Technical Report No: 2003/11

Deriving Security Requirements from Crosscutting Threat Descriptions[*]

C.B.Haley
R.C.Laney
B.A.Nuseibeh

October 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

[*] A newer version of this paper appears in *Proceedings of the Fourth International Conference on Aspect-Oriented Software Development (AOSD'04)*. Lancaster UK: ACM Press, 2004.

Deriving Security Requirements from Crosscutting Threat Descriptions

Charles B. Haley, Robin C. Laney, Bashar Nuseibeh
Security Requirements Group
Department of Computing
The Open University
Milton Keynes MK7 6AA UK

{C.B.Haley, R.C.Laney, B.A.Nuseibeh}@open.ac.uk

ABSTRACT

It is generally accepted that early determination of the stakeholder requirements assists in the development of systems that better meet the needs of those stakeholders. General security requirements frustrate this goal because it is difficult to determine how they affect the functional requirements of the system.

This paper illustrates how representing threats as crosscutting concerns aids in determining the effect of security requirements on the functional requirements. Assets (objects that have value in a system) are first enumerated, and then threats on these assets are listed. The points where assets and functional requirements join are examined to expose vulnerabilities to the threats. Security requirements, represented as constraints, are added to the functional requirements to reduce the scope of the vulnerabilities. These requirements are used during the analysis and specification process, thereby incorporating security concerns into the functional requirements of the system.

General Terms

Management, Design, Reliability, Security.

Keywords

Security requirements, threats, assets, problem frames

1. INTRODUCTION

The security needs of a given system are often not determined until well into the implementation, resulting in late and expensive attempts to shoehorn security into the work in progress. Unfortunately, expressing specific security requirements is difficult. They tend to be stated as crosscutting concerns that impact many functional requirements. Moreover, security requirements are often stated in terms of how to achieve security (e.g. *the system shall use cryptography*) and not in terms of the problem to be solved, leaving it unclear how the security requirements affect the functional requirements.

Security requirements are concerned with how assets are to be protected from harm [23]. An *asset* is something in the context of the system, tangible or not, that is to be protected [13]. 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, thereby causing harm.

Aspect-oriented software development would seem to provide excellent tools for analyzing security requirements. Security requirements and functional requirements clearly crosscut each other. The concerns of the two sets of requirements are quite different, but can (and must) be composed together. However, aspect-oriented research has focused primarily on how concerns crosscut during a system's design and implementation phases. Some researchers are working near the requirements/design border (e.g. [3, 8, 25, 27]), but with the exception of [27], their arguments tend to be couched in implementation terms. The result is concepts and vocabulary based on implementation artifacts. *Join points* are "hooks where enhancements may be added" to an implementation [5]. Pointcuts are collections of join points [15]. *Advice* is a "method-like construct that can be attached to pointcuts" [19]. It is difficult to use this vocabulary when talking about early-phase artifacts. We need to remap the vocabulary.

To achieve this remapping, we must first determine what crosscuts what, then tease out the implications of this crosscutting. For our purposes, functional requirements form one set of concerns. Amongst other things, functional requirements describe how *objects* are transformed by the system. Another set of concerns are *threat descriptions*, which describe relationships between objects and threats. Join points are located where objects are shared by both threat descriptions and functional requirements; these objects are *assets* because they need to be protected. *Vulnerabilities* are found at join points by composing threats with functional requirements. The resulting *advice* is a set of security requirements that reduce the size of the vulnerability, protecting the assets.

This paper presents an approach to deriving security requirements using aspect-oriented software development crosscutting concepts and problem frames [14]. Threat descriptions are composed with a problem frames representation of functional requirements, giving vulnerabilities. Vulnerabilities are ameliorated by security requirements. Security requirements are expressed as constraints on the functional requirements, making them a more natural part of the specification process, comparable with other constraints such as safety and cost [23]. Alternatively, they can be expressed

as *trust assumptions* [9], which indicate that the security requirement is assumed to be satisfied in another context.

The remainder of this paper is structured as follows. Section 2 provides some background information on problem frames and trust assumptions. Section 3 presents our approach to deriving security requirements, using a running example. Section 4 presents related work, and section 5 describes conclusions and future work.

One note: our focus is on requirements. As such, we have put aside what today is probably the largest source of security problems: those that arise because of a 'faulty' implementation [26, 30]. Examples include the infamous buffer overflow; incorrect and incomplete input validation, especially in HTML forms; and faulty error handling.

2. BACKGROUND INFORMATION

We use problem frames [14] as a tool to organize information to facilitate derivation of security requirements. This section presents some background information on problem frames, along with a discussion of requirements and specifications in a problem frames context.

When delving into security issues, analysts must 'look outside the box'. In the problem frames context, this means that given properties must not be overly trusted, and that the analyst must be willing to include architectural considerations in their reflections. This challenge is further discussed below.

This section concludes with a short discussion of trust assumptions, which we use to narrow the context within which threat analysis is performed. There is a brief discussion about what they are, why they are useful, and where they come from.

2.1 Problem Frames

Problem frames [14] are a tool used during problem analysis. Problem frames give the shape of a solution for various problem classes. When using problem frames, the analyst decomposes larger problems into a collection of smaller ones, where each subproblem fits one of the basic problem frames. These subproblems are later recomposed, providing the solution for the original problem.

In a problem frames context, a requirements engineer describes problems by describing the interaction of *domains* that exist in the *world*. The problem frames notation captures domains in a problem along with the interconnections between them. For example, assume that the requirements elicitation process for a box that protects documents from fire produces the requirement *open the fireproof box when a door-open button is pushed*. Figure 1 illustrates one set of domains that could satisfy the requirement; a basic automatic door system with three domains, two of which are *given* and one of which is *designed*. One given domain is the box's door mechanism domain, capable of opening and shutting the box's door. The second given domain is the one requesting that the door be opened; this domain includes both the 'button' to be pushed and the human pushing the button. The designed domain is the *machine*, the domain that will bridge the gap between the other two domains in order to fulfill the requirement that the door open when the button is pushed. The oval presents the requirement that the machine is to satisfy.

Every domain has *interfaces*, which are defined by the *phenomena* visible to other domains. Phenomena (e.g. events and signals) are

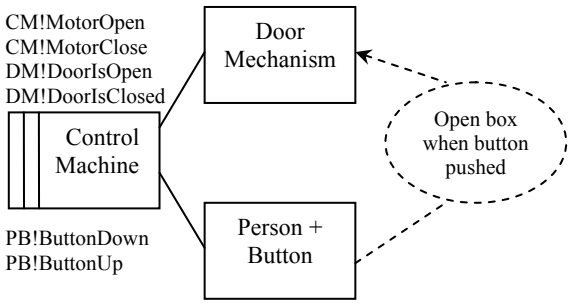

Figure 1 – A basic Problem Frames diagram

visible; they can be observed. The notation shows the phenomena shared between two domains on the line between the domains, along with indicating which domain controls the phenomena. In Figure 1, the Person + Button domain above produces the event phenomena ButtonDown and ButtonUp, as indicated in the diagram. Alternatively, the designer could have chosen to produce a single event: OpenDoor. The Control Machine produces the Boolean phenomena MotorOpen and MotorClose (turn on and off the motor) on the interface between the machine and the Door Mechanism. The Door Mechanism produces the Boolean phenomena DoorIsOpen and DoorIsClosed.

Requirements are optative, describing desired behavior instead of existing behavior [14]. Descriptions of the *desired* behavior of individual parts of the system (the designed domains) are also optative. Descriptions of the *actual* behavior of given domains (their phenomena: inputs, outputs, and states visible at their interfaces) are indicative; they describe an "objective truth" about the behavior of the domain.

2.1.1 Requirements and Specifications

According to Zave and Jackson [33], a *requirement* is an optative description of what the system is to do. Requirements describe a *desired effect*, or a *goal*. Jackson [14] describes a requirement as "the effects in the problem domain that […] the machine is to guarantee." Van Lamsweerde's [16] approach is similar, defining a goal as "an objective the system under consideration should achieve" and then saying that a requirement is a goal that can be achieved by an *agent* in the software-to-be [18]. The *i** framework uses a comparable definition; goals model the intentions of stakeholders [2].

Again referring to Zave & Jackson, *specifications* are about phenomena. The specification of an individual domain is a description of the behavior of the domain in terms of its phenomena, indicative and optative, visible at its interface. The specification of a system is the collection of domain specifications that together permit the fulfillment of the requirement(s).

The distinction between requirement and specification is an important one, especially when working with security requirements. A security requirement does not describe how security is to be implemented, but instead describes what is desired. It is the specification that describes how, in terms of its externally visible phenomena, the requirement is fulfilled. For example, *the information on the network will be encrypted with Blowfish* is not a security requirement, but is instead a specification. The underlying requirement would be something resembling *information shared at the interface of the domains in this problem is not to be generally understandable*.

2.1.2 Indicative vs. Optative

Indicative domain properties are normally expected to be constant, e.g. the same stimulus in the same context produces the same response. This is what Jackson meant by "objective truth" [14]. Unfortunately, when reasoning about security we should put aside this nice concept and assume that all domain properties are optative. Consider the pushbutton in the domain shown in Figure 1; when the button is pushed, the circuit connected to the button is closed. This would seem to be an indicative property. Now put some confidential information in the box, and then consider the same button from the point of view of an attacker. The attacker might cut the wire, connect an alternate or second button to the wire, or put a circuit in the middle that analyzes the context of the button push and either passes it on or doesn't. The property can no longer be considered objectively true. It has become optative; what we *want* to be true.

Security requirements (the constraints) are similar to functional requirements. They are optative, describing characteristics of the system that the requirements engineer desires to be true. The lesson learned from the above discussion is that, unlike functional requirements, security requirements should assume that indicative domain properties are optative, because a goal of an attacker might be to change the behavior of some indicative phenomena. A successful attack means one of two things: phenomena exist that were not described in the problem, or phenomena behavior (the specification) assumed to be indicative (to be true), is not[1].

2.2 Parallel Elaboration of Requirements & Architecture

The Twin Peaks model [24] shows that the elaboration of requirements and architecture should proceed in parallel, each influencing the other. This is doubly true in the context of security requirements, because security is a systems problem [23]. One cannot accurately determine the security requirements outside the context of the system.

To illustrate the idea, consider a trivial functional requirement *business plans shall be written using a word processor and stored on a file server*. In addition, assume the existence of the general security requirement *business plans are to be treated as company-confidential information*. Without knowing the domains involved in the problem, how do we know how to keep the information confidential? We can postulate the existence of computers used to write and store business plans, but we cannot go much further. The designer could choose to put the machines in a locked room, in which case the *room key* becomes a phenomenon in the problem and the security requirements must describe the constraints on obtaining and using a key. Alternatively, the designer might specify a client/server architecture in which the client machines are publicly accessible. In this case, the client machine domain can be physically accessed by anyone and the business plans are potentially visible where the client and server domains connect (the network). The security requirements must describe constraints on who can use the client machine and on who can 'see' the information where the domains connect.

[1] These two positions could be reduced to one. The argument would be that attack does not alter the behavior of existing phenomena. Instead, new phenomena are created, thus changing the problem. This distinction is not important for this paper.

It is highly likely that applying a security requirement to a problem will create subproblems, add domains to the existing problem, or both. For example, the specification to fulfill a security requirement *information shared between the client and server domain must not be accessible* must be evaluated in terms of visible phenomena. The designer must assure either that information shared between the domains is not visible outside the problem or that 'seeing' what passes between the domains does not reveal the information. Either way, the physical properties of the connection need to be described.

2.3 Trust Assumptions

A *trust assumption* is a decision to trust the given properties of some domain and to go no further in the analysis [9]. Trust assumptions are used to bound the context of a problem, limiting the number of domains that are directly involved in the analysis.

The notion of trust assumptions and the need for their explicit capture are well summarized by Viega and McGraw in [30]:

> *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.*

and

> *System architects must constantly deal with trust issues during an application's design cycle.*

We use the definition of trust proposed by Grandison & Sloman [7]: "*[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 context, we say that the analyst trusts that some domain will participate competently and honestly in the satisfaction of a constraint.

Two examples should help clarify the use and utility of trust assumptions. The first concerns physical security. Assume that an analyst ensures confidentiality using a locked-room approach. The analyst is probably assuming that the corporate key-giver will ensure that keys are given only to authorized individuals. However, the key-giver may be required to respect a safety constraint, meaning that keys must be provided to the fire department and/or the safety officer(s). These people may not be authorized to see the information kept in the room. The conflict renders the analyst's assumption invalid. Making the assumption explicit would improve the chances that such conflicts are noticed.

For the second example, assume the existence of an availability security requirement stating that 8 hours of backup power must be provided. A designer might choose to satisfy the requirement with backup generators. Appropriate phenomena would be added to the problem to control the generators, detect going beyond 8 hours, etc. In this case, it is probably safe to assume that the provider of the generators can be trusted to supply generators without a backdoor into the control circuitry that would allow an attacker to control the generator's response. By making this assumption, the analyst can stop at the generator domain instead of having to include the generator supplier (and its suppliers, etc.) in the problem context.

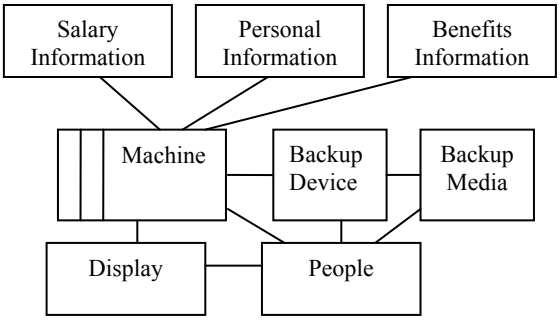

Figure 2 – Example Context Diagram

3. USING THREAT DESCRIPTIONS

This section describes how one composes threat descriptions with functional requirements to derive security requirements.

A threat description is a descriptive phrase of the form *performing action X on/to asset Y could cause harm Z.* Some examples:

- Exposing the company's business plan could cause loss of revenue.
- Altering the balance of an account could cause financial loss.
- Destroying the village in the valley could cause loss of life.

A threat description can be represented in a prescriptive form by inverting and negating the phrase. For example, the first example above would become *avoid loss of revenue by preventing the exposure of the company's business plan.* Representing threat description in both descriptive and prescriptive forms might help an analyst be more certain that all the implications of the threat descriptions have been determined.

A threat description may be represented by a tuple *(threat, asset)*, where the tuple element 'threat' is itself a tuple *(action, harm)*. The first threat description example is shown in tuple form below.

$$\big((\text{exposing}, \text{loss of revenue}),\ \text{business plan}\big)$$

It is important to note that the action in a threat description is the inversion of some common security goal, such as *confidentiality* (further discussed below). The action is not intended to describe an attack, which is an act (or sequence of acts) of an attacker carried out to exploit a vulnerability and therefore carry out a threat. For example, an attacker might carry out the first threat above by diverting the business plan to a second printer while it is being printed. The diversion is the attack, exploiting what is probably a vulnerability in the network.

3.1 The Composition Process

Deriving the security requirements is an iterative process. Each iteration recomposes the threats with the functional requirements to derive new security requirements; if none are found then there is no need to iterate further. Iterations are required because identifying and eliminating vulnerabilities will often create new vulnerabilities.

Each iteration of the approach requires four general steps:

1. Identify objects in the problem context that might participate in a threat. These are candidate assets.
2. Identify threats on the candidate assets, creating threat descriptions. An asset is an object that participates in a threat.
3. Crosscut threat descriptions with the functional requirements, looking for whether an asset is in a domain involved in the subproblem, ending up with the set of tuples illustrated below.

   $$\{(\text{threat},\ \text{asset},\ \text{subproblem})\}$$

   Compose the threat with the subproblem to determine whether there is a vulnerability that permits fulfillment of the threat associated with the asset. Enumerate constraints on the functional requirements to weaken the vulnerability to an acceptable level.
4. Identify conflicts.

Each of these steps is presented below.

It must be noted that we are not describing a mechanical process. There is no crank that, when turned, produces the answer. Steps are approximate and can be combined. Depending on the problem, sub-iterations might be required. Finally, the skill and experience of the analyst is crucial.

3.2 Description of the Example

A common example is used throughout this section to illustrate the concepts. Space constraints force the example to be small and somewhat contrived. The example consists of a small Human Resources system having four functional requirements. A problem context diagram, showing the initial set of domains to be used by the problem frame diagrams, is shown in Figure 2.

- REQ1: Salary, personal, and benefits information can be entered, changed, and deleted. This information is stored for eventual use in producing paychecks.
- REQ2: A subset of his or her own personal and benefits information will be available to each managerial employee for perusal.
- REQ3: An 'address list' subset of personal information consisting of the employee's name, office, and work telephone number will be generally available.
- REQ4: All information will be backed up daily.

We postulate the existence of three general security requirements:

- Salary information is highly confidential and is to be shown only to authorized individuals.
- Personal and benefits information shall be restricted to the employee and to others with a need to see it.
- The 'address list' of employees of the company is company confidential.

Finally, we find one general requirement:

- The system shall not be expensive to administer.

Figures 3 through 5 present subproblem diagrams for the first three functional requirements. Phenomena are not included in the diagrams, but will be added later where needed.

3.3 The Iterations

3.3.1 Identify Candidate Assets

The goal of this step is to find all the objects in the context of the problem that might have value, directly or indirectly. In general, these consist of all the information objects stored in or accessed by the system-to-be and any tangible objects such as the computers themselves. An object is has direct value when the potential harm caused by a threat is to the object itself. An object has indirect value when a threat involving that asset causes harm

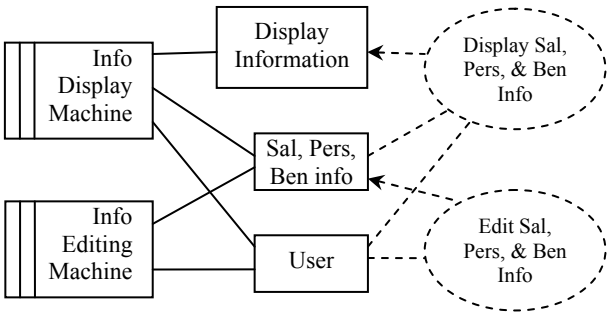

Figure 3 – REQ1 – Payroll data

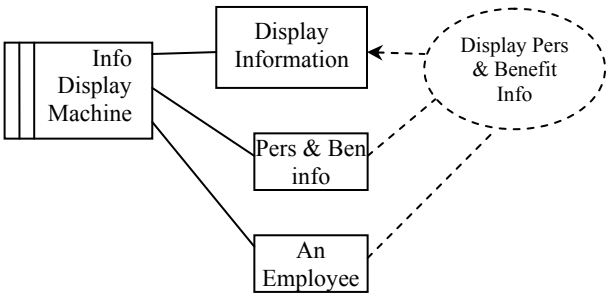

Figure 4 – REQ2 – Employee-visible information

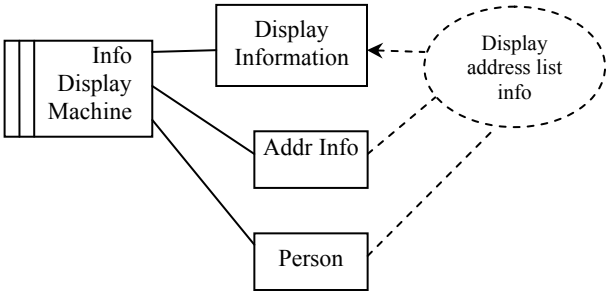

Figure 5 – REQ3 – Address list

somewhere else, such as to revenue, to costs, or to reputation. An object can have both direct and indirect value; when money is taken from a bank, it loses both the money and its reputation.

One potential asset might contain, or enclose, other potential assets. A good example is a database that contains individual information assets. Another example is backup media, which can contain any number of information assets.

When using problem frames, listing the objects in the problem is straightforward. They are the domains, given and designed, in the problem context. For our example, they are *Salary Information, Personal Information, Benefits Information, Machine, Backup Device, Backup Media,* and *People.*

Because of space limitations, this paper restricts the discussion to threats involving information assets.

3.3.2 Identifying Threats

3.3.2.1 Security Concerns

In general, harm is caused by the negation of one or more crosscutting security concerns. For information assets, these concerns are described as CIA: *confidentiality*, *integrity*, and *availability* [26]. The concerns are similar for tangible assets: *exposure*, *modification*, and *deprivation* (theft or destruction).

Confidentiality concerns are about restricting access to information, and are applied to information assets or information to be garnered from physical assets (exposure). It should be noted that confidentiality is not the same thing as privacy [10, 21]; privacy incorporates as an additional concern the *use* one makes of an information asset, and is not discussed in this paper. Authentication is related to confidentiality, and is concerned with ensuring that users are who they say they are, and remain so during the 'session'.

Integrity is concerned with ensuring that an asset is not modified without authorization. This concern covers several scenarios, including direct modification of data, modification through unauthorized transactions, indirect modification using backup media, and transient modification before it is displayed to a user. Non-repudiation, where a person cannot claim after the fact to have not performed some act, is related to integrity.

Availability concerns are about ensuring that an asset is usable (available) when it is supposed to be. For example, cutting power to a computer makes it unavailable. If access is over a network then anything that blocks data movement on the network will affect availability. Clearly, physically removing an asset also affects availability (deprivation).

These concerns crosscut every object in a system. They also crosscut each other. For example, availability supports confidentiality when the object in question is a physical asset. Having the asset stolen makes it unavailable and allows the thief to know any information associated with the asset.

3.3.2.2 Using the Concerns

The concerns are used to enumerate the threat descriptions. One asks questions of the form "what harm could come from violating the [insert concern here] of [insert object here]?" Answers to these questions are threat descriptions.

Threats can have a 'time' element, stating that the harm will occur only if the violation occurs before or after some point, or within some interval. For example, a company's earnings report is confidential (and therefore valuable) only up to the moment it is made public. The time element is important when looking for and countering vulnerabilities, as it gives an indication of how severe a given vulnerability is and what measures are appropriate for countering the vulnerability.

We note again that an object may not have any value in itself, but instead is valued by the harm indirectly caused to something else. For example, information about the amount of money paid to redecorate the company president's office has no intrinsic value, but may be highly valued because exposing the figure could damage the reputation of the company. In other words, when evaluating how assets are associated with threats, one must look for direct and indirect effects.

Remember that the initial set of assets came from the problem frames context. Threat descriptions indicate which threats involve which assets. Assets crosscut the domains in the context. Domains crosscut subproblems. Therefore threats crosscut subproblems. Any subproblem that incorporates an asset might contain vulnerabilities allowing the threats associated with that asset to be

consummated. If analysis locates a vulnerability in a subproblem, then security requirements must be added to the subproblem to reduce the size of the vulnerability to an acceptable level.

A domain that is a projection of more than one domain found in the problem context can contain multiple assets. Actions on the domain as a whole must be analyzed as if they operate on the component assets. For example, consider a database domain in a backup/recovery subproblem. The database contains the individual information assets. If one of the contained assets is related to a threat involving integrity, then the domain itself is related to that threat and must be analyzed in that context. A threat against salary information will also be a threat against backup media containing salary information.

3.3.2.3 The Example

Continuing the example, asking questions about the object *salary information* might produce the following threat descriptions:

- (Confidentiality) Releasing salary information to unauthorized individuals could damage the company's reputation. Additionally, it increases costs by making 'employee theft' by competitors easier.
- (Integrity) Unauthorized changes could increase costs by increasing the size of the payroll and damage reputation by provoking lawsuits or involving police.
- (Availability) Denying access could damage the company's reputation, reduce employee motivation, and incur damage payments for employees who were not paid on time.

Similar answers can be found for *personal information* and *benefits information*. The result is threat description tuples of the form

((Release, damage reputation), salary information)

Turning our attention to the information assets in the machine domain, we determine:

- (Confidentiality) Exposing the machine could result in the information on its disks being exposed, resulting in the harm described above.
- (Integrity) Modifying (or replacing) data on the hard disk can result in the harm described above.
- (Availability) Destroying the machine can result in loss of access to information, resulting in the harm described above.

Looking at the backup device, we see:

- Replacing the device with one that makes covert copies can result in exposure of all the information assets.
- Causing the device to modify information before it is written to backup media can result in the integrity harm described earlier.
- Causing the device to write incorrect backups, when coupled with destruction of the machine, could trigger the availability harm described earlier.

Looking at backup media, we can conclude that all of the harm to information assets can be triggered through examination, alteration, or destruction of the backups.

3.3.3 Composition – Identify Security Requirements

3.3.3.1 Discussion

The previous step identified the threat descriptions: threats and the assets that participate in the threats. One must now crosscut the threat descriptions with the subproblems to determine which threats apply to a given subproblem. The threats are composed with the subproblems, looking for vulnerabilities that might allow the threat to come to pass. If vulnerabilities are found, then security requirements are generated to reduce the vulnerability.

The analyst begins by listing the subproblems being considered and identifying the assets in each subproblem. This produces a set of tuples of the form $\{(\text{asset}, \text{subproblem})\}$. This set and the threat descriptions set found above are combined using a natural join to find which threats affect which assets in which subproblems, as shown below.

$$\{(\text{threat}, \text{asset}, \text{subproblem})\} = \{(\text{threat}, \text{asset})\} \mid\times\mid \{(\text{asset}, \text{subproblem})\}$$

For small problems such as the examples in this paper, generating the set of tuples informally is sufficient. When working with larger problems, generating the tuples and performing the join will help ensure that nothing is forgotten.

For each tuple, the analyst looks at the domains in the subproblem to determine whether they create vulnerabilities. For example, consider the domain 'person + button' in the subproblem shown in Figure 1. If the safe contains confidential information, then a threat exists involving an unauthorized person viewing that information. The very structure of the subproblem allows the threat to be realized. There is no way to restrict who uses the button, because the only phenomena visible are related to the button itself. To fix the problem one must either a) restrict the people who can get to the button through some external means, b) separate the person domain from the button domain and add appropriate authentication phenomena between the person domain and the machine, or c) add authentication phenomena at the existing interface. In cases b & c, appropriate domains to access authentication information must be added, which could easily create new vulnerabilities.

After being satisfied that the domain structure is valid, the analyst looks for vulnerabilities at the interfaces between domains and in the phenomena shared across them. Can threats be realized through eavesdropping on phenomena? Are there connection domains that are not modeled, such as networks? Do the phenomena have physical properties that make them vulnerable, such as electromagnetic or power usage signatures? Are the phenomena caused by physical items, such as keys, that might be duplicated? What happens if a sequence of phenomena is recorded and played back?

It is not sufficient to look only at the domains containing the threatened assets. There might be indirect vulnerabilities where interfaces between two different domains allow the threat on the asset domain to be carried out, as shown by the 'person + button' example discussed above. All domains in the subproblem must be examined, as must the domains in other subproblems that eventually recompose with the subproblem under examination.

If examination reveals a vulnerability, then an appropriate security requirement must be added to the subproblem. For example, if a replay vulnerability exists (a phenomena sequence can be recorded, and the replayed to produce some harm), then the

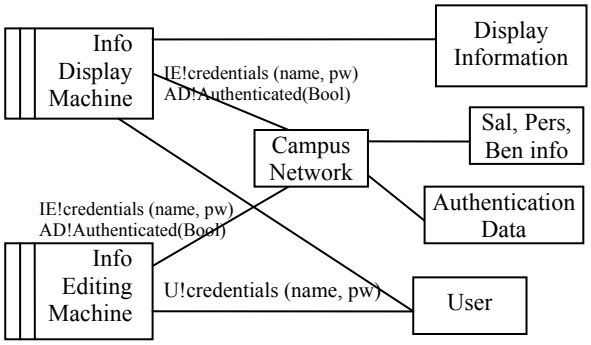

Figure 6 – REQ1' – Payroll data

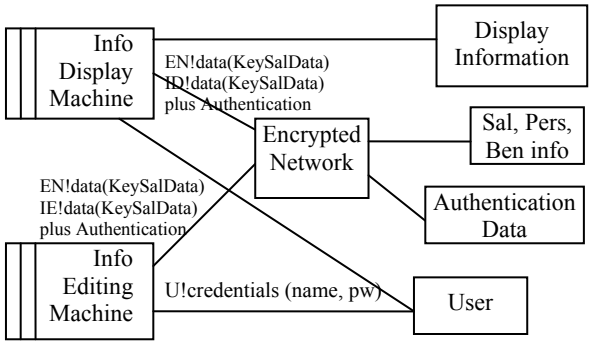

Figure 7 – REQ1'' – Payroll data

requirement *replays must be detected and ignored* must be added. If a connection domain between domains X and Y is vulnerable to eavesdropping, then the requirement *information phenomena between domains X and Y must not be understandable by an eavesdropper* must be added.

Unless new requirements are satisfied using trust assumptions, their satisfaction will probably involve changing the visible phenomena and could involve changing the domain structure of the subproblem. New domains might create new subproblems. New phenomena could change design decisions. These changes could create new assets and/or new vulnerabilities. Probably a turn through the requirements/architecture spiral will be required. Certainly the threats analysis must be repeated.

Iterations through the process can add domains to the overall problem context. This happens because the analyst must look further and further down through the phenomena chain to ensure that the requirements are being met. A point will come where the analyst is unwilling or unable to go further, because either the process is too difficult, the expected return too small, or because the problem is believed solved in another context. The analyst uses trust assumptions at these points. The assumptions state the analyst's beliefs that the properties of domains in or out of the context can be trusted to an acceptable level. By using a trust assumption, the analyst is putting bounds on the problem that the system-to-be must solve.

3.3.3.2 The Example

Turning to our examples, we see that every problem is affected by at least one of the threats.

Problem REQ1

Starting with Figure 3, Payroll Data, we see that several threats are related to unauthorized exposure of assets contained in the projection domain 'Sal, Pers, Ben Info'. Noting that the 'user' can apparently be anyone, we add the constraint *only HR staff can edit or view information.* We do not have enough information to know whether or not there are vulnerabilities at the interfaces; we need to see the phenomena including any needed to meet the above constraint. Turning to the designer, we obtain a subproblem diagram with the designer's chosen authentication scheme, as shown in Figure 6.

From this diagram we conclude that the interface between the user and the machine seems safe. We are told that the Authentication Data and Information domains are supplied domains and not to be designed as part of this effort. However, there is a public network between the machines being used to process the data and the data itself. We must add the requirement *information passing over the network must not be understandable by an eavesdropper*.

When confronted with the problem, the designer produced the diagram shown in Figure 7.

We note the encrypted network, and immediately add a constraint *encryption keys must not be revealed*. Asking the designer about the keys, we are told that a key exchange protocol will not be used, that the keys are physically built into the machines, and that the machines can be trusted not to reveal them[2]. Noting that problems REQ1 and REQ2 both access the information domains but with different security requirements, we enquire into the properties of the two given domains, Info and Authentication Data. We need to know how they control what information they release to a client on the network. We are told that it is the encryption key that tells the domains what information they may release. We are told further that these domains are under the responsibility of the IT organization and physically and logically secure; we do not need to worry about them. These assurances translate into trust assumptions, shown in Figure 8.

Problem REQ2

Reasoning about REQ2, employee access to personal and benefits information, is very similar to the above, except that the population of authorized users is much larger and the salary information must not be displayed.

Analyzing this problem will result in a combination of the results of REQ1 (above) and REQ3 (below). For space reasons the argument and diagrams are not included in this paper.

Problem REQ3

There are two threats that can affect REQ3, the address book. The first is that allowing the address information to leave the company can cause harm. The second is that if the machine prepares the address list by using the larger data assets, it might be possible to carry out the threats involving these assets. When provoked, the designer produces the diagram shown in Figure 9. The same encrypted network is being used in this subproblem as was used in the payroll data subproblem, but with different keys, so we will need to eventually add the same trust assumptions. The analyst was told that because different encryption keys are used, no information in the Salary and Benefits Information domains is accessible. Thus no vulnerability exists that could permit carrying

[2] The assumptions in this paragraph are unrealistic and naïve, but are acceptable for this paper.

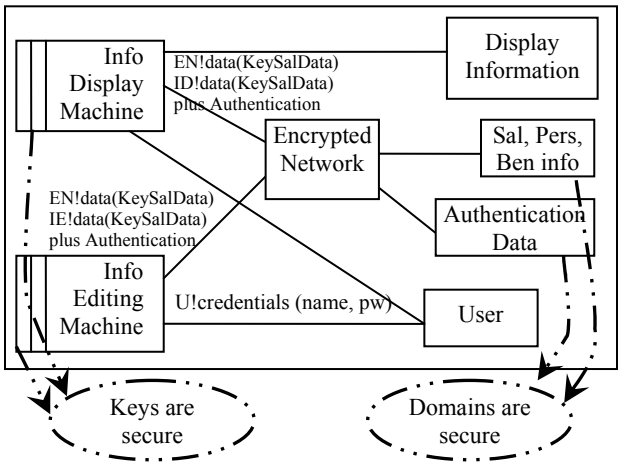

Figure 8 – REQ1''' – Payroll data

out threats against these more confidential data assets. Also, because the keys are different, the Info Display machine cannot expose confidential data 'lifted' from the network. A trust assumption is added to capture this information.

The designer has not added authentication phenomena to the interaction between the user and the system, so we have not resolved that threat. The designer says that the company uses many agency workers and that it is impractical to give all such workers a username & password; doing so would cost far too much money. The security officer says that the information must be protected. The requirements, reasonable cost and appropriate security, are in conflict. The conflict is resolved below.

3.3.4 Resolving Conflicts

3.3.4.1 Discussion

Security requirements often conflict with each other, as well as with other requirements. For example, the result of applying the CIA concerns can conflict with revenue & ease-of-use. Viega and McGraw [30] provide an example from the credit card world, saying that the credit card companies know ways to reduce fraud dramatically, but they do not use them because the cost of business lost would exceed the loss caused by the fraud.

3.3.4.2 The Example

The address-list display problem brought out a conflict between administration cost and security. Discussions between the security officer and the IT organization led to the realization that because the system was limited to inside company buildings, in theory only company employees and agency workers could access the system. This satisfied the security officer, so a trust assumption that the building security system restricts the membership of the user domain to *employees* was added. The completed diagram is shown in Figure 10.

There is seldom a clear-cut answer to a conflict. The analyst must make decisions based on estimates of risk and cost. Work to assist the analyst with these decisions is in progress, and will be presented in a future paper.

4. RELATED WORK

Several projects are looking at requirements and non-functional requirements including security.

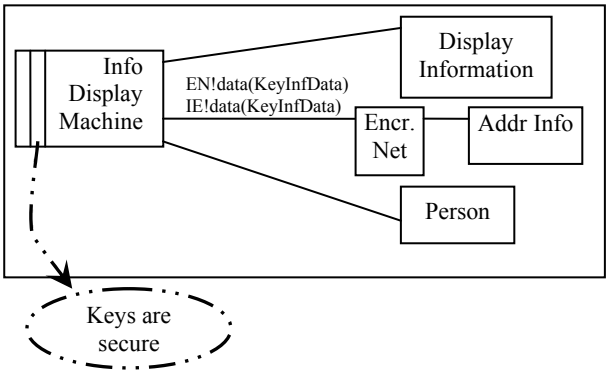

Figure 9 – REQ3' – Address list

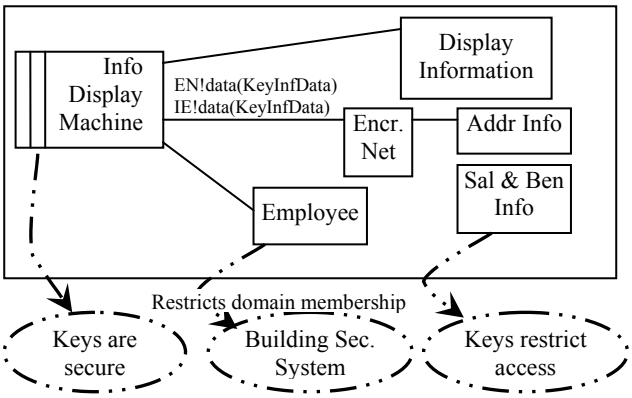

Figure 10 – REQ3'' – Address list

Rashid et al propose that ideas from aspect-oriented software development can be used when mapping non-functional requirements onto functional requirements [27, 28]. They start by identifying the non-functional requirements (NFRs) that affect (crosscut) more than one functional requirement, determine what the effect of the overlap is, then model the composition of the requirements. In their work, security is treated identically to other NFRs. Their work is more general than the work presented in this paper. It focuses on managing the interplay and the results of composition of the requirements, not deriving requirements from the NFRs.

van Lamsweerde et al use "obstacles" to model security & safety [18] in KAOS, and are developing the notion of anti-goals to describe and close vulnerabilities [17]. In KAOS one starts with generic "root anti-goals" which are the inversions of CIA (plus privacy) and determines which agents could benefit from application of the anti-goals, while we start with asset identification and then determine the threats involving the assets. The approaches converge at the end. In KAOS one adds 'avoid' predicates to close the vulnerabilities, while in this work security constraints are added to functional requirements. We speculate that in KAOS, concentrating on who benefits (as opposed to what can be attacked) and the closed nature of the domain model will tend to limit the vulnerabilities found during generation of the anti-model. More work is required to determine and compare the expressive powers of each approach.

Alexander is looking at detecting vulnerabilities using misuse cases [1], as is Sindre et al [29]. McDermott uses 'abuse cases'

[22]. In and Boehm have adapted the WinWin framework to include security requirements [12], and Heitmeyer has done the same with SCR [11].

Several teams are looking at the role of trust in security requirements engineering. In the *i** framework [31, 32], Yu, Lin, & Mylopoulos take an 'actor, intention, goal' approach where security and trust relationships within the model are modeled as "softgoals": goals that depend on another actor to satisfy them. The Tropos project [6] uses the *i** framework, adding on wider lifecycle coverage. Neither model captures the analyst's assumptions about the domains that make up the solution to the problem. As such, an *i** model complements the framework presented here, and in fact can be used to determine the initial goals, requirements, and constraints.

He and Antón [10] are concentrating on privacy, proposing 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.

5. CONCLUSIONS & FUTURE WORK

We have shown how representing security NFRs as crosscutting threat descriptions assists with composing these requirements with the functional requirements. Composition, an iterative process, gives us a set of constraints on the functional requirements, which we call security requirements. Security requirements are analyzed along with other constraints when producing specifications for the problem.

The principal advantage of our approach is that it permits conversion of general security requirements into something more akin to functional requirements. The constraints apply to specific functional requirements, much as other constraints such as those related to safety do. All of the constraints can be considered together and in the context of a particular requirement. The crosscutting nature of the threat descriptions permits consistency analysis to ensure that assets are treated uniformly and appropriately throughout the system.

Much work remains to be done. The composition process described in this paper is informal. It would be useful to have some traceability between constraints and the threat(s) they counter; we are looking at the representation Rashid et al propose in [27] and at an adaptation of the multi-dimensional concerns matrix proposed by Ossher and Tarr in their work describing on-demand application remodularization [25]. Better integration with some of our colleagues' research, such as the organizational access control work of Crook [4] and the abuse analysis work of Lin [20], is desired. As noted in the related work section, the expressive powers of this approach and others (e.g. anti-goals in [17]) need to be better understood.

A principal future focus will be the introduction of cost and risk into the approach described by this paper. Quantifying the levels of trust in trust assumptions and the levels of harm in threat descriptions provides a starting point for calculating the potential risk associated with a vulnerability. Add the notion of cost or difficulty to constraints intended to reduce a vulnerability, and we have a way to compare the cost of closing a vulnerability with the expected harm if nothing is done. This information could help an analyst determine whether the effort to reduce a given vulnerability is appropriate or necessary.

6. ACKNOWLEGEMENTS

The financial support of the Leverhulme Trust is gratefully acknowledged. Thanks also go to Jonathan Moffett and Michael Jackson for their continued support and helpful comments, and to our OU colleagues Debra Haley, Janet van der Linden, and Alistair Willis for their careful and thoughtful review of the paper.

7. REFERENCES

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

[2] Castro, J., Kolp, M., & Mylopoulos, J. "A Requirements-Driven Development Methodology," In *Proceedings of The 13th Conference on Advanced Information Systems Engineering (CAiSE'01)*. Interlaken, Switzerland, 4-8 Jun 2001, pp. 108-123.

[3] Clarke, S., & Walker, R. J. "Composition Patterns: An Approach to Designing Reusable Aspects," In *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*. Toronto, Ontario, Canada: IEEE Computer Society Press, 12-19 May 2001, pp. 5-14.

[4] Crook, R., Ince, D., & Nuseibeh, B. "Modelling Access Policies Using Roles in Requirements Engineering," *Information and Software Technology (Elsevier)* 45(14), November, 2003: pp. 979-991.

[5] Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K., & Ossher, H. "Discussing Aspects of AOP - a Panel Discussion," *Communications of the ACM* 44(10), Oct, 2001: pp. 33-38.

[6] 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, DIT-03-027. University of Trento, May 2003.

[7] 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.

[8] Grundy, J. "Aspect-Oriented Requirements Engineering for Component-Based Software Systems," In *Fourth IEEE International Symposium on Requirements Engineering (RE'99)*. Limerick, Ireland: IEEE Computer Society Press, 7-11 Jun 1999, pp. 84-91.

[9] 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.

[10] He, Q., & Antón, A. I. "A Framework for Modeling Privacy Requirements in Role Engineering" in 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.

[11] 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, pp. 84-89.

[12] In, H., & Boehm, B. W. "Using WinWin Quality Requirements Management Tools: A Case Study," *Annals of Software Engineering (Kluwer)* 11(1), Nov, 2001: pp. 141-174.

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

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

[15] Kiczales, G., Hilsdale, E., Hugunin, J., & Kersten, M. "An Overview of AspectJ," In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*. Budapest, Hungary: Springer-Verlag, 18-22 Jun 2001, pp. 327-353.

[16] 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, pp. 249-263.

[17] 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" in Requirements for High Assurance Systems Workshop (RHAS'03), *Eleventh International Requirements Engineering Conference (RE'03)*, Monterey, CA USA, 2003.

[18] van Lamsweerde, A., & Letier, E. "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering* 26(10), Oct, 2000: pp. 978-1005.

[19] Lieberherr, K., Orleans, D., & Ovlinger, J. "Aspect-Oriented Programming with Adaptive Methods," *Communications of the ACM* 44(10), Oct, 2001: pp. 39-41.

[20] 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, pp. 371-372.

[21] Liu, L., Yu, E., & Mylopoulos, J. "Security and Privacy Requirements Analysis Within a Social Setting," In *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*. Monteray Bay, CA USA, 8-12 Sept 2003.

[22] 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, pp. 366-374.

[23] Moffett, J. D., & Nuseibeh, B. *A Framework for Security Requirements Engineering*, Department of Computer Science, YCS368. University of York, UK, 2003.

[24] Nuseibeh, B. "Weaving Together Requirements and Architectures," *IEEE Computer* 24(3), March, 2001: pp. 115-119.

[25] Ossher, H., & Tarr, P. "Multi-Dimensional Separation of Concerns and the Hyperspace Approach," In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluver Academic Press, 2000.

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

[27] Rashid, A., Moreira, A. M. D., & Araújo, J. "Modularisation and Composition of Aspectual Requirements," In *Proceedings of the 2nd International Conference on Aspect-oriented Software Development (AOSD'03)*. Boston, MA USA: ACM Press, 17-21 Mar 2003, pp. 11-20.

[28] Rashid, A., Sawyer, P., Moreira, A. M. D., & Araújo, J. "Early Aspects: A Model for Aspect-Oriented Requirements Engineering," In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*. Essen, Germany, 9-13 Sep 2002, pp. 199-202.

[29] 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, pp. 120-131.

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

[31] 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, pp. 226-235.

[32] Yu, E., & Liu, L. "Modelling Trust for System Design Using the i* Strategic Actors Framework," In R. Falcone, M. P. Singh, & Y.-H. Tan, eds. *Trust in Cyber-societies, Integrating the Human and Artificial Perspectives*. Springer-Verlag Heidelberg, 2001: pp. 175-194.

[33] Zave, P., & Jackson, M. "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology* 6(1), Jan, 1997: pp. 1-30.