Department of Computer Science, University of York
Report YCS 368

A Framework for Security Requirements Engineering

Jonathan D. Moffett
Department of Computer Science[1]
University of York, UK

jdm@cs.york.ac.uk

Bashar A. Nuseibeh
Security Requirements Group
Department of Computing
The Open University, UK

B.A.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 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.

1. Introduction

1.1 Motivation

Although security requirements engineering has recently attracted increasing attention, it has lacked a context in which to operate. This lack was pointed out in (Baskerville 1993), where he presents three generations of security design methods: checklists; mechanistic engineering methods[2]; and integrated design. 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

[1] This work was carried out in the role of Visiting Research Fellow at the Security Requirements Group in the Department of Computing at The Open University.

[2] Our exposition of a security risk analysis process in section 2.2 falls into this category.

order to show the truth of this statement, and then motivate the remainder of the paper by showing why it matters.

Previous Work on Security Requirements

Extensive work has been carried out on security requirement during the last few years. (Lee, Lee et al. 2002) point out the importance of considering security requirements in the development life cycle, but does not show how to integrate them with other requirements. (Heitmeyer 2001) 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 and Fox 1999), followed independently by (Sindre and Opdahl 2000) and elaborated by (Alexander 2002), describe abuse and misuse cases, extending the use case paradigm to undesired behaviour. (Liu, Yu et al. 2003) 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 Letier 2000) use the concept of security goals, and describe obstacles in the KAOS method, which prevent security goals from being met, but they do not then take the further step of defining security requirements at the same level as operationalised KAOS functions. (Antón and Earp 2001) use the GBRAM method to operationalise security goals for the generation of security policies and requirements, but also do not define security requirements.

None of the above define what security requirements *are*. On the other hand, when discussing non-functional requirements (NFRs), (Kotonya and Sommerville 1998) defines NFRs as "restrictions or constraints" [on system services] and similar definitions can be found in other text books. Security requirements are an instance of NFR, and our view is identical to that of Kotonya; they are requirements for constraints on system functions. (Rushby 2001) appears to take a similar view, stating "security requirements mostly concern what must *not* happen".

The Importance of Security Requirements

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 (Anderson 1996). 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 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.

1.2 Outline of Approach

We propose a framework 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 assets from those threats, and are operationalised into security requirements, which take the form of (a subset of) the constraints on the functional requirements.

1.3 Outline of Paper

The remainder of this paper is organised as follows. Section 2 introduces our proposed framework; sections 2.1 – 2.4 expound its concepts; sections 2.5 – 2.7 explore some of its consequences; and section 2.8 examines the relationship between system security requirements and software requirements. Section 3 is a case study, whose functional requirements are set out in section 3.1; it is in two parts, covering system security requirements in section 3.2 and their realisation in software requirements in section 3.3. Section 4 considers some of the many issues that are raised by our proposal, and section 5 concludes the paper.

2. The Framework

We illustrate our framework in figure 1, which is a UML class diagram showing its structure. The framework is a meta-model, in the spirit of the KAOS conceptual model (Dardenne, van Lamsweerde et al. 1993).

2.1 Requirements Engineering Concepts

On the left-hand side of figure 1 we represent, in simplified terms, some generally accepted concepts of requirements engineering. A business has goals, some of which can naturally be described as *functional goals*, i.e. to carry out some business task. There are also *non-functional goals* (NFGs) as described in, e.g. (Kotonya and Sommerville 1998). Examples are reliability, usability, safety and, of course, security. We mention non-security NFGs for completeness, but do not attempt to treat them in this paper.

The right-hand side of the diagram is best described in terms of a security risk analysis and management process, which we do in section 2.2 below. See (Peltier 2001) for more detail on security risk analysis and management processes.

In this section we outline existing requirements engineering concepts, which we find useful in our discussion. We have felt free to approximate where precision is not critical for our argument; for example, we use the terms function, operation and system service interchangeably.

2.1.1 Goals and Requirements

Any organisation has a number of goals, which drive and control the business, and are stated at a high level. Some of these, e.g. the articles of association of a limited company, define the general activities of the organisation, and there are usually more detailed goals stated in policy documents, which describe these activities more precisely. We call these *functional goals*.

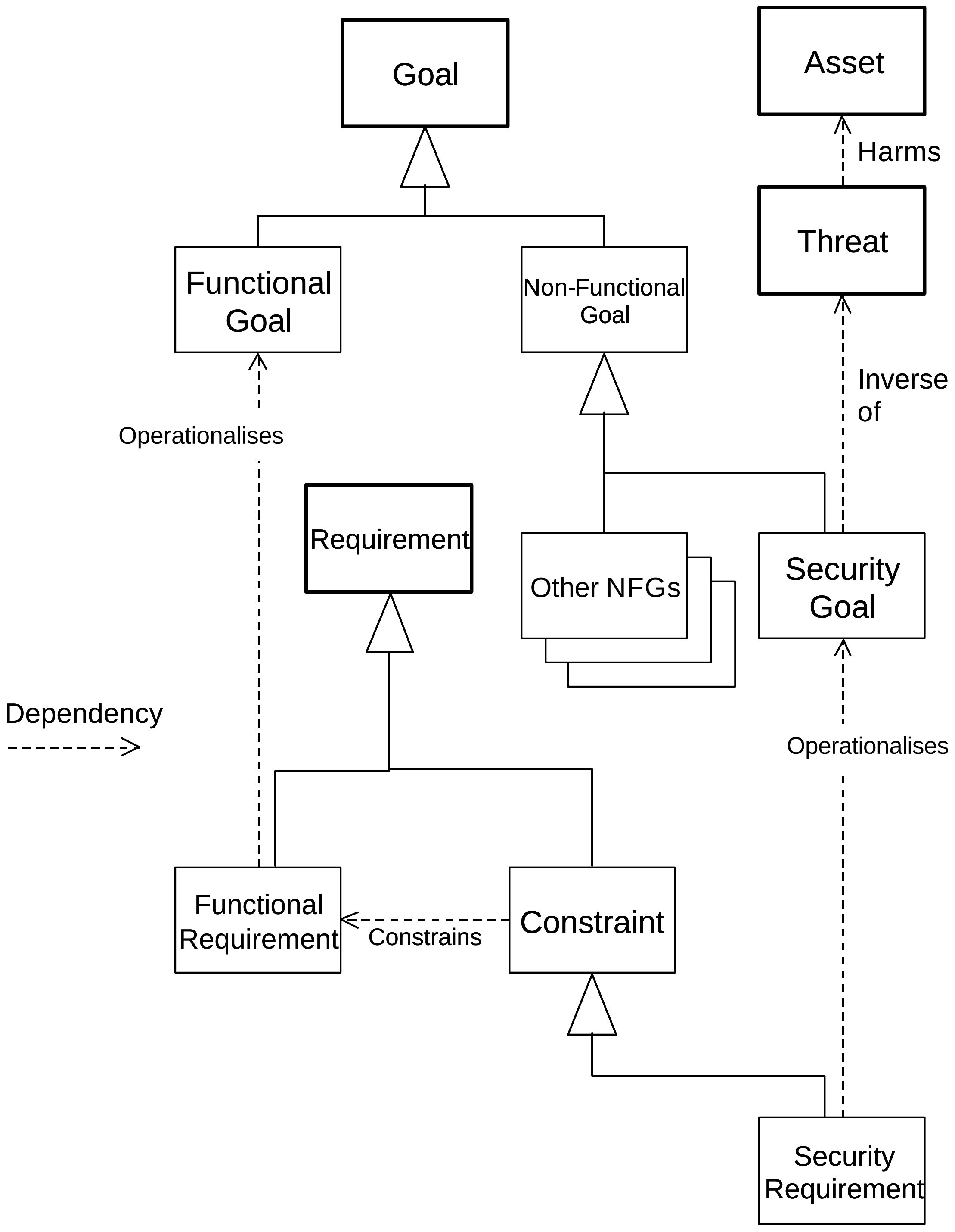


Figure 1: The Security Requirements Framework

Functional goals are refined, using an approach such as KAOS, until they are made operational, as functions to be performed by agents. We call these *functional requirements.*

These functional requirements may have *constraints* on them, where a constraint is a limitation of the freedom of performance of a function. In addition to the examples of constraints that support security, which we will give below, other examples of constraint on a function are: a requirement to complete a function within a specified time, in support of a performance requirement; and a requirement to present a function through a GUI, in support of a usability requirement.

2.1.2 Twin Peaks

The concept of "twin peaks", described in (Nuseibeh 2001), emphasises that requirements and design cannot be separated. Analysis of a requirements specification will lead to a design proposal, and analysis of the design will show the need for further requirements. Thus, as the development process progresses, what started as a small amount of detail within the requirements and design specifications, will broaden out as detail is added.

2.1.3 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 (Mitnick 2002). 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 (Jackson 2000) has enabled us to articulate a multi-domain approach. Requirements are about what happens in the world, while 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*[3] target of specification, and all other domains as *indicative*. As we shall show in our discussion of our simple example, 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 Jackon'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

[3] *Optative* properties are required to be implemented
Indicative properties are assumed to exist already.

engineering, see e.g. (Leveson 1995) 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, Cooper et al. 2000)) 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.

2.2 Security Risk Analysis and Management

Security risk analysis and management are well-established techniques, which were first developed in the 1960s, and have since been refined and developed. Security risk analysis is concerned with identifying and evaluating the risks to a system, and security risk management then makes decisions on appropriate security measures. It is illustrated in Figure 2.

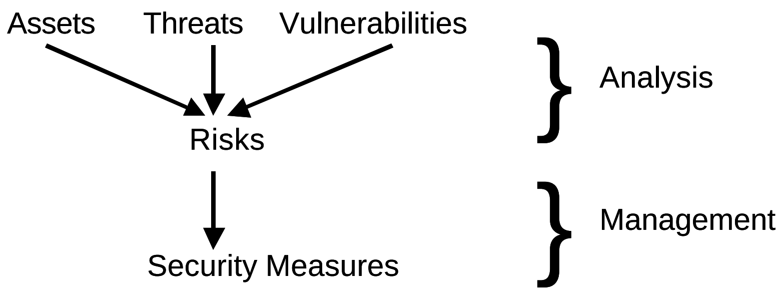


Figure 2: Security Risk Analysis & Management Framework

The meanings of terms in this area are not universally agreed. We will use the following

- *Threat*: Harm that can happen to an asset
- *Impact*: A measure of the seriousness of a threat
- *Attack*: A threatening event
- *Attacker*: The agent causing an attack (not necessarily human)
- *Vulnerability*: a weakness in the system that makes an attack more likely to succeed

2.2.1 Asset Identification

Security is about protecting assets from threats. This is an important point, which has been taken on board more in the area of safety than security. It has been observed that, in safety-critical systems, the computer never harms anyone; it is the system as a whole, including an embedded computer, whose malfunction causes harm. It is significant that Nancy Leveson called her book (Leveson 1995) "Safeware", not "Safe Software", even though a large part of it is concerned with how computers can contribute to, or prevent, accidents.

Similarly, in security, the computer in the abstract can never cause any harm. The assets that are affected by a computer only have value in a real world context; a bit stream that is leaked from a computer only does harm when it is information in the hands of a human competitor or enemy, or if it causes an ATM to put my currency notes into the hand of a thief. So, in the same sense that there is no unsafe software, there is no insecure software; there is only software which, together with the environment in which it is embedded, protects assets or exposes them to attacks.

The first step of a security risk analysis process, after deciding on its scope, is the identification of all relevant assets, with the aim of evaluating them. There are many types of asset controlled by a system, including:

- Information
- Money
- Intangibles, such as an organisation's confidence and public reputation.

The intangibles may be at least as important as the direct assets; organisations have been destroyed by the loss of public confidence that they have suffered from security incidents.

After performing asset identification, we have the Assets that are at the top of the right-hand column of Figure 1. Our aim in this process is to value, as well as to identify, assets, but we cannot perform the valuation until we have identified the relevant threats.

2.2.2 Threat Identification & Asset Valuation

In security risk analysis, asset valuation is not a simple matter of determining its market value. It is valued in negative terms, by the amount of harm that would be caused if a particular threat comes to pass. For each asset type, it is necessary to identify the threats that apply. Thus for stored information there are the following possibilities:

- Unauthorised exposure
- Unauthorised alteration
- Loss of availability.

How much impact (harm) will the business suffer if each of these threats come to pass? The size of the impact will depend upon the nature of the business and the particular item of information; the balance between exposure of information and loss of availability of information will differ widely between a commercial business system and a research organisation, for example.

For other asset types the only threat is loss, e.g. money or reputation.

It is clear that for many asset types, threat valuation is a very approximate exercise and, although ideally we wish to value in precise money terms, in practice it is necessary to categorise, e.g. does this threat have a Low, Medium or High impact on the organisation?

After performing threat identification and asset valuation, we have the Threats, and the quantified Harm that they can cause to Assets, as shown in Figure 1.

Priorities vs asset values

There is a need to reconcile the risk analysis approach of asset valuation with requirements priorities. The requirements will usually be classified by Priority, into classes such as Essential, Useful and Nice-to-Have. By contrast, asset values (with respect to security threats) have a different scale. The two scales may need to be reconciled, but this creates no difficulty in principle.

2.2.3 Vulnerability Analysis, Risk Assessment and Security Measures

The remaining steps of the risk analysis and management process are mainly, but not entirely, concerned with the design and implementation stages of the life cycle. We summarise them here, as we shall have some observations in section 4 below on how requirements engineering can make a contribution to them

Vulnerability Analysis

In this step we analyse a baseline system in order to identify its vulnerabilities. For an existing system the baseline will be that system, with its known security measures and weaknesses. For development of a new system the baseline will take into account the security facilities of the envisaged infrastructure, and standard good practice.

For each threat, the baseline is analysed in order to identify the vulnerabilities, i.e. the means of exploiting a threat successfully. We assess levels of likelihood of attempting, and of succeeding in an attack, and combine them into an assessment of the importance of the vulnerability.

Risk Assessment

Risk assessment combines the results of vulnerability analysis with the impact valuation of threats to assets, and reaches an overall conclusion about the level of risk to an asset.

Security Measures

There are several possible responses to risk

- Avoid it completely by withdrawing from an activity
- Accept it and do nothing, if the risk level is trivial
- Reduce it with security measures.

There are a number of possible security measures that reduce vulnerability

- Reduce likelihood of attempt, e.g. publicise security measures in order to deter attackers
- Reduce likelihood of success by preventive measures, e.g. access control, encryption, firewalls
- Reduce impact, e.g. use fire extinguisher / firewall

- Recovery measures, e.g. restoration from backup

Risk Management identifies the possible security measures, and decides which to choose, based on two main principles:

- Ensure complete coverage
- The expenditure on security measures, and their benefits, should be commensurate with the risks; low risks do not justify high expenditure.

2.2.4 Comment on the Security Risk Analysis and Management Process

The process that is outlined above has served several generations of security professionals well, but has been overtaken by advances in software engineering. In particular, the stages of Vulnerability Analysis, Risk Assessment and Security Measures can be distributed across several stages of the development life cycle, as discussed in section 0.0 below. However, the Asset and Threat stages remain valid and useful.

2.3 Security Goals

The traditional security risk analysis and management process took us satisfactorily as far as the identification of assets and their valuation against threats but, surprisingly, it does not include the concept of security goals. The next step in our framework is the identification of security goals, in order to define, at a high level, what we are aiming at to achieve security. This is a simple step; each threat needs to be inverted to become a goal, by inserting "protection from":

- The threat of unauthorised exposure is converted to the goal of protection from unauthorised exposure, commonly known as Confidentiality
- The threat of unauthorised alteration is converted to the goal of protection from unauthorised alteration, commonly known as Integrity
- The threat of loss of availability is converted to the goal of Availability.

This is a simple step, but it is important because it converts our security concerns into a form that is compatible with the other requirements of an organisation, which is an essential step if we are to integrate security requirements into the mainstream process.

We now have the third box in the right hand column of Figure 1.

Other Security Goals

There are of course other security goals, such as aspects of authentication, which derive directly from other threat types, but are not discussed here because they are not needed for our main argument.

More important, in our meta-model we are representing security goals as the inverse of threats. We are leaving it as an open question at present whether there are other security goals, which are not the inverse of threats. For example, is Anonymity a separate goal, or is it better regarded as a means of achieving a goal such as Confidentiality? Work on understanding the goals of Anonymity will be required to answer that question.

2.3.2 The Source of Security Goals

Where do security goals come from? The obvious source is as described above, from threats 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, which depends upon the bank's Reputation goal, of protecting customers' money. 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.

2.3.3 Characteristics of Security Goals

An organisation's security goals have some characteristics which make them harder to manage than functional goals:

- They cannot be immediately 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 it must be evaluated against the security goals and appropriate security constraints introduced.

For example, although some constraint may be necessary to achieve the Confidentiality goal, it is not by itself sufficient for the purpose. 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 be introduced at a later iteration of our requirement. This implies in practice that a copy of the information exists that can be read using a function that has not been defined in our requirements. Unconstrained use of this function can violate the Confidentiality goal, and therefore there will need to be a security requirement that constrains it. At still later a level, 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, as additional functions are introduced, additional security requirements to constrain the use of those functions are 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. 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.

2.4 Security Requirements

We define *security requirements* to be the constraints, on functional requirements, that are derived from security goals. A simple example is:

> The system shall not display salary information except to members of Human Resources Dept.

There may also be temporal constraints:

> The system shall not display salary information outside normal office hours;

and complex constraints on traces:

> The system shall not display information about an organisation to any person who has previously accessed information about a competitor organisation (the Chinese Wall Security Policy, (Brewer and Nash 1989)).

Availability goals will need constraints on response time:

> The system shall display salary information within 5 seconds for 99% of requests.

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 4.5 and 4.6 below.

Why Define Security Requirements as Constraints?

Note that 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 are:

- *Security goals*. Security goals are necessary as a starting point, but they are more abstract than functional requirements and it would be necessary for the designer to carry out further work, possibly dependent upon requirements domain knowledge that he does not possess, in order to decide how the security goals should constrain the functions.
- *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 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.

2.4.2 The Scope of Security Requirements

The scope of security constraints on functional requirements must be global. They are not to be interpreted as constraining any one statement of functional requirement, but of all instances of that function.

To illustrate this using a simple example from the case study, assume that there are two functional requirements, derived from two separate business goals: for members of HR Dept to Display salary information; and for Auditors to Display salary information. Since they are derived from separate goals, the two functional requirements must be kept separate, to enable traceability. However, an additional prohibition, that staff who are suspended are prohibited from Displaying salary information, must clearly apply to both requirements, and to any other requirement for this function.

2.4.3 How Are Security Requirements Elicited?

In order to derive security requirements, each relevant security goal needs to be examined for possible relevance, and then operationalised constraints must be derived from it. We could do this informally, but goal refinement methods such as KAOS (Dardenne, van Lamsweerde et al. 1993) or i* (Liu, Yu et al. 2003) provide a more methodical approach.

2.4.4 Are there any other 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 the constraints on system functions, 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 expressing a general goal, rather than providing a specification, and there should be no statement to this effect as part of the specification. 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 achievement of the security goals.

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 system is properly and completely expressed as appropriate constraints on the functions.

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.

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.

2.5 Analysing Security Requirements

2.5.1 Internal Analysis of Requirements (Verification)

Security requirements are simply one kind of constraint, and are therefore subject to the same kind of internal analysis as any other kind of constraint. Taken as a whole are the functional requirements and their associated constraints complete and mutually consistent? For example, security constraints can conflict with safety constraints.

This verification activity is not special to security requirements, and we do not discuss it further.

2.5.2 External Analysis of Security Requirements (Validation)

Even if internal analysis of the requirements has verified that they are consistent, it is still necessary to validate them against the organisation's security goals. In particular, will the security constraints actually achieve the security goals?

We give a very simple example of the kind of analysis that could be done at this stage, using our case study. There is a security requirement that People who are not members of HR Dept are prohibited from Displaying salary 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 Displaying salary information. One could argue that this contradicts the security goal of Confidentiality. An additional security constraint, that People who are suspended are prohibited from Displaying salary information, could therefore be added.

The place of Analysis in the framework is discussed in section 4.1 below

2.6 Security Requirements and Security Properties

Security "properties" are often referred to, especially in formal specifications, and 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 2.4.4 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.

2.7 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 NFGs 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 Salary 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 Payroll 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 Payroll 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.

2.8 Software Security Specifications

It will be apparent from our argument so far that we do not believe that there can be a software (or machine) security specification independent of the software specification as a whole. We take the view that security requirements are simply one of many constraints on the functions of a system. The functional aspects of a system requirements specification consist of definitions of required behaviour and constraints on that behaviour (plus traceability information).

It is the job of a designer to determine the optative properties of a software machine and possibly of other domains, given his assumptions about the indicative properties of relevant domains, in order to satisfy all the requirements. Some of the properties of the software will be specified to meet security requirements, but that will not necessarily be apparent from the specification itself. However, we can discover that a particular property derives from a security requirement, by using traceability information which provides the rationale for the design.

2.8.1 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 2.7 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 2.7 above, if we have a Comprehensibility requirement that only members of HR Dept are permitted to read Payroll 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.

3. Case Study

This case study, of a Payroll Information Display system, is used to illustrate the arguments that we have been making above. It is unrealistically simple, to enable points to be illustrated easily.

3.1 System Functional Requirements

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

> Req1: On request from a Person (member of People), the system shall display salary information for a specified payroll number to that Person.

This is realised by the system shown in the problem frame diagram, Figure 3. It shows a Person interacting with the Machine, as a result of which specified payroll information is displayed. Person is a biddable domain.

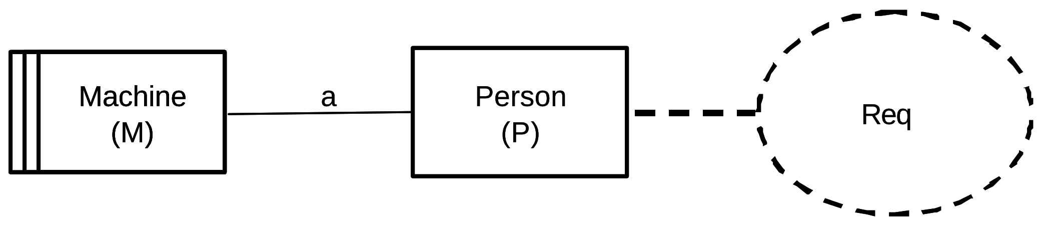
Extending the Problem Frames notation with a causal notation derived from (Moffett, Hall et al. 1996) the Machine specification (Spec) is very simple[4]:

> Spec1: a.P!{PayrollNumber} **shall cause** a.M!{SalaryInformation(PayrollNumber)}

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.

[4] "a" is the label of the interface and P! and M! indicate that Person P and Machine M respectively initiate the PayrollNumber and SalaryInformation interactions. "shall cause" prescribes that the first interaction shall always result in the second one.



a: P! {PayrollNumber}

M! {SalaryInformation}

Figure 3: Payroll Information Display System Problem Frame Diagram

3.2.1 System Security Risk Analysis (Assets & Threats)

Asset Identification

Organisations have a number of types of asset. We could perform the following analysis on any of these types, but examination of the functional requirements reveals only one relevant asset type:

- Information

Other assets would need to be considered in a fuller example, including: tangible assets such as money or products and intangibles such as Reputation.

Threat Identification & Valuation

There are several possible types of threat to Information, including:

T1: Unauthorised disclosure.

T2: Unauthorised alteration.

T3: Unauthorised unavailability.

This is not a complete catalogue, but will suffice for this discussion.

3.2.2 Security Goals

Having identified a threat, we need to take the trivial step of stating a security goal, the prevention of that threat:

SG1: Prevent unauthorised disclosure of information. This is usually described as the goal of Confidentiality.

SG2: Prevent unauthorised alteration of information. This is usually described as the goal of Integrity.

SG3: Ensure availability of information.

Asset values

The threat types, and therefore the security goals, can be defined for a general class of asset, such as information. The impact of the threat is a measure of the cost that will be incurred, and is the asset value with respect to that threat. This value is not uniform across an asset type; loss of Confidentiality has a valuation of nil for publicly available information but the disclosure of some other kind of information may carry a substantial cost to the organisation. In our example we will assume that the organisation sets a High value on maintaining the confidentiality of personnel information.

We can therefore state a specific security goal for confidentiality of personnel information, and similar ones for integrity and availability[5]:

SG4: Confidentiality of personnel information. Valuation - High.

SG5: Integrity of personnel information. Valuation - High.

SG6: Availability of personnel information. Valuation - Medium.

3.2.3 From Security Goals to Security Requirements

We have now derived and valued the organisation's security goals for a particular kind of asset. How are these relevant to the Payroll Information Display System application?

First, we can note that the way in which a goal such as SG4 is expressed does not map naturally onto our functional requirement Req1.

Req1 is a functional requirement, and functional requirements are the basis for a structural and behavioural description of the application. SG4, SG5 & SG6 are security goals for the organisation, which may (or may not) impose constraints on the functional requirements. We need to operationalise the security goals into constraints on functional requirements.

In order to derive security requirements, each relevant security goal needs to be examined for possible relevance, and then operationalised constraints must be derived from it. For this case study we carry out the process informally. Two separate tasks have to be carried out:

- Use domain knowledge to transform the entities described in the goal into entities described in the functional requirement. In this case the task is straightforward; the security goals refer to personnel information and we know that payroll information, referred to in the functional requirement, is a kind of personnel information.
- Transform Confidentiality, Integrity and Availability into constraints on the operations that are used in functional requirements. This is discussed below.

[5] For simplicity we have again used plausible assumptions.

Security Requirements for Confidentiality

In order to derive constraints for this goal, we need to know who is authorised to access payroll information. In this case we assume that members of Human Resources (HR) Department, and no one else, are authorised for this. We can therefore state the following constraint:

> Req2: The machine shall not display salary information except 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 Payroll 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.[6]

Security Requirements for Availability

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

3.2.4 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; the only goals that are shown are security goals.

3.3 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 System Security Requirements. Note the difference between requirements Req1 (section 3.1) and Req2 (section 3.2.3):

- 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 Req2 is a constraint, which 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.

A design decision is therefore needed before we can deal with the impact of the security requirement Req2 on the Machine.

[6] 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.

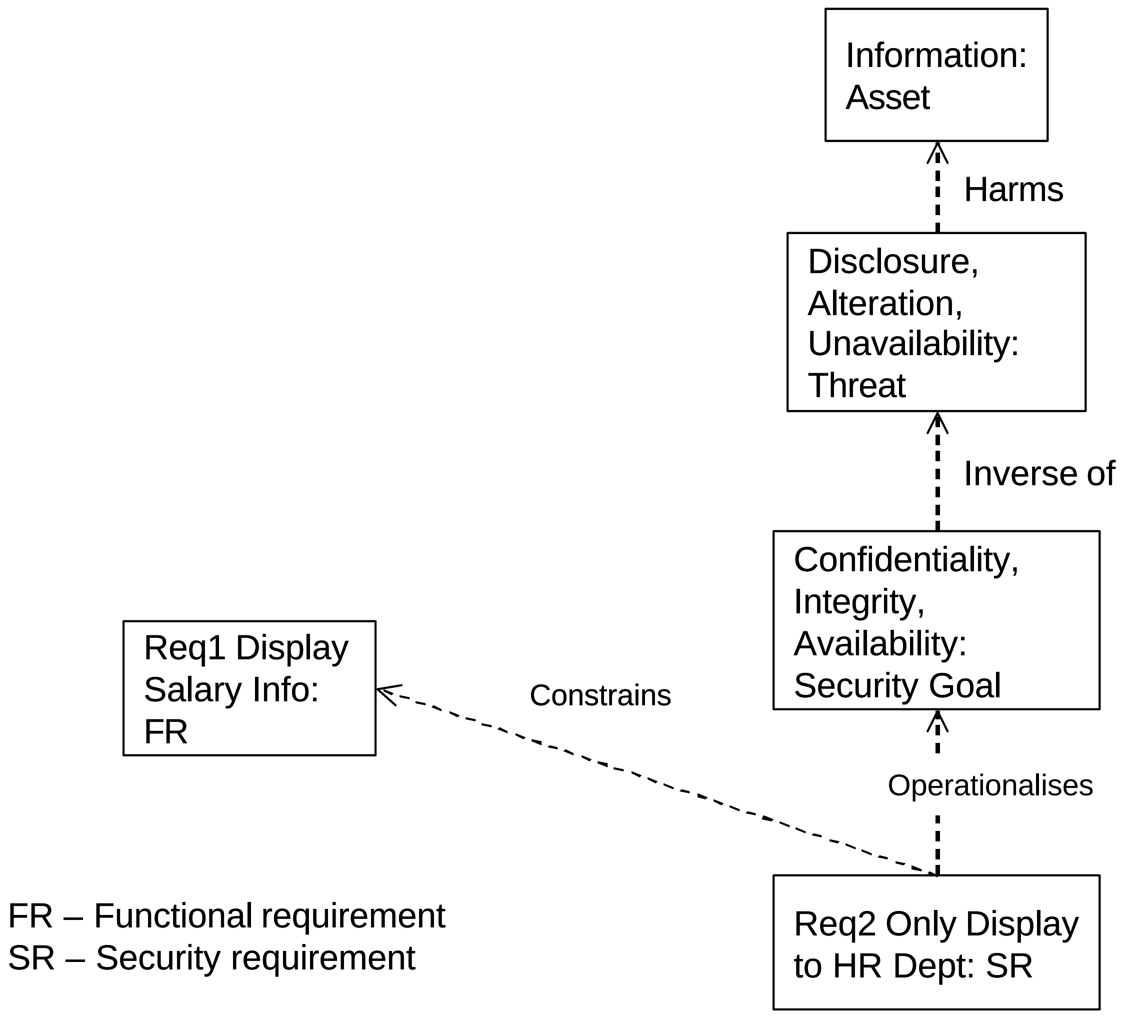

Information: Asset
Harms
Disclosure, Alteration, Unavailability: Threat
Inverse of
Confidentiality, Integrity, Availability: Security Goal
Req1 Display Salary Info: FR
Constrains
Operationalises
FR – Functional requirement
SR – Security requirement
Req2 Only Display to HR Dept: SR

Figure 4: Security Requirements for Display System

3.3.1 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 (Jackson 2000) explicit focus on the computer and its software. There are three main 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 principles, hard-won by experience, require a systems approach.

3.3.2 Introducing the Security Requirements into the Problem Frame

The problem frame diagram does not show the way in which the security requirement Req2 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 or 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 issue, which is relatively straightforward, before discussing the issue of representation of identity in answer to the second one. As we stated, there are two possibilities for the location of the security constraint – in the Person or the Machine domain (or both of them). What follows is an informal architectural design exercise for the system.

3.3.3 Constraining the Person Domain

We could solve the security requirement by requiring the Person domain to contain only members of HR Dept. That implies that no one but HR Dept members can interact with the machine. Although this is not a detailed design exercise, it may be helpful to 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 this design decision, then Req2 is satisfied outside the Machine domain, and there is no need to consider Req2 in the machine's software specification.

3.3.4 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, up to now defined as

```
a: P! {PayrollNumber}
   M! {SalaryInformation(PayrollNumber)}
```

There are several possibilities: add or change the P! interaction; or change the M! interaction.

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

- Encrypting the information in the M! 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.
- Including in the P! interaction a capability (ticket) that can be verified by the Machine. This has similar disadvantages.
- 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 pursue the consequences of changing the interaction to:

> a': P! {PayrollNumber, SourceId, Credentials}
> M! {SalaryInformation}

SourceId is the claimed identity of the person who submits the request. Credentials are information, such as a password, that provides assurance that the interaction has actually been initiated by the person who is identified by SourceId.

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

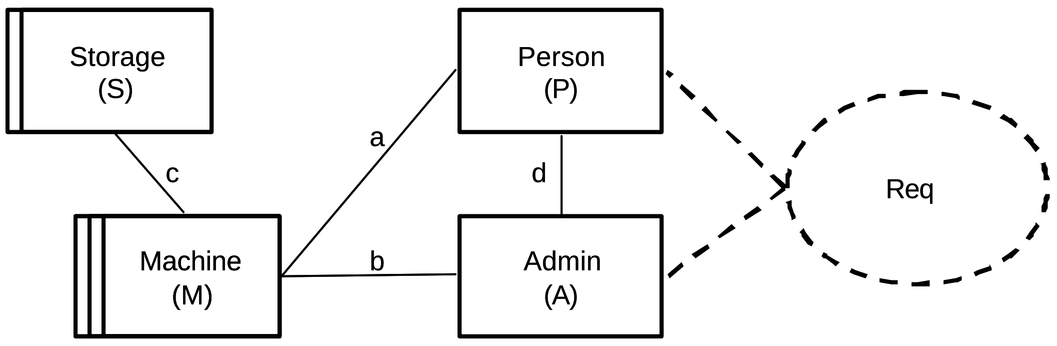
- Use of a personal identity makes it possible to provide Accountability (which requires verification of who has carried out an action), which is often an organisational goal.
- Use of a user / password combination is familiar, implementable in standard ways, relatively cheap and adequately secure for many purposes[7].

3.3.5 Adding Admin to the Problem Frame

However, the solution proposed in section 3.3.4 above has in practice the consequence of requiring the introduction of additional functionality for the administration of identities and credentials. Our problem frame therefore needs to be expanded, as shown in figure 5.

This diagram now shows an Admin domain, which can interact with the Machine to provide the SourceId, Dept and Credentials of People, and with People to agree the SourceId and Credentials with them.

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



a: P! {PayrollNumber, SourceId, Credentials}

M! {SalaryInformation}

b: A! {SourceId , Dept, Credentials}

c : M! {SourceId, Dept, Credentials}

d : A! {SourceId, Credentials}

Figure 5: Payroll Information Display System with Personal Identification Problem Frame Diagram

Requirements

The Requirements have been expanded, with a different Req1, an additional functional requirement and a constraint on the new functional requirement:

> Req1': On request from a Person, the machine shall display salary information for a specified payroll number to that Person. The request shall include the Person's SourceId and Credentials
>
> Req2: The machine shall not display salary information except to members of HR Dept. [unchanged]
>
> Req3: On request from Admin, the machine shall store SourceId, Dept and Credentials information.
>
> Req4: The Credentials for a specified SourceId shall be reliably bound to the Person with that SourceId.

The Derivation of the Additional Requirement

From what process do the alteration of Req1 to Req1' and the new requirements Req3 & Req4 emerge?

Alteration of Req1 to Req1'. The design activity in section 3.3.4 resulted in a solution which could not be expressed in the original Problem Frame, which did not include the concept of explicit identities. It was therefore necessary to go back and augment the Problem Frame for this purpose.

Req3. This is a solution to how the Machine is to obtain the information that is needed for authentication. In practice this may not be a requirement for a new function, but for an already existing function provided by the organisation and the Machine.

Req4. It is necessary to ensure that the Credentials of a Person can be exercised by no one else.

Emerging Security Requirements

A new functional requirement, Req3, has been defined, and new stored information. Since additional functionality has been defined, there is the possibility of the Security Goals being breached. In particular, there is an Integrity Requirement on SourceId, Dept and Credentials information. It will therefore be necessary to perform a risk analysis on the new functionality, and introduce a further security constraint, which (short-circuiting the risk analysis) is:

> Req5: The machine shall not store or display SourceId, Dept and Credentials information except on the request of members of Admin Dept.

Note on Integrity Requirements

Why have we not placed an integrity constraint upon Req3 stating, roughly speaking, that the information must be accurate for each SourceId? This is because we are assuming that there is an organisational goal of Accuracy applying to all functionality, and regardless of threats from malicious attackers. On the other hand, we have defined the constraint Req5, deriving from a security Integrity goal, because a malicious attacker could otherwise cause the Machine to store incorrect information which could be exploited in an attack.

The Machine Specification

The Machine specification has now been expanded, and Spec1 altered to Spec1':

> Spec1': a.P!{PayrollNumber, SourceId, Credentials } **shall cause**
> a.M!{SalaryInformation(PayrollNumber)}
>
> Spec2: **not** (*authenticates*(SourceId, Credentials) **and** *dept*(SourceId) = "HR Dept")
> **shall prevent** a.M!{SalaryInformation(PayrollNumber)}
>
> Spec3: b.A!{ SourceId, Dept, Credentials } **shall cause**
> c.M!{SourceId, Dept, Credentials}
>
> Spec4: **not** (*authenticates*(SourceId, Credentials) **and** *dept*(SourceId) = "Admin Dept") **shall prevent** c.M!{ SourceId, Dept, Credentials }

authenticates is a boolean function and *dept* returns the department of a SourceId.[8]

[8] Note that Spec4 presents a well-known chicken and egg problem (who introduces the first adminstrator?), which can be solved in various more-or-less satisfactory ways.

Why have we not used a conventional notation, such as Statecharts, for the machine specification? The reason is the need for a strong "prevent" with global scope, to represent the security constraint, as discussed in section 2.4.2 above. A guard on a state transition is local to that transition, and will not prohibit the firing of a separate transition, which violates that guard.

It will, of course, be necessary to show that the Machine specification, together with the properties of the other domains, satisfies the Requirements. This will involve us, first of all, in specifying the behaviour of Persons in HR Dept, and then composing the specifications. It is not done here, for brevity.

3.3.6 The Twin Peaks Revisited

Let us assume that we follow the path set out in section 3.3.4 above. 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 3.2.3
- A design activity in section 3.3.4, using the SourceId approach, which is infeasible with the original Problem Frame
- A modification of the Problem Frame in section 3.3.5, which now enables us to define a software specification that, given our knowledge of Domain properties, will satisfy our System Requirements.

It is clear that a Twin Peaks approach is essential in any practical use of this framework.

4. Discussion

A number of issues have arisen in the course of proposing the framework

4.1 The Place of Analysis in the Framework

Analysis, both to support development and to validate its results, is essential to this framework. There are two main stages at which it is relevant:

- The refinement of security requirements from security goals
- The specification of domain properties, including software specification, which will together satisfy the security requirements.

Aside from techniques of general applicability, such as formal methods, we are aware of three kinds of technique, which are specifically relevant to the analysis of security requirements:

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

Goal Refinement

(van Lamsweerde and Letier 2000) has already demonstrated how Obstacles can be used in goal-oriented requirements engineering to identify the failure of specific goals to meet more general ones, and i* (Liu, Yu et al. 2003) provides a complementary technique. As one would expect, they appear to be mainly relevant to the first stage, goal refinement.

Abuse and Misuse Cases

Abuse and misuse cases, exemplified by (Alexander 2002) are techniques that have been developed to elicit security requirements. However, in our view they cannot be regarded as a complete technique for this purpose, although they are clearly useful techniques for identifying gaps in security requirements, and for validating them after they have been defined. 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 are a technique, currently under development (Lin, Nuseibeh et al. 2003), that identifies possible domain (including software) properties ("illicit" behaviour) that could cause the system security requirements to be broken:

- This could be illicit behaviour that is permitted by the combined properties of the domains
- There could also be illicit behaviour that is permitted as a result of perturbation of the problem frame, by changing either the domain properties or the frame topology.

This is a technique which bridges the gap between goal refinement analysis and design analysis. It is the subject of further research.

4.2 Revised Security Risk Analysis Model

It is clear that the security risk analysis model presented in section 2.2 above is no longer adequate for the purpose. Although the asset identification and threat identification & evaluation stages remain valid, vulnerability analysis needs to be apportioned between several life cycle stages, as does risk assessment and decisions about security measures. This is not surprising, as no thought about life cycle stages has normally been given in security risk analysis, which has in the past typically been carried out *post hoc* on existing operational systems.

Proposal of a revised security risk analysis model is future work, but we can already identify some aspects of vulnerability analysis and security measures that belong within security requirements engineering.

Two activities which could be described as vulnerability analysis have already been discussed in section 4.1 above:

- Analysis of goal refinement, in order to identify obstacles that prevent security requirements from meeting the security goals
- Analysis of the problem frame in order to discover abuse frames.

We also observe that the specification of security constraints on operations could be regarded as the first step towards decisions about security measures, and the realisation of system security requirements in the course of defining a software requirements specification takes this a step further.

What is clear is that a relatively self-contained security risk analysis process will need to replaced with one that is intertwined with the processes of the software engineering life cycle. In that respect, there may be something to learn from the development of safety-critical systems, for which (IEC 61508) defines a life-cycle for safety analysis and its relationship to functional development.

4.3 Security Policies

"Security Policies" is a phrase used extensively in the literature, and it is not our intention to review the relevant literature here. We note that the phrase can refer to a wide variety of concepts, from high level goals in an organisation's corporate policy document, to security requirements, such as (Brewer and Nash 1989), to design principles such as the Bell & Lapadula "policy" model (Bell and LaPadula 1976). We believe that our proposed framework will make it easier to categorise security policies according to the stage at which they are defined.

4.4 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 and Anderson 1990), and it should be possible to make progress on the requirements of security fault-tolerant systems.

4.5 The Need for 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 two main issues to be dealt with:

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

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

operation, its originator and source; temporal constraints; and constraints on traces of the behaviour of a system.

Availability Requirements

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 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.

4.6 Specification Language

Since this is a framework, intended to encompass diverse means of expression, we do not intend to recommend any particular specification language. 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 constraints. However, there are at least two general issues of specification language, which need further research:

- How to state security requirements without using too many negatives;
- The scope of stated security requirements.

Stating Security Requirements

The reader will have noticed some awkwardness in the language used in section 3 above. 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

It would of course be possible to use an Open World assumption, with the default and overrides reversed, although this is not regarded as good security practice.

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.

The Scope of Security Requirements

We pointed out in section 2.4.2 above, that the scope of security requirements must be global. This implies that notations, such as state transition diagrams, which only have local scope for their guards or constraints cannot model security requirements. A language such as causal logic (Moffett, Hall et al. 1996) does not suffer from this limitation, although it needs development if it is to be of general use.

4.7 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.

5. Conclusions

This paper has set out a framework for security requirements, which has several features:

- Security requirements are derived from security goals, and take the form of constraints on the functions of a system.
- Security requirements are therefore automatically integrated with the system's functional requirements and constraints derived from other sources. For example, if Safety requirements are also defined in terms of constraints on operations, security and safety constraints are expressed in identical terms and the analysis of their interaction is directly possible.
- It is essential to define security requirements in terms of the real-world assets of a system. Their realisation in software (and by physical and procedural means also) must then be shown to satisfy these 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 the effective co-operation of the two disciplines of requirements and security.

Acknowledgements

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

References

Alexander, I. (2002). "Misuse Cases in Systems Engineering." *Computing and Control Engineering Journal* **13**(6): 289-297.

Anderson, R. (1996). *Security in Clinical Information Systems*. IEEE Symposium on Security and Privacy, Oakland, CA.

Antón, A. I. and J. B. Earp (2001). Strategies for Developing Policies and Requirements for Secure E-Commerce Systems. *Recent Advances in E-Commerce Security and Privacy*. A. K. Ghosh, Kluwer Academic Publishers**:** 29-46.

Baskerville, R. (1993). "Information Systems Security Design Methods: Implications for Information Systems Development." *ACM Computing Surveys* **25**(4): 375-414.

Bell, D. E. and L. J. LaPadula (1976). *Secure computer system: Unified exposition and Multics interpretation*. MTR-2997, (ESD-TR-75-306), available as NTIS AD-A023 588. MITRE Corporation.

Brewer, D. F. C. and M. J. Nash (1989). *The Chinese Wall Security Policy.* IEEE Symposium on Security and Privacy, Oakland, CA, IEEE Computer Society Press.

Dardenne, A., A. van Lamsweerde, et al. (1993). "Goal-directed Requirements Acquisition." *Science of Computer Programming* **20**: 3-50.

Heitmeyer, C. (2001). *Applying `Practical' Formal Methods to the Specification and Analysis of Security Properties*. Information Assurance in Computer Networks (MMM-ACNS 2001), St. Petersburg, Russia, Springer-Verlag.

IEC 61508. *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems* 1998 ISBN.

Jackson, M. (2000). *Problem Frames: Analysing and Structuring Software Development Problems*, Addison Wesley. ISBN 020159627X.

Kotonya, G. and I. Sommerville (1998). *Requirements Engineering - Processes and Techniques*, John Wiley. ISBN 0 471 97208 8.

Lee, P. A. and T. Anderson (1990). *Fault Tolerance: Principles and Practice*, Springer-Verlag. ISBN 3211820779.

Lee, Y., J. Lee, et al. (2002). "Integrating Software Lifecycle Process Standards with Security Engineering." *Computers & Security* **21**(4): 345-355.

Leveson, N. G. (1995). *Safeware: System Safety and Computers*, Addison Wesley. ISBN 02011 19722.

Lin, L., B. Nuseibeh, et al. (2003). *Introducing Abuse Frames for Analysing Security Requirements (poster presentation)*. RE'03: 11th IEEE International Requirements Engineering Conference, Monterey Bay, CA, USA.

Liu, L., E. Yu, et al. (2003). *Security and Privacy Requirements Analysis within a Social Setting*. RE'03 - 11th IEEE International Requirements Engineering Conference, Monterey Bay, CA, USA.

McDermott, J. and C. Fox (1999). *Using Abuse Case Models for Security Requirements Analysis*. Annual Computer Security Applications Conference, Phoenix, Arizona.

Mitnick, K. (2002). *The Art of Deception: Controlling the Human Element of Security*, John Wiley & Sons Inc. ISBN 0471237124.

Moffett, J. D., J. G. Hall, et al. (1996). "A Model for a Causal Logic for Requirements Engineering." *Journal of Requirements Engineering* **1**(1): 27-46.

Nuseibeh, B. A. (2001). "Weaving Together Requirements and Architectures." *IEEE Computer* **34**(3): 115-117.

Peltier, T. (2001). *Information Security Risk Analysis*, Auerbach. ISBN 0-8493-0880-1.

Rushby, J. (2001). *Security Requirements Specifications: How and What?* Symposium on Requirements Engineering for Information Security (SREIS), Indianapolis.

Sindre, G. and A. L. Opdahl (2000). *Eliciting Security Requirements by Misuse Cases*. 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-PACIFIC 2000), IEEE Computer Society Press.

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

Zwicky, E. D., S. Cooper, et al. (2000). *Building Internet Firewalls*, O'Reilly UK. ISBN 1565928717.