R. Falcone, et al., Eds., Springer-Verlag Heidelberg, 2001, pp. 175-194.

[30] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *Transactions on Software Engineering and Methodology (ACM)*, vol. 6 no. 1, Jan 1997, pp. 1-30.