ISSN 1744-1986

Technical Report N° 2004/23

Core Security Requirements Artefacts

Jonathan D. Moffett,
Charles B. Haley,
Bashar Nuseibeh

21st June 2004

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

http://computing.open.ac.uk

Core Security Requirements Artefacts

Jonathan D Moffett Charles B Haley Bashar Nuseibeh

Security Requirements Group
Department of Computing
The Open University, UK

{J.Moffett, C.B.Haley, B.Nuseibeh}@open.ac.uk

Abstract

Although security requirements engineering has recently attracted increasing attention, it has lacked a context in which to operate. A number of papers have described how security requirements may be violated, but apart from a few hints in the general literature, none have described satisfactorily what security requirements *are*.

This paper proposes a framework of core security requirements artefacts, which unifies the concepts of the two disciplines of requirements engineering and security engineering. From requirements engineering it takes the concept of functional goals, which are operationalised into functional requirements, with appropriate constraints. From security engineering it takes the concept of assets, together with threats of harm to those assets. Security goals aim to protect from those threats, and are operationalised into security requirements, which take the form of constraints on the functional requirements.

In addition we explore the consequences of the fact that security is concerned with the protection of assets, while computers only provide interfaces. We show how to specify the relationship between security requirements and the specification of software behaviour, using Jackson's Problem Frames approach.

Table of Contents

When you are up to your elbows in alligators, you must never forget that you set out to drain the swamp. [anon]

1. Introduction

The subject matter of this paper is the description of the core artefacts that are needed to carry out security requirements engineering; this is the process of eliciting, specifying and analysing the security requirements of a system. It is a development of our earlier paper [42].

We propose a framework which integrates the concepts of the two disciplines of requirements engineering and security engineering. From requirements engineering it takes the concept of functional goals, which are operationalised into functional requirements, with appropriate constraints. From security engineering it takes the concept of assets, together with threats of harm to those assets. Security goals aim to protect assets from those threats, and are operationalised into security requirements, which initially take the form of (a subset of) the constraints on the functional requirements.

This paper has a view of security requirements, based on the following principles:

- The "what" of security requirements – its core artefacts – must be understood before the "how" of construction and analysis.
- Security cannot be considered as a feature of software alone; it is concerned with the prevention of harm in the real world. We must therefore consider both the security requirements of real-world systems and the specification of software that demonstrably meets those requirements.
- Since security is largely concerned with prevention of misuse of system functions, security requirements can most usefully be defined by considering them at the same level as functional requirements, and as constraints upon them.

Scope of the Framework

This framework has been developed in order to understand the place of security requirements within the development of an individual application, and our proposals are limited by that scope. The application will, of course, be developed in the context of a software operating environment, a hardware environment, and a human cultural environment. All of these environments will have properties which impact upon the application. However, we have not covered these in this paper. In particular, we believe that there are many issues with regard to the properties of the software operating environment, which need to be tackled in later work.

1.2 Background

Although security requirements engineering has recently attracted increasing attention, it has lacked a context in which to operate. This lack was pointed out by Baskerville [6], where he presents three generations of security design methods:

- Checklists: these are lists of points to be checked, on the assumption that experience of previous applications can be applied to the current one.
- Mechanistic engineering methods: a process is used, which focuses on security in isolation from other aspects of system design.
- Integrated design: A development process includes security as a facet of the whole development.

Unfortunately he was unable to point to any examples of integrated design methods that were used in practice. His comments apply to design, but it is also true today that there is no satisfactory integration of security requirements engineering into requirements engineering as a whole. In this section we review existing literature, in order to show the truth of this statement, and then motivate the remainder of the paper by showing why it matters.

1.2.1 Previous Definitions of Security Requirements

Extensive work has been carried out on security requirement during the last few years. However, there has been a lack of a satisfactory definition of them. Work on the subject has tended to be carried out independently by the security and requirements communities.

The Security Community

From the security community side, there are several papers on security requirements. Tettero [54] defines security requirements as the confidentiality, integrity and availability of the entity for which protection is needed. While we accept that this is a clear definition, we will argue below (section 4.2) that it is too abstract. Lee et al [34] point out the importance of considering security requirements in the development life cycle, but do not define them.

ISO/IEC 15408 [25] does not define them in its glossary. However, in one place they are depicted as being at a higher level than functional requirements, but elsewhere the reference to "security requirements, such as authorisation credentials and the IT implementation itself" appears to us as being at too *low* a level! However, although we do not find the definition of security requirements very consistent, the inclusion of assurance requirements (the "degree of confidence" required in the security mechanisms of a system) is important although we have not attempted to address it in this paper.

The Requirements Community

There have also been several relevant papers on the requirements community side. Heitmeyer [20] shows how the SCR method can be used to specify and analyse security properties, without giving the criteria for distinguishing them from other system properties.

A number of papers have focussed on security requirements by describing how they may be violated. For example, McDermott & Fox [38], followed independently by Sindre & Opdahl [50] and elaborated by Alexander [2], describe abuse and misuse cases, extending the use case paradigm to undesired behaviour.

Liu, Yu & Mylopoulos [37] describe a method of analysing possible illicit use of a system, but omit the important initial step of identifying the security requirements of the system before attempting to identify their violations.

van Lamsweerde and colleagues have written several papers on the subject. In the latest [55] he uses the concept of security goals, but does not define what he means by security requirement or give an example of one. Antón & Earp [5] use the GBRAM method to operationalise security goals for the generation of security policies and requirements, but also do not define security requirements.

Firesmith [15] defines security requirement as "any requirement that specifies a minimum, mandatory amount of security", which does not take us much further forward.

None of the above define what security requirements *are*. On the other hand, when discussing non-functional requirements, of which he regards security as one, Kotonya [30] defines them as "restrictions or constraints" on system services and similar definitions can be found in other text books. Rushby [46] appears to take a similar view, stating "security requirements mostly concern what must *not* happen". Using the Tropos methodology, Mouratidis et al [43] state that "security constraints define the system's security requirements". Our own view, elaborated in the remainder of this paper, is consistent with these definitions: that security requirements are most usefully defined as requirements for constraints on system functions.

1.2.2 The Importance of Security Requirements

We distinguish between the goals of stakeholders and the requirements of the system, as agreed by the customer[1]. Individual stakeholders may have different and conflicting goals, which need to be elicited by the requirements engineer. On the other hand the system's requirements must be free of conflicts, because the resolution of conflicts between goals is the job of the requirements engineer, not the implementer.

It is important to know what security requirements are, because the issue of their definition in actual applications is not trivial. Consider the description of a clinical information system in [4]. The report presents a view of the security goals of a Clinical Information System from the point of view of the doctors. It makes explicit assumptions that the doctors should have control of the system, while the administrators should be subordinate. It is well known that, in many health services, there is a power struggle between doctors and administrators. In a hypothetical system in which that power struggle has not been resolved, we can consider two hypothetical sets of candidate security requirements. In set 1, proposed by the doctors, some actions are considered legitimate for doctors, but prohibited for administrators. In set 2, proposed by the administrators, the situation is reversed; some actions that would have been legitimate by the standards of report 1 are security violations, and vice versa. It cannot be left to the

[1] As defined in [22] *IEEE Recommended Practice for Software Requirements Specifications.*

implementers to resolve conflicts between points of view; a requirements document must state unambiguously what is to be allowed or prohibited to whom; i.e. what are the constraints that are to be imposed on the use of functions of the system. Only then can we analyse the requirements for misuses or abuses.

1.3 Outline of Paper

The remainder of this paper is organised as follows. Section 2 introduces the artefacts and places them in a framework: 2.1 discusses artefacts generally, and distinguishes between core and support artefacts; 2.2 illustrates the core security artefacts and their dependencies by means of a diagram (figure 1); 2.3 discusses the process implications of the dependencies; and 2.4 briefly introduces problem frames, the notation with which we illustrate the relationship between system requirements and software specifications.

Section 3 is a case study, which uses the framework concepts and shows how they are applied. It is in four parts: 3.1 introduces the application; 3.2 shows how its system security requirements are derived from security goals; 3.3 shows alternative designs by which the problem frame, consisting of a software specification interacting with its surrounding domains, satisfies the security requirements; and 3.4 reviews the case study.

Section 4 expands on our introduction by discussing the main artefacts in the light of the case study: in sections 4.1 – 4.3 security goals, security requirements and their analysis are discussed; sections 4.4 – 4.7 cover covert channels, the need for a multi-domain approach, security functions and security policies.

This paper has inevitably left open many issues, and section 5 considers some of them: security in insecure systems; the need for a taxonomy of constraints; data-driven security requirements; specification notations; and the place of risk analysis in this framework. Section 6 concludes the paper.

2. The Framework

2.1 Artefacts

Requirements engineering rightly concentrates on deciding *what* is to be done, before deciding *how* to do it. This paper follows the spirit of requirements engineering by concentrating on the "what" before the "how". In this paper we use the term **artefact** to describe the "whats", or objects, of security requirements engineering.

By artefact we mean any object that is created as part of the process of system development. Typically artefacts are documents, electronic or otherwise, but they could include physical models, test rigs, and other things. We make the distinction between core artefacts and support artefacts.

We assume that the system development process has recognisable stages, each of which produces artefacts that are successively closer representations of a working system. These are **core artefacts**. They are ordered in a hierarchy that describe the system, progressing from the most abstract to the final concrete working

system. This hierarchy is a hierarchy of abstraction, and does not imply the waterfall model of linear progress; progress may be made concurrently or iteratively, rather than in a linear fashion. Indeed, we have pointed out in an earlier paper [44] that requirements and architecture cannot be separated: the "Twin Peaks" concept.

At early stages core artefacts are typically documents or prototypes. The final core artefact is the working system itself, consisting of a combination of physical and software items.

Core artefact documents contain statements of two kinds, following Jackson's notation [28]:

- Statements that describe the assumed or given structure or behaviour of some aspect of the system or its environment. These are **indicative** statements characterised by the use of "is/are" for factual descriptions.
- Statements that describe the required structure or behaviour of some aspect of the system. These are **optative** statements, characterised by the use of "shall".

To give an example, the optative statement that a door *shall* be secure from intruders can be achieved in one of two ways[2]:

- A high security lock *shall* be installed on the door, or
- The existing door lock *is* secure.

The latter statement is a **trust assumption** [17], which is used to place a bound upon the extent of analysis that is considered necessary (see also section 2.4 below).

The core artefacts in which we have most interest in this paper are, on the mainstream requirements engineering side: goals, requirements, and the components and structure of the system architecture; and on the security engineering side: assets, threats and control principles.

Support artefacts are artefacts that help to develop, analyse or justify the design of a core artefact. They may represent formal analysis, informal argument, calculation, example or counter-example, etc. They are the by-products of processes whose aim is to help produce verified and valid core artefacts: either constructive processes which help create them, or analytical processes which test them, both internally (verified) and in relation to their senior artefacts in the hierarchy (valid).

We concentrate on core artefacts, because it is their production which drives the need for development and analysis processes from which support artefacts emerge. Nothing in this paper should be construed as doubting the importance of support activities and artefacts, but they are only important to the extent that they ensure the quality of the core.

---

[2] Plus indicative or optative statements about the door material, hinges, behaviour of people, etc.

Dependencies between Artefacts

In a hierarchy of artefacts, there are dependencies between artefacts. For example, an operationalised requirement is dependent upon a goal from which it has been derived, because alteration of the goal is likely to cause alteration of the requirement. We will call this kind of dependency **hierarchical dependency**.

There is also a reverse kind of dependency: **feasibility**. If it proves impossible to implement a system that satisfies all the optative properties of a requirements specification, then this will force a change in the goals or requirements; the higher-level artefact is dependent on the feasibility of the artefacts below it in the hierarchy.

Although the processes are not the main concern of this paper, the dependency relationships have an important implication for the structure of development processes:

- If an artefact is dependent upon the implementation of another artefact for its feasibility, then if the implementation is not feasible, there must be an iteration path in the process, back to the ancestor from its descendant.

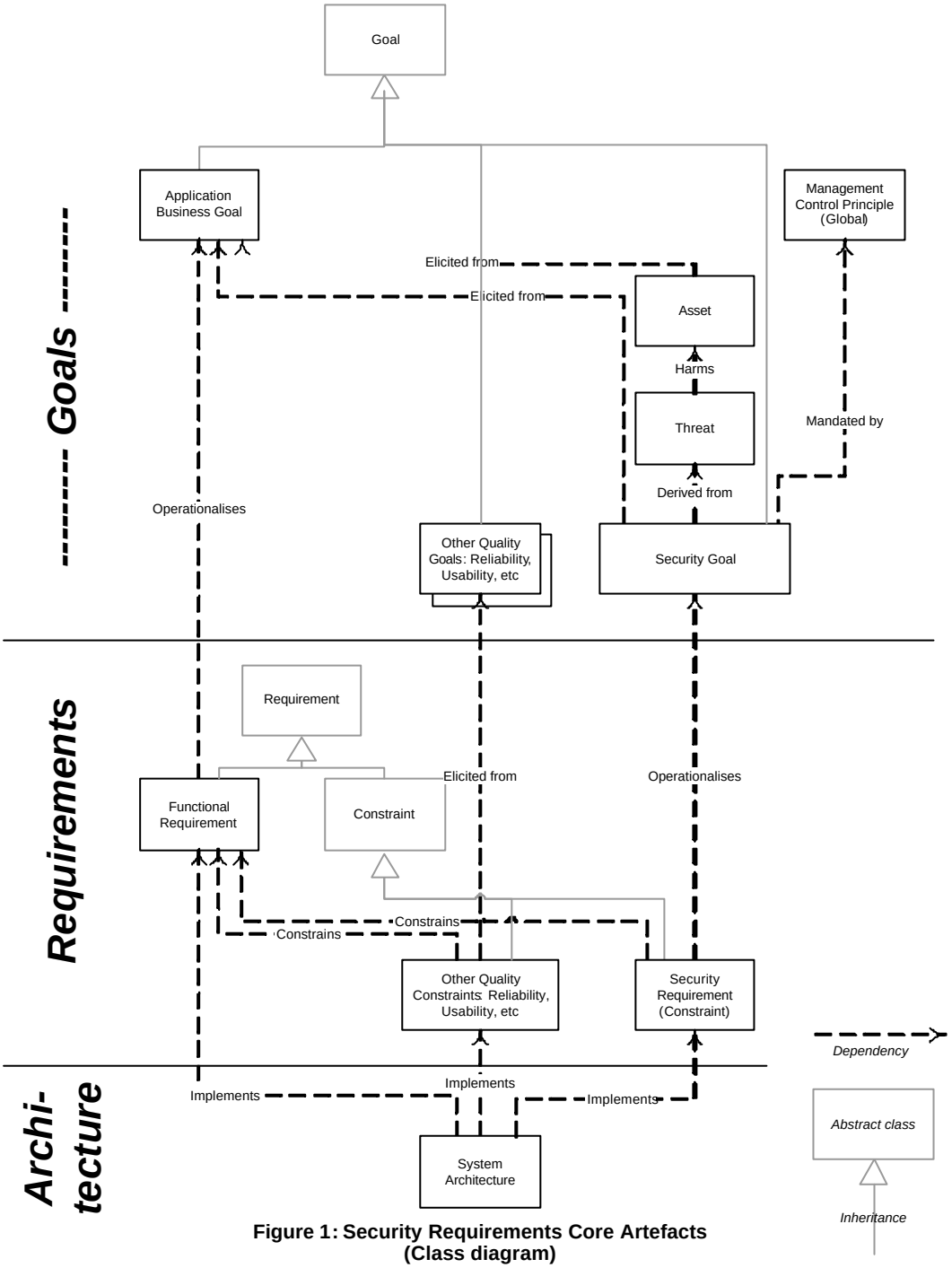

Figure 1: Security Requirements Core Artefacts (Class diagram)

2.2 The Core Artefacts Diagram

Our view of the core artefacts and their relationships is shown in Figure 1. It is a class diagram, with dependency associations. Some classes inherit from the abstract classes Goal, Requirement and Constraint, and these inheritance relations are shown in grey. The dependency associations are shown as black dashed lines, and are interesting because the dependencies drive the security requirements process.

2.2.1 Application Artefact Types

Three columns of application artefact are shown and, in addition, global management control principles:

- The first column shows the application functionality artefacts that would be needed even if security were not under consideration;
- The second column shows the additional application quality artefacts that are needed in order to provide qualities such as usability to the system;
- The third column shows the additional application security artefacts that are needed in order to provide security to the system;
- Management control principles apply globally throughout an organisation and provide constraints that would otherwise have to be derived repeatedly for each security risk analysis. Examples that are directly relevant to security are the principles of Least Privilege (no one shall have more privilege than needed for the performance of their duties) and of Separation of Duties (for important transactions, no single person shall be able to perform all parts of the transaction).

Application Functionality

This column represents the artefacts that are needed in a conventional life-cycle model in order to produce a working system. Beyond the structure imposed by Goals, Requirements and Architecture, we do not intend to place any constraint on the development process or method that is used.

Application Quality

This column represents the artefacts that are needed in order to provide qualities such as usability and performance to the system[3]. They are included in the consideration of security requirements, as it is typically not feasible to implement all the desired qualities of a system, and it may be necessary to trade off security against other qualities.

Application Security

This column represents the artefacts that are needed in order to introduce security into the system. They are briefly introduced here. More detail of the concepts can be found in risk management books such as Alberts & Dorofee [1].

- The relevant **assets** of a system are those assets of the organisation that could result in harm if the system were misused. They have a dependency upon the application business goals because the goals will determine which portion of the organisation's assets may be affected by the system.
- The type of **harm** that can happen depends upon the asset type, e.g.

[3] Security is generally considered to be a "quality", but it is considered separately because it is the focus of this paper.

Money: loss

Information: exposure, corruption, loss, etc

- The **security goals** of the system owner are derived from a combination of three different sources:

  The possible harm to assets;

  Management control principles;

  Application business goals, which will determine the applicability of management control principles, e.g. by defining those privileges that *are* needed for the application prior to excluding those that are not.

  Note that other legitimate stakeholders may have other security goals that conflict with these (see section 1.2.2 above); the set of relevant security goals may be mutually inconsistent, and inconsistencies will need to be resolved during the goal analysis process, before a set of consistent requirements can be reached.

  On the other hand, the goals of attackers are not considered to be a part, even negated, of the security goals of the system, and influence them rather obliquely. See 2.2.2 below

- The (primary) **security requirements** of the system operationalise the security goals by expressing them as constraints on the functions of the system (see 2.2.3 below). They are dependent on the definition on those functions, since they are constraints upon them. Like any other set of requirements, any inconsistencies need to be removed.

  It may not be feasible to implement these primary security requirements without additional functionality. In that case derived security requirements of the system are added. It may be necessary to revisit the system architecture, adding security functionality and/or modifying the existing security requirements.

2.2.2 Security is not Football

The goals of the system owner and other legitimate stakeholders are not directly related to the goals of attackers, because Security is not Football. It is not a zero sum game. In football, the goals won by an attacker are exactly the goals lost by the defender. However, security is different; there is no exact equivalence between the losses incurred by the asset owner and the gains of the attacker. To see this, look at two examples:

- Robert Morris unleashed the Internet Worm [52], causing millions of dollars of damage, apparently as an experiment without serious malicious intent. The positive value to the attacker was much less than the loss incurred by the attacked sites.

- Many virus writers today are prepared to expend huge effort in writing a still more ingenious virus, which causes no or trivial damage (screen message "You've got a Virus"). Here the positive value to the attacker, judged by the amount of effort he is prepared to invest, is much greater than the loss incurred by the attacked site. Generally, there is no simple relationship

between the gains of a virus writer and the losses incurred by those who are attacked.

The consequences of security not being a zero sum game include:

- The evaluation of possible harm to an asset can be carried out without reference to particular attackers, with the caveat that, if the impact on the defender depended on the particular attacker, then the individual attacker would need to be considered when setting security goals. In the approximate world of risk analysis, this is unnecessary in practice.
- Consideration of the goals of attackers cannot be used simply to arrive at the goals of a defender to prevent harm, i.e. their security goals. In view of the point above, it is not necessary, either.

2.2.3 Security Requirements

We define security requirements to be the constraints, on functional requirements, that are needed to achieve security goals.

A simple example of such a constraint is:

> The system shall not provide Personnel Information except to members of Human Resources Dept.

Note that the constraint ("shall not … except to …") is secondary to the function ("provide Personnel Information"); it only makes sense in the context of the function.

There may also be temporal constraints:

> The system shall not provide Personnel Information outside normal office hours;

and complex constraints on traces:

> The system shall not provide information about an organisation to any person who has previously accessed information about a competitor organisation (the Chinese Wall Security Policy, [9]).

Availability requirements will need to express constraints on response time:

> The system shall provide Personnel Information within 1 hour for 99% of requests.

We note that this differs only in magnitude from a Response Time quality goal, which might use the same format to require a sub-second response time.

This paper does not claim to provide a complete taxonomy of constraints nor, since this is a framework rather than a process or method, does it attempt to mandate a specification language. There are discussions of some of the issues in sections 5.2 and 5.4 below.

2.2.4 Goals, Requirements and Architecture

The horizontal view shows three phases of development of the artefacts: Goals; Requirements; and Architecture. We indicate briefly here what we mean by each of these terms:

- Goal: something that any stakeholder wishes to achieve or avoid.
- Requirement: a functional requirement, which describes a function to be provided by the system in terms of an operation that can be used by an agent; or a constraint on a functional requirement. The constraint is an expression of a security or other quality requirement, e.g., performance, usability, etc.
- System architecture: a description of a means of achieving requirements, in terms of the interactions between relevant domains. The problem frame approach that we use for our case study is one such description; it describes how a software specification in a structure of domains satisfies system requirements.

2.3 Process Overview

The dependencies among the core artefacts influence the possible ways in which a security requirements process can be constructed. The process diagram is shown as figure 2; the following points should be noted:

- There are two columns, corresponding to the "normal" application development process and quality goals, and the development of security requirements. It is assumed that no explicit activity is needed to elicit the organisation's control principles, and these can therefore be fed directly into the Identification of Security Goals activity.
- Lines coming out of the bottom of an activity box indicate the successful completion of an activity and, except for Validation boxes, carry with them a core artefact into the next activity.
- Lines coming out of the side of an activity box denote failure and imply the need to iterate back up the process in order to revise an earlier activity. Failure can be one of two kinds:
    - It has been found to be infeasible to create a consistent set of the artefacts that are constructed by that activity, or
    - Validation of the artefacts against a higher level, e.g. validation of security requirements against security goals, shows that they fail to meet their aims. This occurs if it has not been possible to construct a satisfactory correctness argument or a vulnerability has been found (see section 4.3.2 below).

  The iteration may "cascade" upwards if the architecture is not feasible without a revision of the business or security goals.

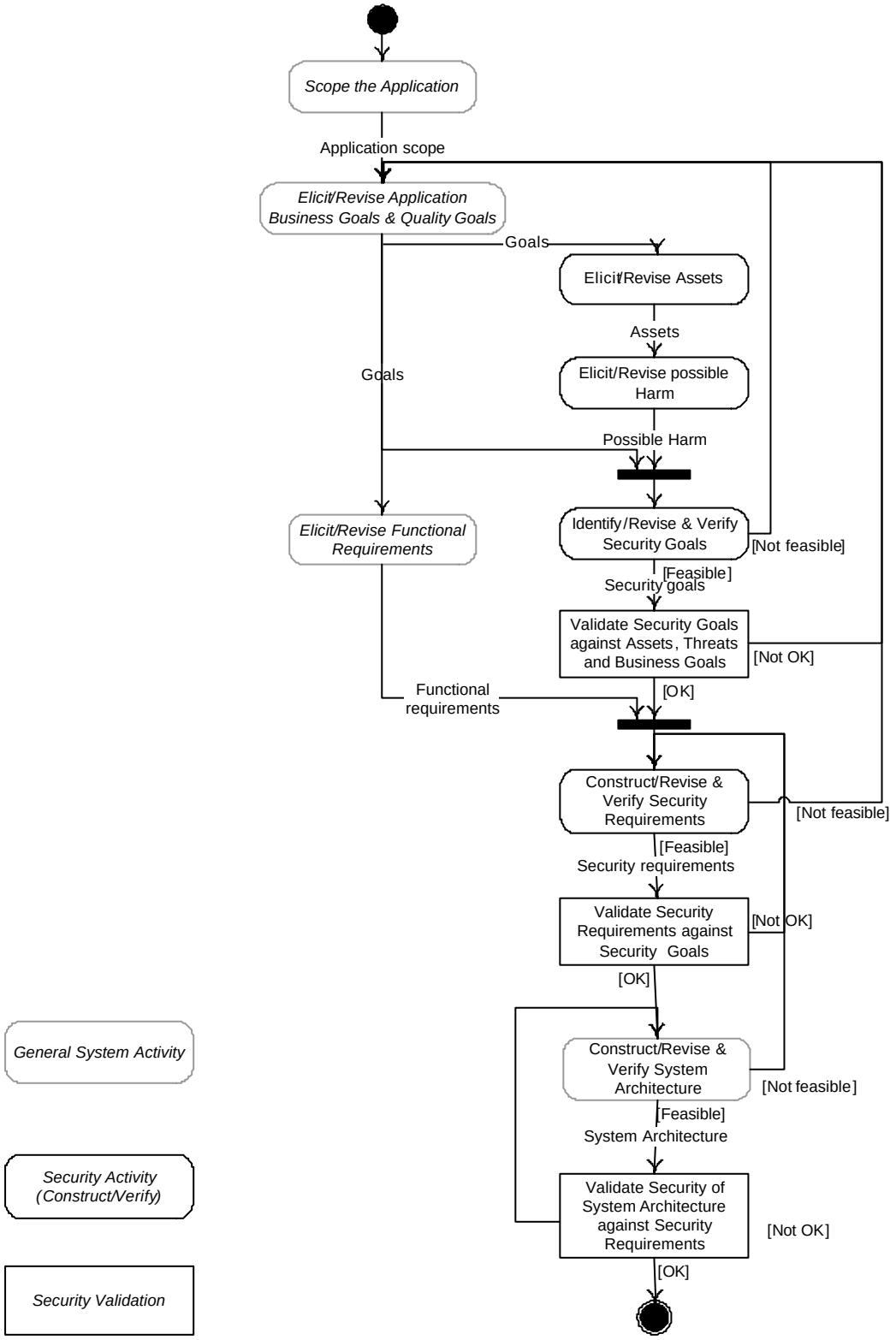

Figure 2: Security Requirements Process Overview (Activity Diagram)

2.4 Problem Frames

We use a notation based on Jackson's Problem Frames [28] as a tool in our case study (section 3.3 below) to help derive software security requirements. This section presents some background information on problem frames. We do not

claim that this is the only possible approach, but it is the most useful one of which we are aware.

In common with other approaches, such as the Concert framework [56], the Problem Frames approach recognises that requirements can exist at several levels of abstraction. In this approach the focus is on the relationship between requirements on the real world system, which Jackson calls simply "requirements", and requirements on the software, which he calls the "specification". Where there is any ambiguity, we refer to the former as system requirements" and the latter as the software specification.

When using problem frames, a requirements engineer describes problems by describing the interaction of domains. The notation describes the domains in a problem along with the interconnections between them. For example, in our case study (section 3.1 below) the requirements elicitation process for an automatic door produces the need for a machine[4] to display personnel information on request, with requirement *the system shall display personnel information to the requestor*. Figure 3 shows the initial problem description. There are only two domains: the Personnel Information Machine, conventionally shown with two bars in the box to denote that it will be the subject of the software specification; and Person, containing people. The requirements are shown in the oval with a dashed line. The connection between the domains is shown by a line between their interfaces. It is labelled **a** to give a reference to a description of the interaction phenomena between the domains. In this case there are two:

P!{Payroll#} and PIM!{PersonInf(Payroll#)}

P! and PIM! denote that the phenomena are initiated by the Person and machine domains respectively, followed by the contents of the phenomena; P can supply a payroll number, and PIM can return the personnel information for that payroll number.

Correctness Arguments

A major aim of this approach is to show, by means of a *correctness argument*, that the problem frame's domains, interactions and specification will satisfy the system requirements. This argument may be both positive and negative:

The positive argument will attempt to demonstrate why the problem frame satisfies the requirements. However, it is often impossible to produce a complete formal proof of this.

The negative argument tests the problem frame by searching for contradictions to the argument. In the case of security requirements these are called *vulnerabilities*. A vulnerability is discovered, as described by van Lamsweerde [55] in a different context, by negating the requirement and then attempting to show that the

[4] In our view of problem frames the machine may be an abstract entity; for example, the Personnel Information Machine and Credentials Administration Machine that are discussed in 3.3.3 might both be implemented in the same physical computer.

negation of the requirement can be satisfied. They are discussed further in section 4.3.2 below.

Further Notation

We have found it necessary to extend beyond the standard notation of Problem Frames in order to describe security requirements adequately, in two respects: trust assumptions; and a causal specification language.

We introduce trust assumptions [17] in order to make explicit the assumed properties of domains on which the satisfaction argument depends. They are denoted by ovals with irregularly dashed lines. Examples are found in our case study, e.g. section 3.3.2 below.

The use of a causal specification language is needed because Jackson's book on Problem Frames uses state-machine diagrams for the specification of required software behaviour. Although guards on transitions can be shown in this notation, they are guards whose satisfaction is *sufficient* to enable behaviour. Security constraints need stronger guards also, whose satisfaction is *necessary* to enable behaviour. We are therefore using, as our specification language, a causal notation derived from our earlier notation [41] to describe a machine or domain specification. The relationship between the interactions described above is specified to be:

P!{Payroll#} **shall cause** PIM!{PersonInf(Payroll#)}

shall cause prescribes that the first interaction shall always result in the second one. The other element of the notation that we use in this paper is shall prevent, which prescribes that the first interaction shall always prevent the second one from occurring. It overrides shall cause.

3. Case Study

This case study, of a Personnel Information display system, is used to illustrate the framework that we have set out above and to bring out further issues for discussion. It is unrealistically simple, to enable points to be illustrated easily.

We use an informal notation for goals, and a notation for requirements that is based on Problem Frames, as discussed in section 2.4 above.

3.1 System Business Goals and Functional Requirements

We assume that the business goals have been elicited and that there is only one goal:

G1: Provision of people's personnel information to them.

We assume that initial requirements have been elicited and that there is only one functional requirement:

REQ1: On request from a Person (member of People), the system shall display personnel information (PersonInf) for a specified payroll number (Payroll#) to that Person.

The problem frames diagram shown in Figure 3 is a first attempt at representing the requirement. It shows a Person interacting with the Machine, as a result of which specified personnel information is displayed.

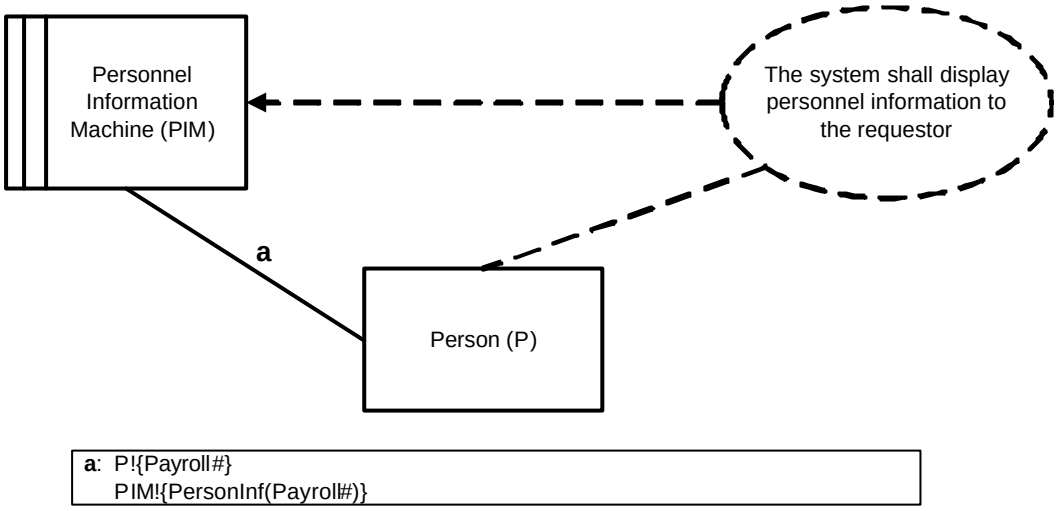

Figure 3: Initial Problem Description

The specification of the behaviour of the PIM is:

1. P!{Payroll#} **shall cause** PIM!{PersonInf(Payroll#)}

3.2 Case Study Part I: from System Security Goals to System Security Requirements

In this section we will first discuss how to derive security goals and then how to obtain security requirements for this application.

3.2.1 Security Goals System Security Risk Analysis (Assets & Harm)

Asset Identification

Examination of the business goal G1 reveals only one relevant asset: *personnel information*. Other assets would need to be considered in a fuller example, including: tangible assets such as money, products, or the computers themselves; and intangibles such as reputation.

Harm Identification

There are at least the following types of possible harm to personnel information:

H1: Unauthorised disclosure.

H2: Unauthorised alteration.

H3: Unavailability.

Security Goals

Having identified the relevant harm, we need to take the step of stating security goals for this application, the prevention of relevant harm:

SG1: Confidentiality: Prevent unauthorised disclosure of personnel information.

SG2: Integrity: Prevent unauthorised alteration of personnel information.

SG3: Availability: Ensure availability of personnel information.

3.2.2 From Security Goals to Security Requirements

We have now derived and valued the organisation's security goals for a particular kind of asset. These goals need to be related to the possible behaviour of the system, i.e. its functional requirements, in order to be expressed as security requirements, i.e. constraints on those functional requirements.

Each security goal needs to be examined for possible relevance, and then the goals must be operationalised to derive constraints on functional requirements. Two separate tasks have to be carried out:

- Use domain knowledge to transform the entities described in the security goal into entities described in the functional requirement. In this case the task is trivial, as the security goals directly refer to personnel information.
- Transform Confidentiality, Integrity and Availability into constraints on the operations that are used in functional requirements.

Security Requirements for Confidentiality

In order to derive constraints for Confidentiality, we need to know who is authorised to access personnel information. In this case we assume that only members of Human Resources (HR) Department are so authorised. We can therefore state the following constraint:

> REQ1/SR1: The machine shall display personnel information only to members of HR Dept.

This is the application's only security requirement for Confidentiality

Security Requirements for Integrity

Integrity is about ensuring that assets are not altered without authority, but none of the operations of the Personnel Information Display System alter information,

so there are no constraints on operations that are derived for this security goal. This application has no security requirements for Integrity.[5]

Security Requirements for Availability

An Availability goal will be translated into temporal constraints on every functional requirement, but for brevity we do not pursue it here.

Correctness Argument

Although we are not working with problem frames in this section, it is till appropriate to attempt to make a correctness argument, that our security requirements will satisfy the security goals. At an informal level, this has been carried out by the paragraphs above. At a practical level, the analyst may wish to question the step from "authorised" to "members of HR Dept". Is this a necessary and sufficient set of people? What about senior managers? Should *all* members of HR Dept be authorised? For the purpose of this example, we will assume that the requirement is correct.

3.2.3 Security Requirements Model

The security requirements and their context for this system are shown below in figure 4. This is a simplified instantiation of the meta-model of figure 1; only security goals are shown.

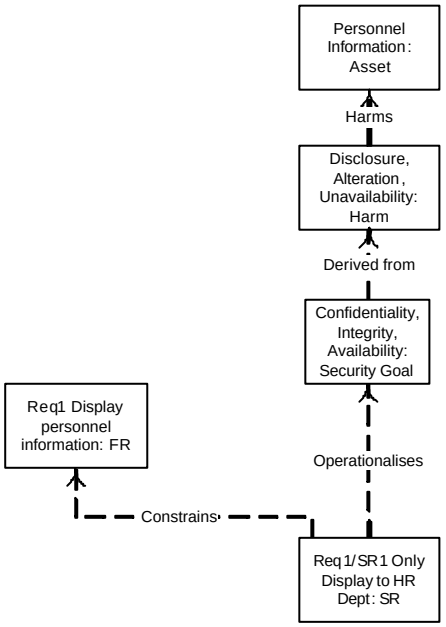

Figure 4: Security Requirements for Payroll Information Display System

[5] We assume that there is a requirement for accuracy regardless of whether the application has any security goals, and do not consider accuracy in this paper.

3.3 Case Study Part II: from System Security Requirements to Software Security Specifications

In this part we will examine how a Software Security Specification can be derived from the derived System Security Requirements. Note the difference between requirements REQ1 (section 3.1) and REQ1/SR1 (section 3.2.2):

- REQ1 is a functional requirement upon the behaviour of the Machine, and will therefore be (in transformed form) a part of the software specification.
- The security requirement REQ1/SR1 is a constraint that could be applied either in the Person or the Machine domain. It can be achieved either by: restricting membership of the Person domain to members of the HR Dept; or by ensuring that the Machine domain is able to identify whether or not a request has been submitted by a member of HR Dept, rejecting it if not.

A design decision is therefore needed before we can deal with the impact of the security requirement REQ1/SR1 on the Machine.

3.3.1 Introducing the Security Requirements into the Problem Frame

The problem frame diagram in Figure 3 does not show how the security requirement REQ1/SR1 fits into the picture. There are two issues that have to be resolved by system design decisions:

- Where should the security constraint be implemented – in the Person domain, the Personnel Information Machine domain, or both?
- How should we introduce meaning to "members of HR Dept"?

At the risk of repetition, we stress that this is a *system design* decision, in the domain of system engineering. It may or may not have an impact on the problem frame diagram, in any of the following ways:

- Adding constraints to the machine specification or the properties of other domains
- Altering the interactions between domains
- Introducing additional domains to the problem.

We will first deal with the first question, which is relatively straightforward, before discussing the issue of representation of identity in answer to the second one. What follows is an informal architectural design exercise for the system.

3.3.2 Constraining the Person Domain

We could solve the security requirement by requiring the Person domain to contain only members of HR Dept, which states that no one but HR Dept members can interact with the machine. Although this is not a detailed design exercise, we illustrate some ways in which this might be achieved:

- By physically isolating the machine from any network and using physical or procedural access control (either a lock or a guard to prevent unauthorised use of the machine)
- By using a network firewall to prevent access from any terminals except those in HR Dept, for which physical or procedural protection is in force.

If we make either of these design decisions, then REQ1/SR1 is satisfied outside the machine domain, and there is no need to consider REQ1/SR1 in the machine's software specification. The Problem Frame diagram in Figure 5 presents a solution using a locked room approach.

Figure 5 illustrates one of the principle uses of *trust assumptions*, limiting the scope of an analysis. To ensure that REQ1/SR1 is truly discharged, the analyst would in theory be required to examine how access to the room is limited to HR Dept members, bringing structural integrity and key management into the problem. The trust assumption obviates this need by simply stating that the analyst trusts the stated indicative property (that access is limited to HR Dept members) is true, thereby justifying changing the domain membership from *People* to *HR Dept Members*.

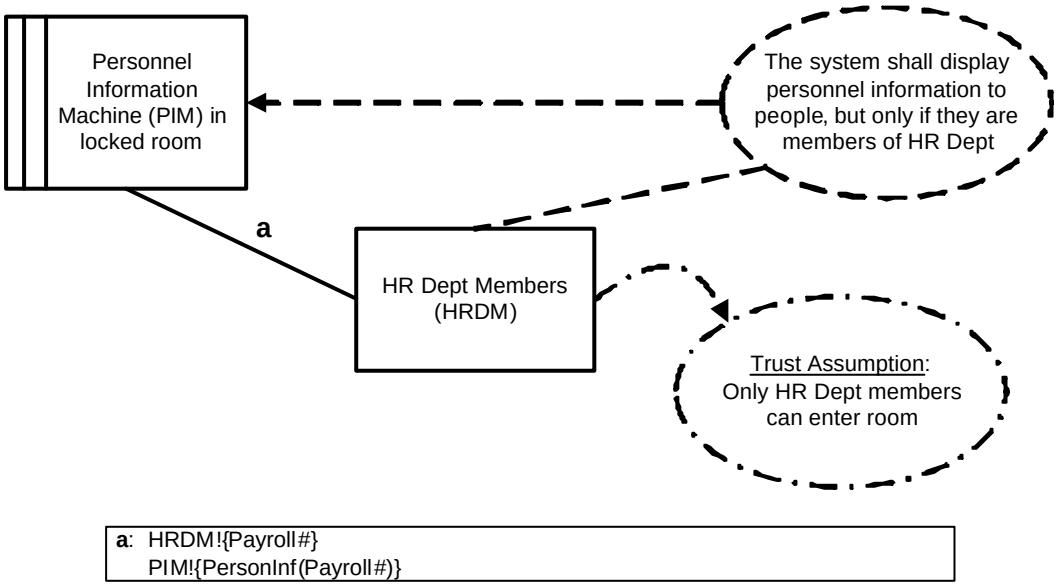

Figure 5: Design with Locked Room

The specification of the behaviour of the PIM is identical to that of the original problem description (section 3.1 above), because the PIM has no part in implementing the security requirement:

1. HRDM!{Payroll#} **shall cause** PIM!{PersonInf(Payroll#)}

Correctness Argument

The positive correctness argument for meeting the security requirement is straightforward:

- The only way to use the machine is by entering the room
- The only people who can enter the room are members of HR Dept, thus satisfying the security requirement.

The negative argument tests each of these steps in turn. Are there any vulnerabilities in the statements that:

- the only way to use the machine is by entering the room? In this case the analyst was sufficiently confident that a trust assumption was not thought to be necessary.
- the only people who can enter the room are members of HR Dept? Here the analyst had sufficient doubt about it that a Trust Assumption was attached to this statement. The risk behind the trust assumption will need to be accounted for during a security risk analysis exercise.

3.3.3 Security Constraint in the Machine

We now consider the issues involved in implementing the security constraint in the machine, instead of (or as well as) in the People domain. It will now be necessary to include some information relating to the identity of the requestor in the interaction between P and PIM. There are several possibilities for changing the P! and/or the PIM! interactions.

To illustrate the number of design possibilities, here are two that we reject:

- Encrypting the information in the PIM! interaction so that it can only be understood by a member of HR Dept. This has the disadvantage of needing a lot of functionality in the Person domain and, without some elaboration, is inflexible.
- Include a constant password (known to all members of HR Dept) in the P! interaction, and "hard code" knowledge of the password in the Machine. The Machine is required to refuse the request unless the password is correct. This is cheap to implement, but suffers from the obvious vulnerabilities of shared and unchanging passwords.

Instead we will use a simple version of user identification and authorisation in which, with each request, the Person provides a User Id and credentials. UserId is the claimed identity of the person who submits the request. Credentials are authentication information, such as a password, that provides assurance that the interaction has actually been initiated by the person who is identified by UserId.

This design decision is attractive for a variety of reasons, including:

- Use of a personal identity makes it possible to provide Accountability (which requires verification of who has carried out an action), which is often an organisational goal.

- Use of a user / credentials combination is familiar, implementable in standard ways, relatively cheap and adequately secure for many purposes[6].

The original requirements are unchanged, but there are two additional functions, which we assume are already in existence, for giving people their UserId and credentials, which are stored in a Credentials Store.

> FUN1: Administrators shall give UserId and credentials to members of the HR Dept.
>
> FUN2: Administrators shall store UserId, credentials and HR Dept affiliation in the credentials store, for members of the HR Dept.

Correctness Argument

Since we have changed the problem frame, it is necessary to iterate the correctness argument, that these requirements satisfy the system goals. However, since the security requirement is unchanged, we do not need to do this here.

PIM Problem Frame

The design is shown in figure 6. Each time a request is submitted, PIM submits the UserId and credentials for validation, and only fulfils the request if they are valid.

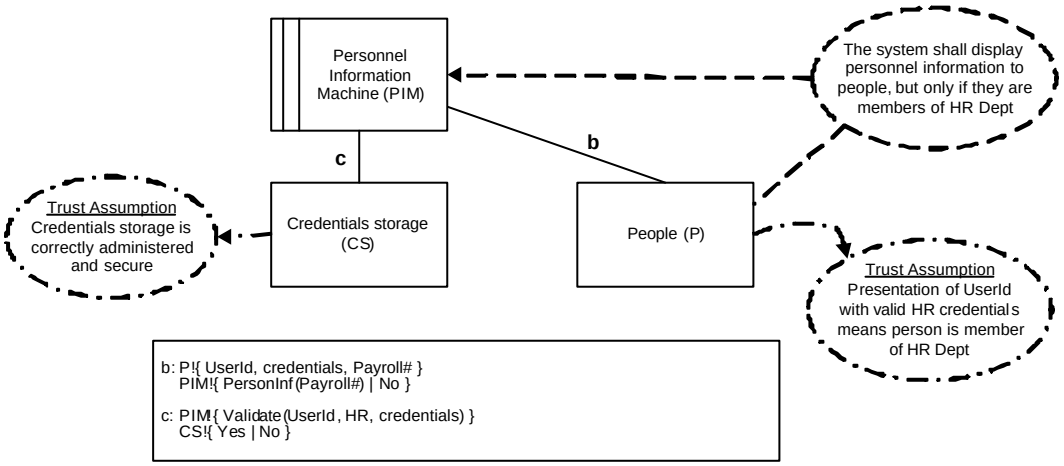

Figure 6: Design with Authentication

The specification of the behaviour of the PIM system is now as follows:

1. P!{ UserId, credentials, Payroll#} **shall cause**
   PIM!{ Validate(UserId, HR, credentials) }
2. **if** isValid(UserId, credentials))
   PIM!{ Validate(UserId, HR, credentials) } **shall cause** CS!{Yes}
   else PIM!{ Validate(UserId, HR, credentials) } **shall cause** CS!{No}
3. CS!{Yes} **shall cause** PIM!{ PersonInf(Payroll#) }

[6] Of course it also has certain well-known vulnerabilities, which would need to be considered in the risk analysis.

4. CS!{No} **shall cause** PIM!{ No }

The truth or falsity of the isValid predicate is determined by the contents of the Credentials Store.

Correctness Argument

The positive correctness argument for meeting the security requirement is as follows:

- The only way for a person to use the machine is by entering a request including UserId and credentials
- The only people who can present a UserId with valid HR credentials are members of HR Dept
- The machine will not satisfy that request without first validating the credentials by an interaction with the credential store.
- The credentials store will only return Yes if the UserId is for a member of HR Dept and has valid credentials.
- The machine will not satisfy the request unless the answer is Yes.

Again, the negative argument tests each of these steps in turn, highlighting those that are in doubt by means of trust assumptions.

Credentials Administration Problem Frame

Since the administration of identities and credentials has entered the problem, our problem frame context needs to be expanded.

Figure 7 shows how credentials are administered, showing that a human Administrator gives a UserId, Dept affiliation and credentials to People, and then provides the same information to the Credentials Administration Machine (CAM). This causes the CAM to store the information in the Credentials Store.

The specification of the behaviour of the CAM machine is as follows:

1. A!{create(UserId, Dept, credentials)} **shall cause** CAM!{store(UserId, Dept, credentials)}

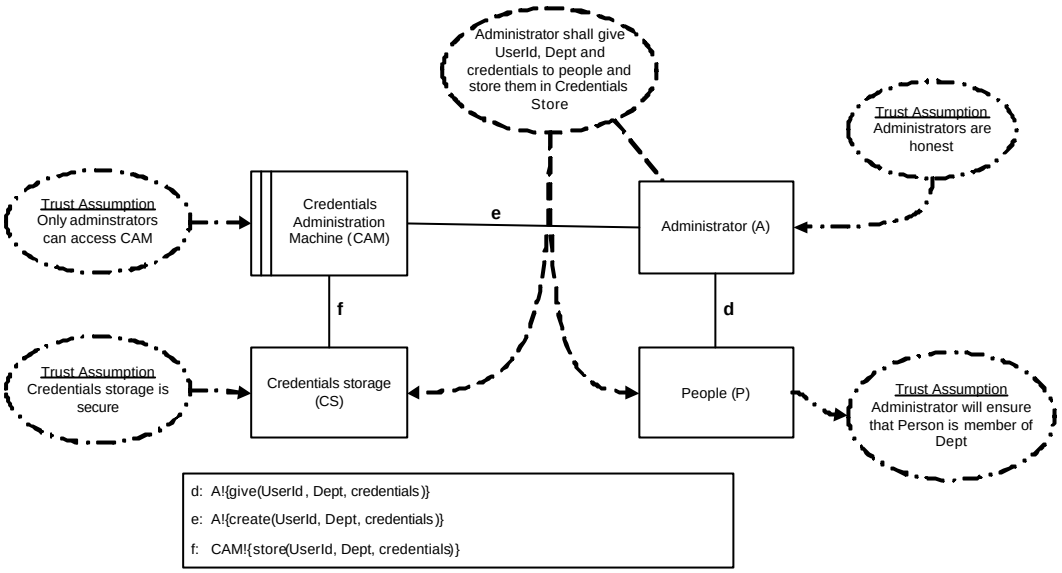

Figure 7: Administration of Credentials

3.4 Review of the Case Study

What steps did we go through in our attempt to define the requirements for a workable secure system? We can identify the following:

- An initial definition of system requirements and problem frame in section 3.1
- A revised definition of the system requirements to include security requirements, in section 3.2.2
- Introduction of the security requirements into the problem frame in section 3.3.1, using the UserId approach, which is infeasible with the original Problem Frame
- A solution, using the UserId approach, in section 3.3.3. This was infeasible with the original problem frame. The problem frame needed modification.
- Since the problem frame had changed, it was necessary to iterate the correctness argument, that the requirements met the goals, although this was trivial in this case
- This finally enabled us to define a software specification that, given the trust assumptions, could be argued to satisfy the system requirements.

4. Discussion

A number of issues have arisen in the course of expounding our view of security requirements artefacts.

4.1 Security Goals

4.1.1 Characteristics of Security Goals

An organisation's security goals have some characteristics which make them hard to manage:

- They cannot immediately be discharged by the specification of requirements, but have to be re-interpreted at each iteration of the design.
- They may interact with each other.

Security Goals are not Discharged by Security Requirements

At every iteration between requirements and design, whenever a new functional requirement is introduced that requirement must be evaluated against the security goals and appropriate security requirements introduced. One cannot assume that the existing security requirements are sufficient in the presence of new functional requirements.

For example, although some security requirements may be necessary to achieve the Confidentiality goal, they may not be operating in isolation. We will assume that there is also an Availability goal to be achieved, and that one of the means of achieving Availability is to perform regular data backups. A Backup functional requirement will therefore be introduced at a later iteration of our requirements. This implies in practice that a copy of the information exists that can be read using a function that was not defined in our original requirements. Unconstrained use of this function can violate the Confidentiality goal, and therefore there will need to be a security requirement that constrains its use. At a still later stage, an engineer's access to the Machine for maintenance purposes can also provide access to the information, using yet another function, which generates yet further security requirements.

The original security goal has not changed, but at each iteration of the requirements, when additional functions are introduced, additional security requirements to constrain the use of those functions may need to be added.

Security Goals Interact

Security goals interact. For example, it might be decided to introduce an encryption function in order to achieve Confidentiality. This is a new function for the system, and its use must be evaluated against all the security goals. One of those goals is Availability, and analysis shows that Availability is threatened by the loss of a secret key; our solution to confidentiality has undermined Availability. Therefore further measures need to be taken to ensure that the Availability goal is still met, either by ensuring that the secret key is always available or by reconsidering the design decision.

4.1.2 Are All Security Goals Negative?

Most security goals should be represented as the avoidance of harm. We have sympathy with the view of van Lamsweerde in [55], where only Confidentiality is described as an Avoid goal, with Integrity and Availability being Maintained and Achieved. However, it is necessary to consider the amount of possible harm in order to decide how important each goal is.

We are leaving it as an open question at present whether there are other application security goals, which do not need to be related to the possibility of harm to assets. For example:

- Is Anonymity a separate goal, or is it better regarded as a means of achieving a goal such as Confidentiality?
- Should provision of the Provenance of information, to help judge the amount of reliance that can be placed on the information, be regarded as a separate goal?

In neither case do we have a definitive answer.

It is clear that other security goals *can* be stated directly, if they arise from an organisation's management control principles, derived from previous experience of the need for security. For example, the Principle of Least Privilege – that no one should have more authority than is needed to carry out there duties – may have been adopted generally by an organisation. It can be used directly to derive system security requirements without the need for a detailed consideration of possible harm. However, the lack of a "value" – an assessment of the amount of harm that it prevents – for a goal of this kind makes it more difficult to manage when trade-offs become necessary.

4.1.3 Derived Security Goals

Where do security goals come from? The obvious source is as described above, from threats of harm to an organisation's assets, e.g. a bank's goal not to lose its own money.

However, they may arise indirectly from other goals. An example of this is a bank's goal to maintain a good reputation. A necessary condition is that it should be able to demonstrate its commitment to protect its customers' money, in addition to its own. Therefore there is a derived goal – protect customers' money – which depends upon the bank's Reputation goal. So, eliciting security goals cannot be done from a narrow security perspective; all of the organisation's main goals have to be taken into account.

4.2 Security Requirements

4.2.1 Why Define Security Requirements at all?

As remarked in the Introduction, there seems to be a curious reluctance in previous published work to define any explicit representation of security requirements. We have already made the point in motivating our work (section

1.2.2 above) that their representation is important because of the possibility of conflict between stakeholders. In addition, since traceability of requirements is essential, we need to be able to refer to each individual security requirement.

If we accepted that it is necessary to be able to refer to security requirements, why have we chosen to define them as constraints? We do not claim to be *correct* in defining security requirements as constraints on functional requirements; we are proposing a software engineering approach, not carrying out scientific research. Our reasoning for proposing this as a *useful* definition is as follows:

- Requirements specifications, in general, describe the functions (or operations or services) to be provided by a system.
- It is clearly desirable for the specification to describe security requirements in a way that enables them immediately to be related to the functions.
- Constraints upon functions are a natural way to do this.

Other candidate forms for security requirements, which we have rejected, are:

- *Security goals*. Security goals are necessary as a starting point, but they are more abstract than functional requirements, and may conflict. If designers were only given security goals to work with, it would be necessary for them to carry out further work that belongs in the domain of the requirements engineer: deciding how the security goals should be operationalised in the requirements; and resolving conflicts when necessary.
- *Security functions*. A security function such as encryption is part of the solution, and the specification of security requirements in terms of security functions may lead to a non-optimal and/or an incomplete design.

It appears to us that, in order to ensure that requirements engineers and system designers each work within their appropriate limits, the appropriate boundary between security requirements engineering and security design is provided by our proposal.

Why a Software Security Specification Cannot Be Considered Alone

We are insisting that security requirements must be regarded as a systems engineering problem, and that software security cannot be considered on its own. This contrasts with Michael Jackson's explicit focus [28] on the computer and its software. There are several reasons for our approach, which needs a broader approach than his because the concerns are wider:

- Security goals, unlike functional goals, cannot be discharged by the specification of a suitable constraint or function; they must be considered at every iteration of the development activity.
- Security analysis needs to consider several domains simultaneously.
- Security requirements may constrain domains other than the machine.
- Security principles, hard-won by experience, require a systems approach.

4.2.2 Residual Security Requirements?

It is often stated that security is only as strong as its weaknesses, and it is therefore important for it to be complete. We must therefore ask whether, by specifying constraints on the functions that are to be provided by an application, we have produced a complete set of security requirements?

If we assume that we have a complete statement of the organisation's security goals, and have taken them all into account in deriving the constraints, then the answer is Yes. It would be tempting to include another requirement for an application: "and nothing else must happen" so as to ensure that the designers do not assume that they need do nothing else to ensure a secure system.

However, we have no means of expressing what we mean by "nothing else", so we are stating a general goal, rather than providing a specification of security requirements to a designer, and there is no point is adding "and nothing else" to the security requirements. However, we should recognise that the security goals have not been discharged by the specification of constraints on system functions; that is a necessary, but not sufficient, condition for the satisfaction of the security goals by the implementation.

This is a proper separation of concerns. To take an example from the case study, the organisation has a security goal of Confidentiality of Personnel Information. If it is to achieve this goal, then it will have to state security requirements on a number of activities and domains, including securing the engineer's hardware interface and communications infrastructure. By proposing a new application we have introduced some additional functions by which the security goal could be breached and the security requirements for the new application is properly and completely expressed as appropriate constraints on the functions.

Two issues arise:

1. Additional functionality introduced in the design;
2. The properties of the machine within which the application is embedded.

Additional Functionality Introduced in Design. When the system requirement results in a design, then the implementation of that design may result in additional functions, adding ways in which security goals could be violated, e.g. through the engineer's hardware interface or through a hacker intercepting communications. In order to achieve the organisation's security goals, additional operational security requirements will need to be derived, from the security goal, for the engineer's system and the communications infrastructure.

Machine Properties. The application will almost always be run on an existing operating system and using existing utilities. This software will provide additional functionality, and therefore expose the application to additional risks. Analysis of these risks is essential, and the security requirements for the application will need to take them into account, but the risk analysis of this software is independent of, and prior to, the development of security requirements for the application.

So, our conclusion is that, if the analysis has been done thoroughly, the security constraints do constitute the complete set of security requirements for the application as far as it is understood at that point. However, the organisation's security goals are never discharged until there is an implemented system, and the security goals must be revisited whenever additional functionality is proposed during the course of development.

4.2.3 Security Requirements and Security Properties

Security "properties" are often referred to, especially in formal specifications; for example, Heitmeyer [20] gives examples, some of which are predicates on state, and others predicates on the relationship of successive states. We need to consider how they fit into this framework. Most security properties are expressed in terms of constraints on traces of the behaviour of a system, and this fits in very well with our own view of security requirements as constraints on the operations of a system. It emphasises that realistic security requirements are likely to be far more complex than the simple constraints that we have used in this paper.

Some security properties may, of course be expressed at a lower level than system requirements, and it will only be possible to discuss them at that lower level.

There are security properties, such as "no covert channels", which do not conform to the constraint model. They are like the "and nothing else must happen" requirement of section 4.2.2 above, and we take the same view, that they are not a concern for system security requirements, but must be addressed at a design or implementation level.

4.2.4 Quality Constraints and Security Requirements

A set of requirements can contain many constraints on functions, derived from a variety of goals, e.g. constraints arising from all the other quality goals that are relevant to a system, such as performance and reliability. If we examine a constraint, such as the following, how do we know that it is a security requirement?

> The machine shall not display Personnel Information except to members of HR Dept.

The answer is, we cannot identify this as a security requirement from its contents alone. Why not? Consider a hypothetical Personnel Information Display System in an environment in which the honesty and discretion of all users has never been in any possible doubt, so that the organisation has no need of any security goals at all. However, it has a goal of Comprehensibility, and the Personnel Information is so difficult to understand that it is considered essential for all information to be interpreted by members of HR Dept, rather than being directly available to all users. Then, although there is no Confidentiality goal, the constraint has been derived in order to satisfy the Comprehensibility goal, and it would be reasonable to call it a comprehensibility requirement, not a security requirement.

From this we conclude that any particular constraint is identified as a security requirement by the source goal from which it is derived, and not from its contents.

Of course, a constraint can be derived from multiple goals and therefore belong to multiple categories.

4.3 Analysing Security Requirements

Although this paper is not about analysis, some comments about its role are needed. We make the conventional distinction between verification and validation:

- Verification is about the analysis of an artefact on its own, ensuring its completeness and internal consistency;
- Validation is about the relationship of the artefact to the artefacts from which it is derived: in the case of goals and requirements, whether the system requirements satisfy the system goals and whether the software specification satisfies the system requirements.

We demonstrate, using a very simple example from our case study, that analysis is possible and useful at an early stage. There is a security requirement that People who are not members of HR Dept are prohibited from displaying personnel information. Analysis (in this case informal) shows that this constraint does not prohibit a member of HR Dept who is currently suspended, possibly because of allegations of dishonesty, from seeing personnel information. This will certainly be regarded by the customer as an example of unauthorised access, so that the security requirement, as stated, is invalid. A more tightly specified security constraint is needed and will be straightforward to generate.

4.3.1 Internal Analysis (Verification)

Security requirements are simply a set of statements, and are therefore subject to the same kind of internal analysis as any other similar set. Taken as a whole are the functional requirements and their associated constraints complete and mutually consistent? For example, security constraints can be mutually inconsistent or conflict with safety constraints. This verification activity is not special to security requirements, and we do not discuss it further.

4.3.2 External Analysis of Security Requirements (Validation)

Even if internal analysis of the requirements has verified that they are consistent, validation is still necessary at two stages.

- Are the system security requirements a valid refinement of the security goals?
- Is the problem frame, including its software security specification and trust assumptions, a valid refinement of the system security requirements?

At each stage there are two kinds of analysis that are appropriate:

- Construction of a correctness argument, that the refinement is correct;
- Vulnerability analysis: the discovery of flaws in the system security requirements so that they do not satisfy the security goals (first stage); or

flaws in the problem frame so that it does not satisfy the system security requirements (second stage).

Correctness Arguments

Correctness arguments are seen by Jackson as an integral part of the problem frames approach and there is no reason why the introduction of security considerations should affect this. Ideally, a proof of correctness would be all that is needed, but a variety of factors, including ambiguity of descriptions, false assumptions, incomplete specification of the context and the complexity of systems, mean that vulnerability analysis is always necessary.

Correctness arguments have two kinds of value:

- They give confidence that the initial design is, at least *prima facie*, correct;
- They are one way of providing structure to the process of vulnerability analysis.

Vulnerability Analysis

Vulnerability analysis finds weaknesses in a design, which show that it does not satisfy its requirements. van Lamsweerde [55] demonstrates vulnerabilities in software by showing that the software is able to satisfy the negation of a security goal. In doing this he demonstrates an important aspect of vulnerability analysis; it validates a lower-level artefact against a defined higher-level one. By contrast, Liu, Yu & Mylopoulos[37] and Lin et al [36] both describe methods of analysing possible "illicit" use of a system, but neither define the requirements or goals that this illicit use violates.

One of the difficulties for vulnerability analysis is to propose a methodical approach to discovering vulnerabilities. In the past this has been done by attack teams who rely upon *ad hoc* methods; it seems likely that most of the flaws described by Landwehr in [32] were discovered in this way. Proposals for tree-structured methods, such as Schneier's Attack Trees [49] suffer from two disadvantages:

- They do not provide any initial structure from which to start the analysis
- There is no way of knowing whether enough branches have been discovered.

The safety critical world has developed a more methodical approach, based on HAZOPs [29], a systematic method of describing what can go wrong in a chemical plant by applying key words such as TOO MUCH, or TOO LATE to each element of the plant. Any deviations that are discovered can be pursued in two directions, by means of cause-consequence analysis:

- Cause analysis follows an attack tree downwards in a search for vulnerabilities
- Consequence analysis searches upwards to discover what harm might result form the deviation.

This has been extended to software and, more recently to requirements by Srivatanakul et al in [53], which applies the HAZOPs technique to UML use cases. This provides some assurance of completeness of coverage, on the assumption that the use case describes all possible happenings. Also, it gives a structure to the start of the vulnerability analysis.

Another possible approach is to generate vulnerabilities by considering each step of the correctness argument in turn, which we have found useful in removing errors from our own case study, but we are not aware of any systematic work on this.

Our own work [17] on trust assumptions provides a focus point for discovering vulnerabilities, by considering the consequences of failure of the assumptions. Our work on threat descriptions [18] helps to locate the points where trust assumptions are necessary by considering where assets are used.

Other published work on vulnerability analysis has come from three areas:

- Goal refinement
- Abuse and misuse cases
- Abuse Frames.

Goal Refinement

The extension of the KAOS goal refinement method to vulnerability analysis [55], and a complementary technique in *i** [37] have been mentioned above.

Abuse and Misuse Cases

Abuse and misuse cases [2, 38, 50] are techniques that have been developed to elicit security requirements by demonstrating vulnerabilities in use cases, though less systematically than the HAZOP-based technique mentioned above. It is not clear whether they are intended for use in eliciting and validating system security requirements or a software security specification. It appears to us that they could be used at either level.

Abuse frames

Abuse frames [36] are a technique, currently under development, that uses problem frames. It identifies possible illicit behaviour as a result of two possibilities:

- Behaviour that is permitted by the combined properties of the domains
- Behaviour that is permitted as a result of perturbation of the problem frame, by changing either the domain properties or the frame topology.

4.4 Covert Channels

Covert channels were originally identified by Lampson [31], and are defined by him as information channels that are not intended for information transfer at all.

As Axel van Lamsweerde has pointed out[7], there is the possibility of creating, and detecting, covert channels in the course of generating requirements.

Using an example based on his suggestion, suppose there is a proposal to create an electronic purse system with a superset of the following goals:

> Enquire about purse limit (functional goal)
>
> Put money into the purse (functional goal)
>
> Only the purse owner should be able to know their purse limit (security goal)
>
> Users should know whether their actions are successful (usability goal).

This can be refined to operationalised requirements and constraints:

> Enquire about purse limit, but only the purse owner is permitted to do this
>
> Put money into the purse, with no constraint on who is permitted to do this.
>
> In order to meet the usability goal, the operations will return a Success or Failure response.

Given these operationalised requirements, it is possible for an attacker to discover a lower bound on the owner's purse limit, by successively putting money into the purse until a Failure response is received. This is a covert channel, which is created, by the conflicting goals of confidentiality of the purse limit and the need to provide responses to invocations of operations[8].

Most covert channels are introduced during software design and programming, and hardware design: they are outside the scope of requirements analysis.

4.5 A Multi-domain Approach

There is ample evidence that security has to be considered in every relevant domain. It is not by chance that Kevin Mitnick, the arch-hacker of recent times, whose exploitation of IP spoofing and other weaknesses in the TCP/IP protocols has given rise to a whole new generation of technical attacks, has written a book on Social Engineering [39]. His attacks illustrate the exploitation of vulnerabilities arising from a combination of the properties of human (procedural), physical and software domains.

Michael Jackson's work on Problem Frames [28] has enabled us to articulate a multi-domain approach. Requirements are about what happens in the world, while

---

[7] Personal communication.

[8] We are not aware of a solution to this particular covert channel, without some element of compromise.

software specifications only deal with interfaces. As we emphasise below, security is about protecting real-world assets, while many security techniques are expressed entirely in terms of the behaviour of software. So problem frames are an essential element in our exposition of security requirements.

In one respect we differ from Jackson, not because we believe that his approach to his chosen problem area is wrong, but because our concerns are different. He explicitly regards the machine as the optative target of specification, and all other domains as indicative. As showed in our discussion of the case study, we do not take this approach. All kinds of security constraint – physical, procedural and software-based – need to be considered, and probably used in combination.

A consequence of this is that we use Jackson's *biddable domains* (usually, people) in their true dictionary meaning: "docile; obedient". We accept that they lack "positive predictable causality … the most that can be done is to issue instructions to be followed", but in the security world this is true of computers as much as it is of people. Both computers and people can be programmed or "trained to follow stipulated procedures and can be expected to do so". Both computers and people may fail to follow the procedures and we must allow for this in our security design. This principle is already well established in system safety engineering, see e.g. Leveson [35] where a combination of physical, procedural and software safety measures is used, taking into account the likelihood of failure of any of them.

Multiple Domains and Security Principles

There are two principles (see, e.g. Zwicky and Chapman [57]) that should be obeyed when designing for secure systems:

- Defence in Depth: it should always be assumed that a constraint is fallible, so if one fails, another should still prevent a successful attack on an asset.
- Diversity of Defence: Defence in Depth is more likely to be successful if the defences that are used are diverse in nature.

It is therefore desirable, whenever possible, to supplement security measures of one kind with those of another; a combination of physical, procedural and software security is likely to be most effective. These principles reinforce the need to take a multi-domain approach.

4.6 Security Functions

A security framework discussion would not be complete without a mention of security functions. Where do functions such as access control, authentication, encryption, etc, fit in? Our answer is that they are *functions* (full stop). We use the same argument as for constraints in section 4.2.4 above. If the designer includes a function in order to satisfy a security requirement (i.e. derived from a security goal), then we could reasonably describe it as a security function, but if that same function is used to satisfy some other kind of goal, that is a different matter.

Pursuing the example of section 4.2.4 above, if we have a Comprehensibility requirement that only members of HR Dept are permitted to read Personnel

Information, and we decide to implement that using authentication and access control functions, then these functions should be described as Comprehensibility functions. On the other hand, if they are used in their more common role of supporting security requirements, then we will call them security functions.

4.7 Security Policies

"Security Policies" is a phrase used extensively in the literature, but it has been used with such a wide variety of meanings that we have avoided its use in this discussion. However, some of those meanings are related to security requirements, and so we provide a brief survey of security policies here.

We have found the following main uses of the phrase:

- An organisation's security policy document
- Individual policies, either part of a policy document or issued individually by an organisation. They are not discussed further here, as it is impossible to generalise about them.
- Access control policy, of which there are several kinds
- Mechanisms for establishing parameters for security functions, such as authentication or cryptography, in secure peer-to-peer communication sessions, as in IPsec [45].

4.7.1 Security Policy Document

An information security policy document is advised by any text on information security management, as exemplified by BS 7799-1 (ISO/IEC 17799) and BS 7799-2 [10, 27]. It sets out the security goals for each area of information technology. Some of these are general goals, some expressed at the requirements level and some are quite technology-specific. A summary of a recommended information security policy can be found in appendix A.3 of BS 7799-2. It, like most policy documents, does not limit itself to the behaviour of software, but also covers physical and procedural policies.

A good security policy document sets out security goals, and more concrete requirements and design attributes where the organisation has made decisions in order to avoid each application needing to repeat the goal refinement process.

4.7.2 Access Control Policy

Access control policies are often referred to as models. They are limited in scope to access control, specifically to the behaviour of the Access Decision Facility of a Reference Monitor [3, 24]. They constrain access attempts to a specific pattern. There are several different kinds of this policy:

- Global access control policies, built into a system
- Mandatory Access Control mechanisms
- Discretionary access control policies.

In the discussion below we attempt to fit them into the following categories:

- Development process requirements
- Requirements
- Mechanisms.

4.7.3 Global access control policies

These are policies that are entirely "hard coded" into a system.

Specific Constraint on Operations

From our point of view the simplest access control policy is one which states a specific global constraint on operations. Brewer & Nash's Chinese Wall Security Policy [9], already mentioned (section 4.2 above) states the following constraint on traces:

> The system shall not provide information about an organisation to any person who has previously accessed information about a competitor organisation.

This kind of policy can be directly stated as a *security requirement*.

Process-defined Constraint on Operations

The Clark-Wilson Integrity Model [11] also specifies a constraint upon operations, but less directly. To apply this model, the developer identifies Constrained Data Items (CDIs) whose integrity requires protecting and then ensures that they are only manipulated by well-formed transactions (Transformation Procedures, or TPs) which can be guaranteed to maintain the integrity predicates required by the system, e.g. no net change in value when performing a double-entry accounting transaction. Integrity Verification Procedures (IVPs) need to be run periodically in order to audit that the data conforms to the integrity predicates.

This is a combination of a *development process requirement* (identifying CDIs and defining their integrity predicates) and design of TPs and IVPs to conform to and verify the integrity predicates (*security requirement*). It appears that the integrity predicate itself is nowhere directly represented in the system (contrast with System Management Policies, below), so a system audit cannot verify, by inspection of the code alone, whether the system conforms to a Clark-Wilson policy.

Clark and Wilson also recommend the use of Separation of Duties for operating upon CDIs. This is another example of specific constraint on operations (discussed above).

4.7.4 Mandatory Access Control Mechanism

The access control literature makes the distinction between Mandatory and Discretionary access control policies:

- Mandatory access control (MAC) policies cannot be changed by users of the system
- Discretionary access control (DAC) policies can be added, removed and altered by users with appropriate authority.

A MAC policy defines *mechanisms* to enforce mandatory constraints on access control. In these policies, there is a built-in mechanism to enforce control, but the actual decisions depend upon labels, attached to subjects and objects, that can be altered by a privileged administrator.

The Bell-Lapadula policy model [7] is a MAC policy. In contrast to those discussed above, does not directly define any constraint on operations. Instead it provides a mechanism for doing so. It requiring that principals and data objects should each be associated with labels, called Clearance and Classification respectively. The labels are partially ordered. The policy enforces the constraint that a subject is only permitted to read a data item if their Clearance is the same as, or dominates, the data's Classification. The Biba Integrity policy [8] provides a similar mechanism intended to ensure data integrity; for further discussion of this, see Sandhu's taxonomy of data integrity [47].

Should requirements engineering be concerned with mandatory access control polices, since they are essentially mechanisms? Possibly not, but there is an aspect which could be relevant: the ability of some of them to withstand Trojan Horses. Downs [14] motivates their use by pointing out that the Bell-Lapadula model enables the prevention of leakage of information by Trojan Horses. Since it is impossible to guarantee the absence of Trojan Horses from those parts of a system outside its Trusted Computer Base [13], they can be rendered incapable of passing information to uncleared subjects if the model is implemented. So a requirement to prevent Trojan Horses leaking information can be stated knowing that there is an implementation mechanism to achieve it.

4.7.5 Discretionary Access Control Policies

Discretionary access control policies (see, e.g. [40]) are used to specify the constraints on operations by means of access rules that can be altered by users with appropriate authority. We include Sandhu et al's Role Based Access Control [48] here. Discretionary access control policies clearly have characteristics in common with *security requirements*, but are not identical to them:

- They express constraints in a way similar to security requirements, but are expressed in terms of the objects known to a machine, e.g. UserIds and files, rather than those that exist in the real world, e.g. people and information.
- They may be at a high level, suitable for the specification of requirements, but are often at a very detailed, implementation-dependent level.
- They can be altered by operations of the system. In this respect they differ from normal requirements, whose change typically requires passage through a change management process and regeneration of the system.

- They are dependent on the Access Decision Facility of a Reference Monitor, and so are not suitable for the definition of security requirements that might be implemented by encryption or some other security mechanism.

The Harrison-Ruzzo-Ullman model [19] specifies constraints within which alterations to access rules can be made in a discretionary model.

Distributed System Management policies

Distributed System Management policies [51] were introduced to enable flexible management of a distributed system. They contain two elements:

- Obligation policies, which trigger defined management actions on the occurrence of pre-defined events;
- Authorisation polices, which are simply discretionary access rules.

These policies are represented as objects which can be manipulated at run-time, in the Ponder language [12], and are therefore more flexible than static policies.

5. Open Issues

5.1 Security Requirements in the Presence of Implementation Flaws

We have been proposing a framework for security requirements with the implicit assumption that designers will then implement a system which satisfies those requirements completely. This assumption is, of course, untrue. The platforms upon which the systems will be implemented will contain a great deal of unwanted functionality, much of which will violate the security requirements, as documented in CERT[9] alerts.

There is therefore a need for investigation of how we can configure problem frames so that, while knowing, that the individual domains are flawed, the risk of violation of the security requirements is minimised. This is a subject for future research, although much work has been done on fault-tolerance, e.g. Lee [33], and it should be possible to make progress on the requirements of security fault-tolerant systems.

5.2 The Need for a Taxonomy of Constraints

We have given some examples of constraints as security requirements, but have not attempted a full taxonomy of relevant constraints, although there is a need for this. There appear to us to be several issues requiring further work:

[9] http://www.cert.org/

Constraint Expressions

Constraints that express security requirements will always be constraints on operations, but what are the contents of the constraint expression? It appears to us that it will include at least some of the following elements, and possible more: predicates on the parameters of the operation, its originator and source; temporal constraints; and constraints on traces of the behaviour of a system.

Grey Security Requirements

Security requirements tend to be expressed in black and white terms, but actually total security is unobtainable, and so security requirements need to be toned down accordingly. Henning [21] examines whether security related service level agreements, analogous to other service level agreements, might be possible. Irvine & Levin [23] discuss the concept of quality of security service. It is clear that the measurement of levels may have to very approximate in many cases: more a rough level of uncertainty than a precise figure. Different units may need to be used, reflecting the different approaches to assurance that are to found in the Common Criteria for security evaluation [26]; the work factor needed to break the requirement by known means, an estimate of its vulnerability to as yet unknown means of attack and the degree of assurance that the design and implementation is free of flaws, may all be factors in the measurement.

Availability Requirements

On the face of it, availability requirements can be regarded as temporal constraints on the response time of operations. System availability requirements can be regarded as universally quantified constraints on the response time of operations. Response time constraints are quite different from the constraints mentioned above, but are very similar, if not identical, to the constraints that are needed to specify performance requirements. Work is needed to see if this approach accurately reflects the requirements of real users.

5.3 Data-driven Security Requirements

All of our discussion so far has been function-driven; we have been advocating an approach in which an application is examined for functions which, if misused, could cause harm to assets. However, a high proportion of an organisation's computer-related assets are represented directly by corporate data that is held in databases. The harm that could be caused to an organisation depends upon the data, independently of the application that manipulates it, and therefore the security requirements can be attached to the data itself.

The Clark-Wilson Integrity Model [11], discussed in section 4.7.3 above, does not include a concept of a direct representation of integrity predicates. However, database integrity predicates, familiar from database textbooks such as Garcia-Molina [16], *do* directly represent and store integrity predicates for data items, and maintain integrity by refusing to commit any transaction that would violate the predicate, or taking corrective action.

Although it is not clear that the concept of integrity in the database world is identical to that in the security world, if it were possible to identify the security requirements for individual data items this would ease the task of security requirements engineering for individual applications. This is in the same spirit as a control principle that mandates separation of duties for critical data items, but more systematic.

5.4 Specification Notation

This paper presents a framework, intended to encompass diverse means of expression, and we do not intend to mandate any particular specification notation, although we have ourselves found the problem frames approach useful. If a formal specification notation is used for functional requirements, then security requirements could be stated by formal predicates. On the other hand, if the functions are defined less formally, then so will be the security requirements. However, there are at least two general issues of specification notation, which need further research:

- Structuring security requirements
- How to state security requirements without using too many negatives.

- ***Structuring Security Requirements***

Although the problem frames approach has been very useful in presenting the relationship between system and software security requirements, our presentation has gone beyond the bounds of the standard approach. Further work is needed on problem frames in this area, and also investigation of whether there are any other structuring approaches, e.g. in the area of safety critical systems engineering, which could be valuable.

- ***Stating Security Requirements***

The reader will have noticed some awkwardness in the language used in the case study. Phrases such as "shall not display … except to …" do not trip lightly from the tongue. It would help if the number of negatives that are used could be reduced.

The default assumptions of Access Control Systems (see any computer security textbook) would adapt very conveniently to security requirements, and their possible use should be studied. In particular, the three levels of priority of access control statements, using the Closed World assumption, might transplant conveniently to security constraints:

- The Closed World assumption – if no positive permission is stated, the default is prohibition
- Positive permissions override the default
- Explicit prohibitions override positive permissions

We note that there is a risk, to be avoided, that the language is so closely modelled on existing access control definition languages that it biases the

designer towards using access control as a solution, without considering other solutions such as encryption.

5.5 Risk analysis

Risk analysis, as represented by a method such as Octave approach [1] comes into Baskerville's [6] category of mechanistic engineering methods. The advent of security requirements engineering brings us one stage closer to his aim of integrated design. However, there is no general agreement about how to integrate conventional risk analysis into the process, and this requires further research.

Scalability

It is remarkable how much space we have taken up in the exposition of the simplest possible case study. Do our proposals fail because they produce an unworkable volume of material?

The size of security risk analysis documents is already notoriously huge, and we do not believe that we are proposing a large, if any, increase. What we *are* doing is making the problem more manageable by adding structure to it; it is clear whether, at any moment, one is doing security goal refinement, or realising the security requirements in a problem frame, or going beyond the scope of this paper into security design.

We have not disposed of the scalability problem, but have made some contribution to breaking it into more manageable portions.

6. Conclusions

We announced three principles at the start of this paper:

- The "what" of security requirements must be understood before the "how" is described. We have achieved this by defining and describing security goals and requirements, and showing their relationship to a software specification within the problem frames architecture.
- Security cannot be considered as a feature of software alone; it is concerned with the prevention of harm in the real world. This has been achieved by making a clear distinction between system and software security requirements.
- Security requirements can most usefully be defined by considering them at the same level as functional requirements. We have proposed, and worked through the implications of, the definition of security requirements as constraints on system functions.

Advantages of our approach are:

- Security requirements are naturally integrated with the system's functional requirements and constraints derived from other sources. An integrated development is possible.

- This has the consequence that interactions and trade-offs between security and other quality requirements can be analysed. For example, interactions and trade-offs between them can be considered in terms of the different required constraints on the same functional requirements.

We claim that this framework will help requirements and security engineers to understand the place of the various synthetic and analytical activities that have previously been carried out in isolation. The framework has raised a number of issues, mentioned in the discussion, but we believe that it provides a way forward to effective co-operation between the two disciplines of requirements and security.

We have no illusions that this paper does more than open up the subject of security requirements, so as to make further progress. However, by concentrating on defining the core security requirements artefacts, we have avoided introducing unnecessary alligators into our swamp, and can point to a pathway out of it.

Acknowledgements

We are grateful for the support of The Leverhulme Trust and the useful comments and feedback from Annie Anton, members of the Security Requirements Group at the Open University and the Secure Network Group at the University of York.

References

1. Alberts, C. and A. Dorofee, *Managing Information Security Risks: The OCTAVE (SM) Approach*. 2002: Addison Wesley.
2. Alexander, I., *Misuse Cases in Systems Engineering.* Computing and Control Engineering Journal, 2002. **13**(6): p. 289-297.
3. Anderson, J.P., *Computer Security Technology Planning Study*. 1972, ESD/AFSC Hanscom, AFB Bedford, Mass.
4. Anderson, R. *Security in Clinical Information Systems.* in *IEEE Symposium on Security and Privacy*. 1996. Oakland, CA.
5. Antón, A.I. and J.B. Earp, *Strategies for Developing Policies and Requirements for Secure E-Commerce Systems*, in *Recent Advances in E-Commerce Security and Privacy*, A.K. Ghosh, Editor. 2001, Kluwer Academic Publishers. p. 29-46.
6. Baskerville, R., *Information Systems Security Design Methods: Implications for Information Systems Development.* ACM Computing Surveys, 1993. **25**(4): p. 375-414.
7. Bell, D.E. and L.J. LaPadula, *Secure computer system: Unified exposition and Multics interpretation*. 1976, MITRE Corporation.
8. Biba, J.K., *Integrity Considerations for Secure Computer Systems.* 1977, Mitre Corporation: Bedford, Mass.
9. Brewer, D.F.C. and M.J. Nash. *The Chinese Wall Security Policy*. in *IEEE Symposium on Security and Privacy*. 1989. Oakland, CA: IEEE Computer Society Press.
10. BS 7799-2, *Information security management - Part 2: Specification for information security management systems*, 1999.

11. Clark, D.C. and D.R. Wilson. *A Comparison of Commercial and Military Computer Security Policies*. in *IEEE Symposium on Security and Privacy*. 1987. Oakland, CA: IEEE Computer Society Press.
12. Damianou, N., et al., *The Ponder Policy Specification Language*, in *Policies for Distributed Systems and Networks*, M.S. Sloman, J. Lobo, and E.C. Lupu, Editors. 2001, Springer. p. 18-38.
13. Department of Defense (USA), *Department of Defense Trusted Computer System Evaluation Criteria*. 1985.
14. Downs, P.D., et al. *Issues in Discretionary Access Control*. in *1985 Symposium on Security and Privacy*. 1985: IEEE Computer Society.
15. Firesmith, D.G., *Common Concepts Underlying Safety, Security, and Survivability Engineering*. 2003, Carnegie Mellon Software Engineering Institute.
16. Garcia-Molina, H., J.D. Ullman, and J. Widom, *Database Systems:The Complete Book*. 2002: Prentice Hall.
17. Haley, C.B., et al. *The Effect of Trust Assumptions on the Elaboration of Security Requirements*. in *12th IEEE Int Requirements Engineering Conference (RE04)*. 2004. Kyoto, Japan.
18. Haley, C.B., R.C. Laney, and B. Nuseibeh. *Deriving Security Requirements from Crosscutting Threat Descriptions*. in *3rd Int Conf on Aspect-Oriented Software Development (AOSD'04)*. 2004. Lancaster, UK: ACM Press.
19. Harrison, M.A., W.L. Ruzzo, and J.D. Ullman, *Protection in operating systems*. Communications of the ACM, 1976. **19**(8): p. 461-471.
20. Heitmeyer, C. *Applying `Practical' Formal Methods to the Specification and Analysis of Security Properties*. in *Information Assurance in Computer Networks (MMM-ACNS 2001)*. 2001. St. Petersburg, Russia: Springer-Verlag.
21. Henning, R.R. *Security service level agreements: quantifiable security for the enterprise?* in *New Security Paradigms Workshop*. 1999. Ontario, Canada: ACM Press.
22. IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, 1998.
23. Irvine, C. and T. Levin. *Quality of security service*. in *New Security Paradigms*. 2000. County Cork, Ireland.
24. ISO 10181-3, *Open Systems Interconnection - Security Frameworks - Part 3: Access Control*, 1991.
25. ISO/IEC 15408-1, *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 1: Introduction and general model*, 1999.
26. ISO/IEC 15408-3, *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 3: Security assurance requirements*, 1999.
27. ISO/IEC 17799, *Information technology -- Code of practice for information security management*, 2000.
28. Jackson, M., *Problem Frames: Analysing and Structuring Software Development Problems*. 2000: Addison Wesley.
29. Kletz, T.A., *HAZOP & HAZAN: Notes on the Identification and Assessment of Hazards*. 1983, Rugby: Institution of Chemical Engineers.

30. Kotonya, G. and I. Sommerville, *Requirements Engineering - Processes and Techniques*. 1998: John Wiley.
31. Lampson, B.W., *A Note on the Confinement Problem*. Communications of the ACM, 1973. **16**(10).
32. Landwehr, C.E., et al., *A Taxonomy of Computer Program Security flaws*. ACM Computing Surveys, 1994. **26**(3): p. 211-254.
33. Lee, P.A. and T. Anderson, *Fault Tolerance: Principles and Practice*. 2nd ed. 1990: Springer-Verlag.
34. Lee, Y., J. Lee, and Z. Lee, *Integrating Software Lifecycle Process Standards with Security Engineering*. Computers & Security, 2002. **21**(4): p. 345-355.
35. Leveson, N.G., *Safeware: System Safety and Computers*. 1995: Addison Wesley.
36. Lin, L., et al. *Introducing Abuse Frames for Analysing Security Requirements (poster presentation)*. in *RE'03: 11th IEEE International Requirements Engineering Conference*. 2003. Monterey Bay, CA, USA.
37. Liu, L., E. Yu, and J. Mylopoulos. *Security and Privacy Requirements Analysis within a Social Setting*. in *RE'03 - 11th IEEE International Requirements Engineering Conference*. 2003. Monterey Bay, CA, USA.
38. McDermott, J. and C. Fox. *Using Abuse Case Models for Security Requirements Analysis*. in *Annual Computer Security Applications Conference*. 1999. Phoenix, Arizona.
39. Mitnick, K., *The Art of Deception: Controlling the Human Element of Security*. 2002: John Wiley & Sons Inc.
40. Moffett, J.D., *Specification of Management Policies and Discretionary Access Control*, in *Network and Distributed Systems Management*, M.S. Sloman, Editor. 1994, Addison Wesley. p. 455-479, Chapter 17.
41. Moffett, J.D., et al., *A Model for a Causal Logic for Requirements Engineering*. Journal of Requirements Engineering, 1996. **1**(1): p. 27-46.
42. Moffett, J.D. and B.A. Nuseibeh, *A Framework for Security Requirements Engineering*. 2003, Department of Computer Science, University of York.
43. Mouratidis, H., P. Giorgini, and G. Manson. *Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems*. in *15th Conference on Advanced Information Systems Engineering (CAiSE'03)*. 2003. Klagenfurt/Velden, Austria: Springer-Verlag.
44. Nuseibeh, B.A., *Weaving Together Requirements and Architectures*. IEEE Computer, 2001. **34**(3): p. 115-117.
45. RFC 3586, *IP Security Policy (IPSP) Requirements*, IETF Network Working Group, August 2003.
46. Rushby, J. *Security Requirements Specifications: How and What?* in *Symposium on Requirements Engineering for Information Security (SREIS)*. 2001. Indianapolis.
47. Sandhu, R.S., *On Five Definitions of Data Integrity*, in *Database Security VII: Status and Prospects*. 1994, North-Holland.
48. Sandhu, R.S., et al., *Role-Based Access Control Models*. IEEE Computer, 1996. **29**(2): p. 38-48.
49. Schneier, B., *Secrets and Lies*. 2000: John Wiley and Sons.
50. Sindre, G. and A.L. Opdahl. *Eliciting Security Requirements by Misuse Cases*. in *37th International Conference on Technology of Object-*

Oriented Languages and Systems (TOOLS-PACIFIC 2000). 2000: IEEE Computer Society Press.
51. Sloman, M.S., *Policy Driven Management for Distributed Systems.* Journal of Network and Systems Management, 1994. **2**(4).
52. Spafford, E.H., *The Internet Worm Program: An Analysis*. 1988, Dept of Computer Sciences, Purdue University, West Lafayette, IND 47907-2004.
53. Srivatanakul, T., J.A. Clark, and F. Polack, *Writing Effective Security Abuse Cases*. 2004, Dept of Computer Science, University of York.
54. Tettero, O., et al., *Information Security Embedded in the Design of Telematics Systems.* Computers & Security, 1997. **16**(2): p. 145-164.
55. van Lamsweerde, A. *Elaborating Security Requirements by Construction of Intentional Anti-Models*. in *ICSE04*. 2004.
56. Vickers, A. and J. Smith, *Issues for Industrial Strength Requirements Management*, in *Requirements Engineering at the University of York*, A.J. Vickers and L.S. Brooks, Editors. 1998, Dept of Computer Science, University of York. p. 48-55.
57. Zwicky, E.D., S. Cooper, and D.B. Chapman, *Building Internet Firewalls*. 2000: O'Reilly UK.