Security Requirements Engineering: A Framework for Representation and Analysis

Security Requirements Engineering: A Framework for Representation and Analysis

Charles B. Haley, Robin Laney, Jonathan D. Moffett, *Member*, *IEEE*, and
Bashar Nuseibeh, *Member*, *IEEE Computer Society*

Abstract—This paper presents a framework for security requirements elicitation and analysis. The framework is based on constructing a context for the system, representing security requirements as constraints, and developing satisfaction arguments for the security requirements. The system context is described using a problem-oriented notation, then is validated against the security requirements through construction of a satisfaction argument. The satisfaction argument consists of two parts: a formal argument that the system can meet its security requirements and a structured informal argument supporting the assumptions expressed in the formal argument. The construction of the satisfaction argument may fail, revealing either that the security requirement cannot be satisfied in the context or that the context does not contain sufficient information to develop the argument. In this case, designers and architects are asked to provide additional design information to resolve the problems. We evaluate the framework by applying it to a security requirements analysis within an air traffic control technology evaluation project.

Index Terms—Requirements engineering, security engineering, security requirements, argumentation.

1 Introduction

In recent years, reports of software security failures have become commonplace. Statistics from the Software Engineering Institute's CERT Coordination Center, a center of Internet security expertise, show that the number of reported application vulnerabilities rose from 171 in 1995 to 5,990 in 2005 [12]. One source of security problems is not considering the security requirements of the complete system. For example, CardSystems Solutions exposed details of some 40 million credit cards by storing historical transaction history data where hackers could get to it [21]; the data became part of their system but was not part of their security planning. The resulting loss has not been disclosed, but is in excess of several million dollars [24]. Another source is not considering security in the application itself; dictionary password and PIN attacks with no "multiple failure" lockout come to mind. These examples suggest that improving software-based system security would have a significant financial impact.

This paper explores how to determine *adequate security requirements* for a *system*. By *adequate security requirements*, we mean requirements that, if respected, lead to a system's security goals being satisfied and by *system* we mean the software and, in addition, people who use the software, and equipment around the software (computers, printers, etc.). Adequate general requirements have been shown to have a very positive impact on the acceptance of a project's deliverable: For examples see the Standish Group's Chaos reports [72], [73], [74], and the introduction to [55]. Although the empirical evidence is not yet unequivocal, it appears that adequate security requirements will have as positive an impact on system security as adequate general requirements have on system success [55].

We claim that adequate security requirements must satisfy three criteria. The first criterion is *definition:* One must know what security requirements are. The second is *assumptions:* Security requirements must take into consideration an analyst's implicit or explicit assumption that an object in the system will behave as expected. The third is *satisfaction:* One must be able to determine whether the security requirements satisfy the security goals and whether the system satisfies the security requirements. We propose three contributions to assist with developing security requirements that satisfy these criteria. The first is a practical definition of security requirements, with yes/no satisfaction criteria within a system context. The second is an explicit role for assumptions, concentrating on their place in security requirements satisfaction arguments. The third is the use of formal and informal structured arguments to validate that a system can satisfy its security requirements. These three contributions are incorporated into our security requirements framework, facilitating the elicitation, validation, and verification of security requirements and other artifacts.

This paper extends and unifies our previous work: extends by reporting of the results of an application of our framework and unifies by showing how our earlier contributions can be applied together coherently and effectively. The security requirements framework was published in [57] and further refined in [35]. Trust assumptions are described in [32], [36]. Threat descriptions, which we use for our threat modeling, were introduced in [31]. Our work on security satisfaction arguments has been published in [33], [34].

- *The authors are with the Department of Computing, Centre for Research in Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-mail: {c.b.haley, r.c.laney, j.moffett, b.nuseibeh}@open.ac.uk.*

Manuscript received 5 Nov. 2006; revised 6 July 2007; accepted 4 Sept. 2007; published online 1 Oct. 2007.
Recommended for acceptance by A. Wolf.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0249-1106.
Digital Object Identifier no. 10.1109/TSE.2007.70754.

The remainder of this paper is structured as follows: Section 2 presents background and motivation, further exploring the three criteria presented above. Section 3 introduces our framework and our two-part satisfaction arguments. Section 4 shows how the arguments are used within our framework. Section 5 presents the application of our framework in an air traffic control technology evaluation project, while Section 6 discusses lessons learned during that project. Section 7 reviews related work and Section 8 concludes.

2 Background and Motivation

We claimed above that adequate security requirements must satisfy three criteria, *definition*, *assumptions*, and *satisfaction*, and that the contributions presented in this paper assist with producing security requirements that satisfy these criteria. We explore these criteria in this section, providing background and motivation for our contributions.

2.1 Criterion One—Definition of Security Requirements

Security needs arise when stakeholders establish that some resource involved in a system, be it tangible (e.g., cash) or intangible (e.g., information or reputation), is of value to the organization. Such resources are called *assets* [14], [38] and the stakeholders naturally wish to protect themselves from any harm involving these assets. Security goals express this desire, describing the involved asset(s) and the harm to be prevented. The difficulty then is to determine, from these security goals, clear security requirements that respond to the needs of a system.

2.1.1 Security Requirements as Nonfunctional Requirements (NFRs)

Security requirements have traditionally been considered to be "nonfunctional" or "quality" requirements ([15], [23], [27], [30], and many others). Like other quality requirements (e.g., performance, usability, cost to run), they do not have simple *yes/no* satisfaction criteria. Instead, one must somehow determine whether a quality requirement has been *satisficed* (satisfied well enough) [60]. This is difficult for quality requirements in general and security requirements present additional challenges. First, once one descends from the very general and obvious statements (e.g., "the system should provide confidentiality"), people generally think about and express security requirements in terms of "bad things" to be prevented. Unfortunately, verifying that something is prevented entails proving a negative: that there are no counterexamples. It is very difficult, if not impossible, to measure negative properties. Second, for security requirements, the tolerance on "satisfied enough" is small, often zero; stakeholders want security requirement satisfaction to be very close to yes/no. Third, the amount of time and money that stakeholders might be willing to dedicate to satisfying a security requirement can depend on the *likelihood* and *impact* of a security failure; one cannot justify a large expense to protect something of low value. One must be able to connect specific development and operational expense to the requirements being satisfied in order to determine cost/benefit information.

Security requirements that express what is to happen in a given situation, as opposed to what is not ever to happen in any situation, would facilitate their analysis. Such requirements would have binary satisfaction criteria, either behaving appropriately in the given situation or not, and one can have test criteria to determine what "appropriately" means. The cost of ensuring behavior in a given situation is easier to measure than the cost of ensuring something never happens, facilitating cost/benefit analysis.

2.1.2 Security Requirements and Context

System context can have a profound effect on security goals and security requirements. As indicated earlier, in this paper, the word *system* represents more than the software. We include the environment the software runs within: the people who will use, maintain, and depend on the system; the physical environment the system is to exist within; the operating environment the software runs within; and any systems, computer-based and otherwise, already in place. Security requirements can vary, depending on the system context. To illustrate, consider some software used by an executive on her desktop computer. The software might not have any intrinsic need for security; a spreadsheet would be a good example. Even though the spreadsheet program may have no intrinsic security goals associated with it, the information the executive manipulates may be confidential, creating a *maintain confidentiality* security goal for the system comprised of the computer, the office, the spreadsheet program, the executive, and the confidential data. The security goal arises because of how the spreadsheet is used, which is a property of the context within which the program resides. When the system components {computer, office, spreadsheet program, executive} are considered alone, no confidentiality security goal arises. The goal arises only when {confidential data} is added.

Continuing the example, one might consider satisfying the confidentiality goal by adding a locking office door to the system, something completely divorced from the software. Alternatively, one might decide that the spreadsheet program should satisfy the goal, perhaps by adding authentication and encryption. However, these solutions are inadequate if the executive is in an office that is not soundproofed and either a) she uses a program that reads the information aloud, permitting an attacker to listen without being seen, or b) if the attacker can hear and decode the keystrokes typed on her keyboard [82].

2.2 Criterion Two—Incorporation of Assumptions about Behavior

When considering system behavior, the analyst must decide which parts of the world to consider as part of the problem and therefore to include in the analysis. An extreme view is that every atom in the universe is part of every problem and, therefore, the analysis must consider everything made of atoms. As this is clearly impractical, the analyst must choose a subset of *domains* (real-world elements) that s/he considers relevant [41], [42]. By so choosing, the analyst defines the system context.

When considering security, one factor influencing an analyst's choice about whether or not a domain is relevant is the analyst's set of *trust assumptions* [79], [80]. Trust assumptions are explicit or implicit choices to trust a domain to behave as expected and can have a significant impact on the security of a system. For example, most analysts would not consider the compiler a security risk and would not include it in a security analysis. In his 1983 Turing award acceptance lecture [76], Ken Thompson demonstrated that this assumption might not be justified by showing how a compiler could be a Trojan horse, introducing trapdoors into applications. Viega et al. [79] claim that "application providers often assume that their code will execute in a non-hostile environment" and then show how this assumption leads to security breaches. Their example shows "secrets" hidden in code, where the secrets can be easily exposed through examination of the executable file. The Thompson and Viega et al. examples illustrate how implicit trust of some domains in the environment can introduce unknown amounts of risk into the system. Viega et al. go as far as to say that "without recognizing all the entities and their trust relationships in a software system during the requirements phase of a project, that project is doomed from the start" [79].

2.3 Criterion Three—Satisfaction of Security Requirements

If one goes to the trouble to produce security requirements for a system, it is reasonable to ask whether the system can satisfy the requirements. The more rigorous the process used to establish satisfaction, the more confidence one can have that the system will be secure. The strongest process is a proof. A weaker alternative to a proof is an argument. A high-quality argument engenders confidence that the requirements will be satisfied. The weaker the argument, the more faith one must have that the result will be acceptable.

No analysis of security requirement satisfaction can include every possible domain that could be a part of the system. Every proof or argument will include trust assumptions, at minimum that the domains not considered will do no harm, and establishment of satisfaction depends upon the validity of these trust assumptions. Rigorous proofs of validity of trust assumptions are hard to come by because malice and accident must be taken into account. Instead of proving that a trust assumption *is* valid, one can instead produce arguments that the trust assumption *should be considered* valid. The argument must be sufficiently convincing, using properties of the system and domains as evidence.

Trust assumption validity arguments are, in effect, subarguments of the proof or argument that security requirements are satisfied and their quality directly affects the validity of the containing argument. The subarguments should be an explicit part of establishing satisfaction of security requirements.

3 A Framework for Security Requirements Engineering

We propose a security requirements engineering framework to facilitate production of security requirements satisfying the three criteria presented above. In the framework:

- Security goals and security requirements aim to protect assets from harm.
- Primary security goals are operationalized into primary security requirements, which take the form of constraints on the functional requirements sufficient to protect the assets from identified harms. Primary security requirements are, consequently, preventative.
- Feasibility, trade-off, and conflict analyses [67, p. 81] may lead to the addition of secondary security goals, which result in additional functional and/or secondary security requirements. Secondary security goals and requirements may call for detective or preventative measures, a possibility further discussed below.
- Security satisfaction arguments show that the system can respect the security requirements. These arguments expose and take trust assumptions into account.

The framework assists with understanding the place of security requirements within the development of an individual application, along with the relationships between the security requirements and other artifacts produced during development.

3.1 Definition of Security Goals

The security community has enumerated some general security concerns: confidentiality, integrity, and availability (labeling them CIA and, more recently, adding another A for accountability [64]). By enumerating the assets in a system, then postulating *actions* that would violate these security concerns for the assets, one can construct descriptions of possible threats on assets. For example, one can *erase* (the action) *customer records* (the asset) of a company to cause *loss of revenue* (the harm). One set of security goals is determined by listing these threats on assets, then *preventing* (or *avoiding*) the action(s) on the asset(s) that realizes the threat.

More security goals can be found by combining management control principles and organization security policies. Management control principles include "standard" security principles such as least privilege and separation of duties [64]; these are checked against assets to determine their applicability. An organization may already have done the analysis and published policies that apply to assets in a system. The security goal is a statement that the policies and/or principles be applied where appropriate in the system.

Note that stakeholders may have conflicting security goals. 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.

Knowing the goals of attackers could be useful when determining security goals for the system, for example when enumerating assets or quantifying harm. However, the goals of the system owner and other legitimate stakeholders are not directly related to the goals of attackers because security is not necessarily a zero sum game like football (soccer in the US). In football, the goals won by an attacker are exactly the goals lost by the defender. Security is different; there is not necessarily a relationship between the losses incurred by the asset owner and the gains of the attacker. For example, Robert Morris unleashed the Internet Worm [71], causing millions of dollars of damage, apparently as an experiment without serious malicious intent. The positive value to Morris was much less than the loss incurred by the attacked sites.

The consequences of security not being a zero sum game are twofold: The first is that the evaluation of possible harm to an asset can generally be carried out without reference to particular attackers; one need only determine that harm can be incurred. The second is that the goals of attackers cannot be used alone to arrive at the goals of a defender to prevent harm, i.e., the system's security goals; further consideration is necessary to determine whether and what harm to the system is incurred if the attacker satisfies his or her goals.

3.2 Definition of Security Requirements

We define security requirements as constraints on the functions of the system, where these constraints operationalize one or more security goals. Security requirements operationalize the security goals as follows:

- They are constraints on the system's functional requirements, rather than being themselves functional requirements.
- They express the system's security goals in operational terms, precise enough to be given to a designer/architect. Security requirements, like functional requirements, are prescriptive, providing a *specification* (behavior in terms of phenomena) to achieve the desired effect.

3.3 From Security Goals to Security Requirements

We propose an iterative hierarchy of security goals and security requirements. The first iteration produces primary goals and requirements that are [directly] derived from the business goals and functional requirements. Further iterations produce *secondary* security goals and requirements. They are added for one or both of the following reasons: to enable construction of an acceptable satisfaction argument for the satisfaction of primary security requirements (see Section 4) or to permit an acceptable *feasible realization* of the primary security requirements.

The term *feasible realization* takes into consideration technical feasibility, cost/benefit plus risk, and stakeholder trade-offs [67]. It may be that no practical way exists to respect a constraint and thus prevent the harm. Perhaps stakeholders do not agree on the goals or requirements. Risk analysis may indicate that the cost of respecting a security requirement is excessive, in which case the analyst may decide to detect violation after the fact and then recover from and/or repair the breach. Consider availability requirements; many such requirements do not prevent loss of availability, but instead imply a recovery capability.

Secondary security goals and security requirements are not secondary in terms of importance, but are secondary because they exist to enable satisfaction, to an acceptable level, of the security requirements from which they are derived.

It is very important to note that secondary security goals and requirements supersede the primary security requirements and can change the context and behavior of the system. For example, choosing to use attack detection instead of prevention implies that the primary security requirement will not be completely satisfied as the attack will not be prevented. The choice means that the secondary goals and associated security requirements are considered *suitably equivalent* to the primary security requirements; they "cover" and "replace," but do not delete them. Secondary security goals can be satisfied by adding either or both of secondary functional requirements and secondary security requirements. Using an authentication mechanism is an example of adding both; the mechanism must be added (function), then used in a secondary security requirement (a constraint). An example of satisfying a secondary security goal by adding function is addition of management capabilities for detecting breaches.

3.4 Development Artifacts and Dependencies

All system development processes have recognizable stages that produce artifacts that are successively closer representations of a working system. These representations are *core artifacts*. They are ordered in the abstraction hierarchy shown in Fig. 1, progressing from the most abstract to the final concrete working system. At early stages, core artifacts are typically documents, models, or prototypes. The final core artifact is the working system itself, consisting of a combination of physical and software items.

Support artifacts are artifacts that help to develop, analyze, or justify the design of a core artifact. They may include formal analysis, informal argument, calculation, example or counterexample, etc. They are byproducts of processes whose aim is to help produce verified and valid core artifacts.

Two sets of core artifacts are of most interest in this paper. On the mainstream requirements engineering side, one finds descriptions of goals, functional requirements, and the system context/architecture (in the large). On the security engineering side, one finds assets, control principles, and security goals and requirements.

Dependencies between Artifacts. There are dependencies in the hierarchy. For example, an operationalized requirement is dependent upon a higher-level goal from which it has been derived because alteration of the goal may cause alteration of the requirement. We 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 sufficiently satisfies its goals or requirements, then the goals or requirements must be changed. The higher-level artifact is dependent on the feasibility of the artifacts below it in the hierarchy.

These dependency relationships have an important implication for the structure of development processes. If an artifact is dependent upon the implementation of

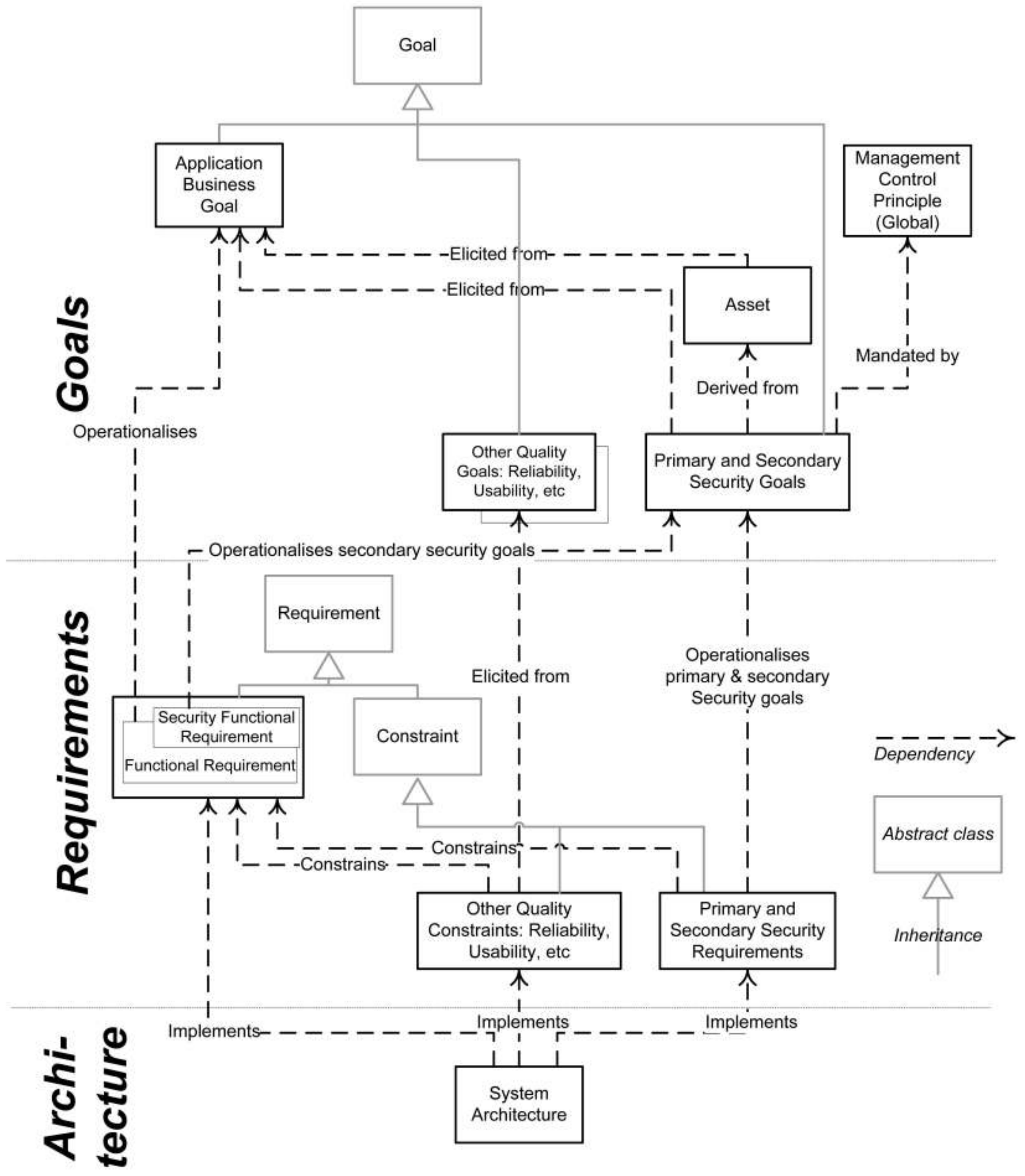

Goal
Application Business Goal
Management Control Principle (Global)
Elicited from
Elicited from
Asset
Mandated by
Derived from
Goals
Operationalises
Other Quality Goals: Reliability, Usability, etc
Primary and Secondary Security Goals
Operationalises secondary security goals
Requirement
Elicited from
Operationalises primary & secondary Security goals
Requirements
Security Functional Requirement
Functional Requirement
Constraint
Dependency
Abstract class
Constrains
Constrains
Other Quality Constraints: Reliability, Usability, etc
Primary and Secondary Security Requirements
Inheritance
Implements
Implements
Implements
Archi-tecture
System Architecture

Fig. 1. Security requirements core artifacts (class diagram).

another artifact 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.

3.5 Activities in the Framework

An ordered set of activities for moving from functional goals to satisfaction arguments is shown in Fig. 2. Boxes in the figure represent activities that produce artifacts. Typically, a box in the figure has two exits, one for success and one for failure. Failure can be one of two kinds. The first is that it is not feasible to create a consistent set of the artifacts to be constructed by that activity. The second is that validation of the artifacts against a higher level, for example, validation of security requirements against security goals, shows that they fail to meet their aims. Iteration may cascade upward if the problem cannot be resolved at the preceding step.

There are four general stages in the activity diagram. Although one could describe these stages in terms of the artifacts that are produced, along with the ordering between them, it is clearer to describe them as activities that are to be incorporated into the development process. The activities are

1. identify functional requirements,
2. identify security goals,
3. identify security requirements, and
4. construct satisfaction arguments.

We discuss each in turn below.

3.5.1 Stage 1: Identify Functional Requirements

The only requirement the framework places upon this stage of the development process is that a representation of the system context be produced. (See Section 3.5.3 for more detail on the context.) How the requirements engineer gets to this point is left open.

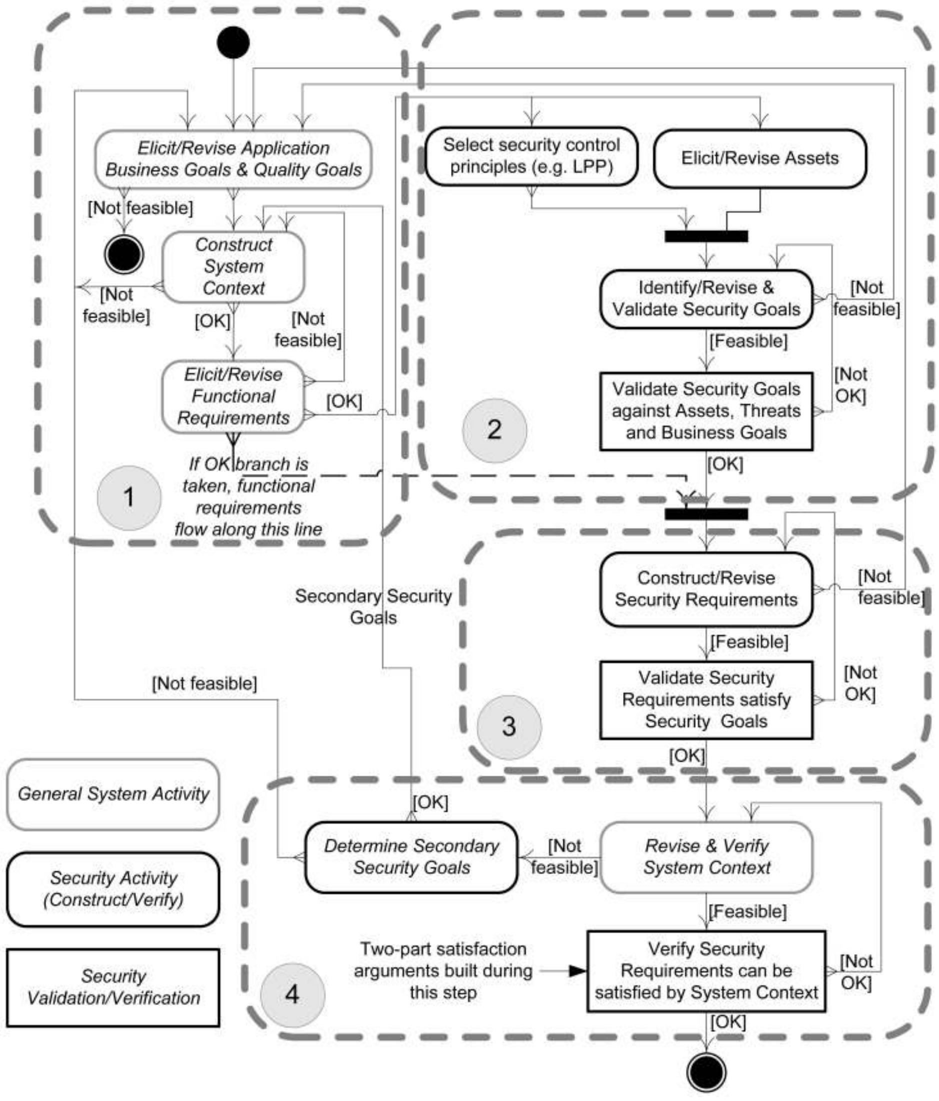

Elicit/Revise Application Business Goals & Quality Goals
[Not feasible]
Construct System Context
[Not feasible]
[OK]
[Not feasible]
Elicit/Revise Functional Requirements
[OK]
If OK branch is taken, functional requirements flow along this line
1
Select security control principles (e.g. LPP)
Elicit/Revise Assets
Identify/Revise & Validate Security Goals
[Not feasible]
[Feasible]
Validate Security Goals against Assets, Threats and Business Goals
[Not OK]
2
[OK]
Secondary Security Goals
Construct/Revise Security Requirements
[Not feasible]
[Feasible]
Validate Security Requirements satisfy Security Goals
[Not OK]
3
[OK]
[Not feasible]
General System Activity
Security Activity (Construct/Verify)
Security Validation/Verification
[OK]
Determine Secondary Security Goals
[Not feasible]
Revise & Verify System Context
[Feasible]
Two-part satisfaction arguments built during this step
Verify Security Requirements can be satisfied by System Context
[Not OK]
4
[OK]

Fig. 2. Process activity diagram.

3.5.2 Stage 2: Identify Security Goals

There are three general steps required to identify the security goals: identify candidate assets, select the management principles to apply, and then determine the security goals. The result is a set of security goals which are validated by ensuring that the business goals remain satisfied.

The first iteration through this step results in the generation of primary security goals. Subsequent iterations result in secondary security goals, which are traceable, perhaps through multiple levels and through security requirements, to the original, primary, security goal(s).

Identify candidate assets. The goal of this step is to find all of the resources in the system context that might have value. In general, assets consist of all of the information resources stored in or accessed by the system-to-be and any tangible resources such as the computers themselves. Assets can be composed of other assets; backup tapes would be a good example.

Select management principles. The functions that the system is to provide must be compared to the management principles that the organization wishes to apply. These principles might include separation of duties, separation of function, required audit trails, least privilege (both need to *know* and need to *do*), Chinese wall, data protection, no outside connections, and no removable media (not intended to be an exhaustive list). The organization might have already done a harm/risk analysis and developed organization-wide security policies for asset types. Which global policies to apply within the system under consideration must be identified and fed into the next step.

Determine security goals. When developing security goals, one must determine whether a harm analysis must be done for the assets. If the analysis has been done elsewhere (e.g., organization-wide policies) and if the assets are covered by the policies, then a list of security goals is generated by applying the management principles to the assets and business goals of the system. The result is a set of *achieve* goals with forms similar to “achieve Separation of Duties when paying invoices” or “audit all uses of account information.”

If the analysis done elsewhere is not considered sufficient, one should do a harm analysis. In general, harm is caused by the negation of the security concerns described in Section 3.1: *confidentiality*, *integrity*, *availability*, and *accountability*. One asks questions of the form "what harm could come to [insert asset here] from an action violating [insert concern here]?" Answers to these questions are threat descriptions [31], which are represented as tuples of the form {action, asset, harm}. Security goals are constructed by specifying that the action(s) on the asset(s) listed in threat descriptions be *prevented*.

The goals identified from the two analyses (if both are done) must be combined and checked to ensure that they are consistent.

3.5.3 Stage 3: Identify Security Requirements

Recall that we define security requirements as constraints on functional requirements that are needed to satisfy applicable security goals. To determine the constraints, we must determine which security goals apply to which functional requirements, which means we must know which assets are implicated in fulfilling a particular functional requirement. We use Jackson's problem diagrams [42] for this purpose; these diagrams describe the *system context*. We do not attempt to identify a particular problem class (problem frame), but instead describe the domains, their interconnections, shared phenomena, and requirements in a system problem diagram.

A simple example of such a constraint is The system shall provide Personnel Information only to members of Human Resources Dept. The constraint ("only to ...") is attached to the function ("provide Personnel Information"); it makes sense only in the context of such a function. One might also impose temporal constraints, e.g., The system shall provide Personnel Information only during normal office hours, or a complex constraint on traces produced by the Chinese Wall Security Policy, [7]:

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

Once a set of security requirements has been developed, one must validate that the security requirements satisfy the security goals. This would be done using satisfaction arguments appropriate to the level of formality used to describe the goals. Given that goals are often written in plain text, the arguments could have a form similar to our inner arguments (see Section 3.5.4). How these arguments are expressed is left open to the designer of the process to be used and are not defined within our framework.

In the same fashion as security goals, the first iteration through this step results in primary security requirements. Subsequent iterations generate secondary security requirements.

3.5.4 Stage 4: Verification of the System

It is important to verify that the security requirements are satisfied by the system as described by the context. We propose two-part satisfaction arguments for this verification step: to convince a reader that a system can satisfy the security requirements laid upon it. The first part, the *outer argument*, consists of a formal argument to prove that the instance of the system under examination satisfies its security requirements, with two important assumptions: that the context is correct and that the implementation will not introduce any conflicting behavior. We recognize that both of these assumptions are very strong and often untrue in practice. Verification that the system can satisfy the requirements cannot ensure the truth of the assumptions, but it does ensure a sound structure for the system that is potentially secure.

The second part, the *inner argument*, consists of structured informal arguments to support the assumptions about system composition and behavior made in the formal argument. Satisfaction arguments assist with identifying security-relevant system properties and determining how inconsistent and implausible assumptions about them affect the security of a system. These security requirement satisfaction arguments are presented in more detail in Section 4.

3.6 Iteration

One reason that an analyst may fail to construct a convincing satisfaction argument is that there is not enough information available to justify an assumption. For example, to justify a claim that users are authenticated, there must be some phenomena exchanged between the user and the rest of the system. The choice of phenomena and behavior is a design decision that may have a significant impact on the system architecture and context. For example, it is possible that architectural choices may have already been made and are being imposed. For these reasons, the framework assumes that the process includes *Twin Peaks* iterations [63], concurrent and incremental elaboration of requirements and design. Designers add detail into the system context so claims can be justified. These iterations move from stage four to stages one and two.

The details added during an iteration may well require new functions, thus functional requirements. Consider a system where, to satisfy a confidentiality requirement, designers choose authentication and authorization. Further assume the choice of a retinal-scanning authentication technique. Appropriate domains and behavior are added to the context to describe how authentication takes place from the point of view of the user (in problem space). However, one cannot necessarily stop at the addition of domains and phenomena. The authentication system may need to be managed. New assets may have been added to the system: for example, the retina description information. New domains have been added: for example, the administrators. The process would then restart in stage 1 with a reanalysis of the functional requirements so that the consequences of the new goal are understood. New assets (e.g., the authentication data) would be found in stage 2 and then new security goals to protect the assets and new security requirements to constrain functional operations wherever the new asset appears would be added.

Another possibility is that an iteration will establish that there is no feasible way to satisfy the security requirement(s). In this case, the designers and the stakeholders must come to an agreement on some acceptable alternative, such as a weaker constraint, attack detection, and/or attack recovery. Appropriate secondary security goals are added, probably resulting in new secondary security requirements. The resulting secondary security goals and requirements "cover" the ones that were not feasible. Satisfying the new secondary goals and requirements is considered to satisfy the original security goals and requirements. Clearly, the "secondariness" of any functional goals added must be remembered. If the hierarchically superior ("more primary") security requirement is changed, then the secondary security goals may need changing.

Last, it is possible that no feasible way to satisfy a security requirement exists and no agreement can be reached on alternatives. In this case, one must return to the original business and quality goals of the application, modifying the initial conditions to change the assets implicated in the system or the security goals of the system. Alternatively, one might decide that it is infeasible to build the system.

4 Security Requirement Satisfaction Arguments

Our security requirements satisfaction arguments are in two parts: a *formal outer argument* that is first constructed based on the behavior specification for the system and *informal structured inner arguments* constructed to support the outer argument. The outer arguments, expressed in a formal logic, are built from claims about the world. These claims are assumptions that go beyond the basic provisos: that the context is correct and that the implementation does not introduce conflicting behavior. Inner arguments, expressed in structured natural language, support the assumptions made in the outer arguments.

4.1 The Outer Argument

The formal outer argument uses claims about system behavior (interplay of phenomena or behavior specification) to demonstrate that the security requirement (the constraint) is satisfied in a system that conforms to the behavior specification. It is expressed using a chosen logic, where the premises are formed from domain behavior properties and the conclusion is the satisfaction of the security requirement. Because the practitioners with whom we worked were more comfortable with propositional clauses (see Section 5), we use propositional logic in this paper. As a result, the outer argument is a proof that

$$\text{(domain behavior premises)} \vdash \text{(security requirement(s))}$$

where $\vdash$ is the syntactic turnstile. The expression $A \vdash B$ means B can be proved from A.

4.2 The Inner Arguments

The inner argument is a set of informal arguments that support the claims used in the outer argument. We propose a form of argument inspired by the work of Toulmin [77], one of the first to try to formalize a structure for arguments. Toulmin-style arguments are well suited for our purpose since they facilitate the capture of relationships between domain properties (grounds in the formal argument), the assumptions that eventually support these grounds, and reasons why the argument may not be valid.

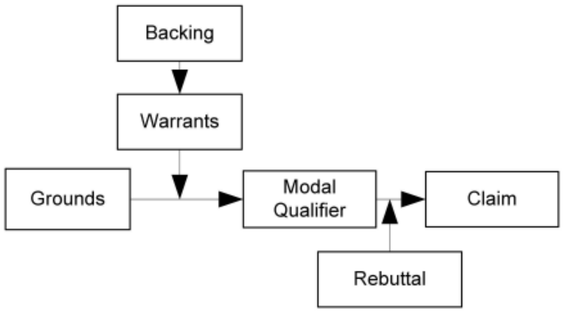


Fig. 3. Toulmin arguments.

Toulmin et al. [78] describe arguments as consisting of six parts, and propose a diagrammatic form for their arguments, shown in Fig. 3. The six parts are:

1. A *claim*: The end point of an argument, or what one wishes to convince the world of.
2. *Grounds*: Underlying support for an argument, e.g., evidence, facts, common knowledge, etc.
3. *Warrants*: Connecting and establishing relevancy between the grounds and the claim. A warrant explains how the grounds relate to the claim, but not the validity of the grounds.
4. *Backing*: Establishing that the warrants are themselves trustworthy. These are, in effect, grounds for believing the warrants.
5. *Modal qualifiers*: Indicating the reliability or strength of the connections from grounds and warrants to claims. Example: A cold front is passing (grounds). Cold fronts normally precede clearing weather (warrant). So, *chances are* (modal qualifier) the weather will clear (claim).
6. *Rebuttals*: Describing what might invalidate any of the grounds, warrants, or backing, thus invalidating the support for the claim.

Newman and Marshall show in [61] that the "pure" Toulmin form suffers because the fundamental recursive nature of the argument is obscured. Grounds, warrants, and backing may need to be argued, making them claims. We agree and extend Toulmin arguments to make explicit the recursive properties of arguments and the relationships between grounds, warrants, and claims, while keeping the basic connections between the components.

At some point, the argument recursion must stop. These "leaf" arguments will depend on unsupported grounds, which we call *trust assumptions* [32], [36]. Trust assumptions are statements about the behavior or properties of the world the system lives within, made in order to satisfy a security requirement and assumed to be true.

We use a simple language to represent the structure of these extended Toulmin arguments (see [33] for the formal grammar). This language captures the essence of Toulmin arguments while facilitating recursion and subarguments.

4.3 Satisfaction Arguments Example

A simple human resources application is used in this section to illustrate our uses of argumentation. Fig. 4 shows the initial problem diagram for this application. There are

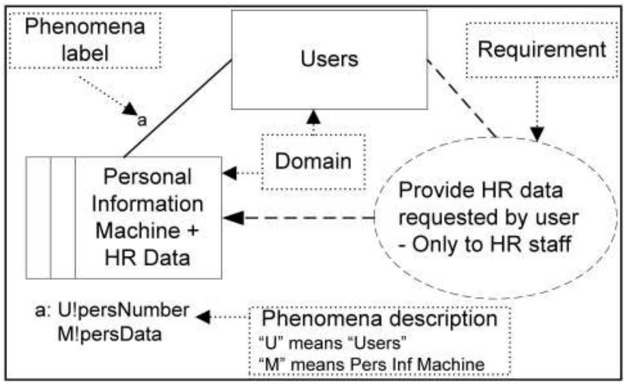

Phenomena label
Users
Requirement
a
Domain
Personal Information Machine + HR Data
Provide HR data requested by user - Only to HR staff
a: U!persNumber M!persData
Phenomena description
"U" means "Users"
"M" means Pers Inf Machine

Fig. 4. HR data retrieval problem.

two phenomena of interest, shown using the naming convention of "sending domain!message": the user's request for personnel information (U!persNumber) and the information returned by the request (M!persData). We assume one security goal: The data is to be kept confidential. One security requirement (constraint) has been derived from this goal: The data must only be provided to HR staff. An informal argument that this requirement satisfies the security goal is: Confidentiality of personnel data implies that people in general cannot be allowed access to this information, but HR staff can be relied upon to maintain its confidentiality. Therefore, a constraint that permits HR staff, but nobody else, to access it will satisfy the security goal. In terms of phenomena, the security requirement is M!persData must occur only when U!pers Number is input by a member of the HR staff.

4.3.1 The Outer Argument

Starting with the HR problem shown in Fig. 4, we first attempt to construct a proof that M!persData occurs only when U!persNumber is input by a member of the HR staff or, more formally, that M!persData $\vdash$ (User $\in$ HR).

There are two domains in the problem: the domain "users" and the machine (which contains the data). To construct the proof, we must first express the behavior of the system more formally. To do so, we use a notation based on the causal logic described in [56]. In this logic, the behavior of the domains in Fig. 4, expressed in terms of the phenomena, is:

U!persNum shall cause M!persData

A major problem is immediately exposed. Given what we see in the behavior description, there is no way to connect the system's behavior to the security requirement, as membership in the Users domain is not made apparent to the machine. No proof can be constructed. We need help from system architects, who can propose (at least) three design choices:

1. Introduce some physical restriction, such as a door guard, to ensure that the membership of "users" is restricted to HR staff. Doing so would permit construction of the following proof:

H is defined as User $\in$ HR (always true because of the physical restriction)
D is defined as phenomenon HR!persData
D $\rightarrow$ H (if info is displayed, then user $\in$ HR because H is always true)
D (info is displayed)
H (therefore user $\in$ HR)

2. Introduce phenomena into the system permitting authentication and authorization of a "user."
3. Introduce a trust assumption (TA) stating that we assert that the membership of "users" is limited to HR staff. Choosing this option would be dubious at best.

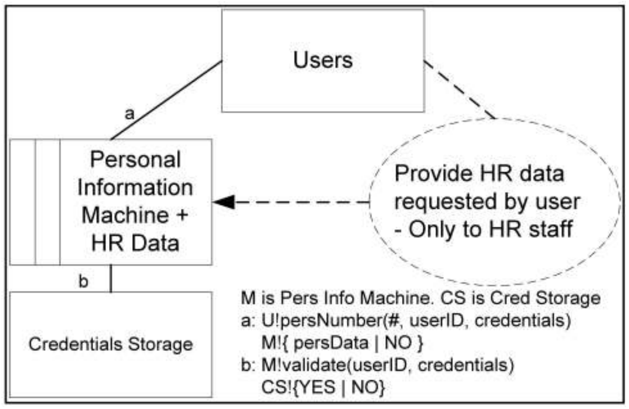

Users
a
Personal Information Machine + HR Data
Provide HR data requested by user - Only to HR staff
b
Credentials Storage
M is Pers Info Machine. CS is Cred Storage
a: U!persNumber(#, userID, credentials)
M!{ persData | NO }
b: M!validate(userID, credentials)
CS!{YES | NO}

Fig. 5. New HR staff problem diagram.

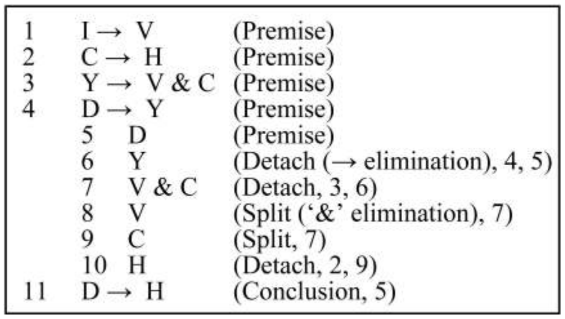
We choose option 2 and the resulting problem diagram is shown in Fig. 5. The user must supply some credentials along with the request for information. These credentials are passed to some external authentication and authorization engine, which answers yes or no. If the answer is yes, then the machine responds to the user with the data; otherwise, the data is refused. The new behavior specification is:

```
1. U!persNumber(#, credentials, userID) shall cause
   M!validate(UserID, credentials)
2. if isValid(userID, credentials)
       M!Validate(userID, credentials)
           shall cause CS!YES
   else
       M!validate(userID, credentials)
           shall cause CS!NO
3. CS!YES shall cause M!persData
4. CS!NO shall cause M!NO
```

The value returned by isValid() is determined by the contents of the Credentials Store.

We must now construct the satisfaction argument for the new "Users" domain. We begin with the outer argument, first defining the symbols to be used. These are shown in the following table.

Symbol	Derived from (see Figure 5)
I : Input request	U!(UserID, credentials, Payroll#)
V: Validate Creds	PIM!Validate(HR, UserID, credentials)
Y: ReplyYes	CS!YES
D: DisplayInfo	PIM!PersonInf(Payroll#)
C: CredsAreValid	isValid(UserID, credentials)
H: MemberOfHR	Conclusion: user is member of HR



```
1    I → V          (Premise)
2    C → H          (Premise)
3    Y → V & C      (Premise)
4    D → Y          (Premise)
       5   D        (Premise)
       6   Y        (Detach (→ elimination), 4, 5)
       7   V & C    (Detach, 3, 6)
       8   V        (Split ('&' elimination), 7)
       9   C        (Split, 7)
       10  H        (Detach, 2, 9)
11   D → H          (Conclusion, 5)
```

Fig. 6. Proof that the security requirement is satisfied.

The following propositional logic premises are derived from the behavioral specification. These premises are the grounds used in the formal argument and, if necessary, will be supported by informal arguments.

Name	Premise	Description
P1	$I \rightarrow V$	Input of request shall cause validation
P2	$C \rightarrow H$	If credentials are valid then user is a member of HR
P3	$Y \rightarrow V\&C$	A Yes happens only if credentials are valid and validated
P4	$D \rightarrow Y$	Display happens only if the answer was Yes

As the requirement is that we display information only to a member of HR, we include D as a premise and H as the conclusion. Thus, we want to show (P1, P2, P3, P4, D $\vdash$ H). A proof is shown in Fig. 6.

4.3.2 The Inner Arguments

Each of the rules used in the outer argument should be examined critically. We begin with the premises P1, P3, and P4. These are probably not controversial because one can say that they are part of the specification of the system to be implemented. The arguments thus consist of one trust assumption, as shown in the following utterance in our argument language:

let G1 = "system is correctly implemented";
given grounds G1 thus claim P1.
given grounds G1 thus claim P3.
given grounds G1 thus claim P4.

Premise P2 is more complex. It is making a claim about the behavior membership of the domain "Users" by saying that if a person has valid credentials, then that person must be a member of HR. An argument for this claim is shown in Fig. 7. This argument incorporates three trust assumptions: G2, G3, and G4.

The three rebuttals in the argument require some consideration. Remember that rebuttals express conditions under which the argument does not hold. If the rebuttals remain in the argument, they create implicit trust assumptions saying that the conditions expressed in the rebuttals will not occur, which may be acceptable. Alternatively, one could construct an argument against a rebuttal. We will do that for R1 in the next section.

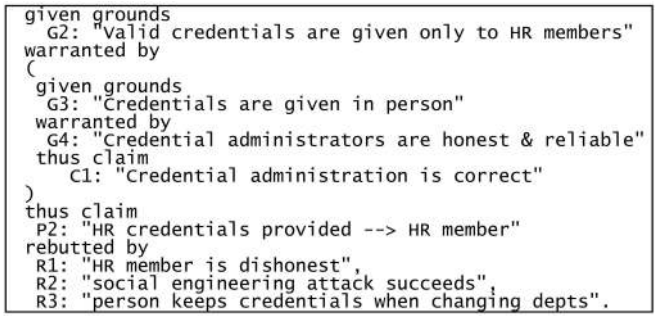
4.4 Removing Rebuttals by Adding Function

Just as one might be required to modify the problem in order to be able to construct the outer argument, at times the most straightforward way to remove a rebuttal might be to add functionality to a system. The additional functionality would permit adding new grounds or warrants to mitigate the conditions that permit the rebuttal.

As an example, consider R1: HR member is dishonest. One could mitigate this risk by increasing the probability that an unusual use of the employee's credentials would be detected, thus raising the probability that the misuse would be detected. To this end, we add two functional requirements to the system:

- All uses of HR credentials are logged
- Any use of HR credentials from a location outside the HR department is immediately signaled by e-mail to the HR director.

The functional requirements would be used as grounds in an argument against the rebuttal R1, shown in Fig. 8. C2 is next added as a mitigating proposition to the rebuttal in argument 1 (R1: "HR member is dishonest" mitigated by C2). Note that C2 might also mitigate R2 (a successful social engineering attack) by revealing unauthorized uses of credentials.



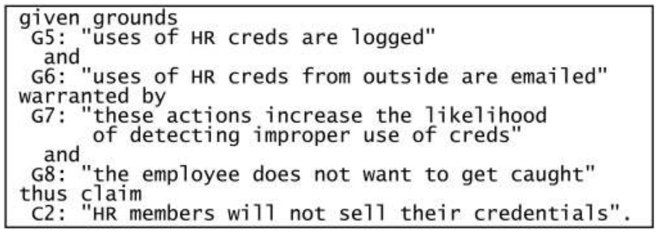
```
given grounds
  G2: "Valid credentials are given only to HR members"
warranted by
(
 given grounds
  G3: "Credentials are given in person"
 warranted by
  G4: "Credential administrators are honest & reliable"
 thus claim
    C1: "Credential administration is correct"
)
thus claim
 P2: "HR credentials provided --> HR member"
rebutted by
 R1: "HR member is dishonest",
 R2: "social engineering attack succeeds",
 R3: "person keeps credentials when changing depts".
```

Fig. 7. Argument 1: for premise P2.

5 Application of the Framework

We applied our framework in the "CRISTAL UK" project [81], a research initiative managed by NATS (formerly National Air Traffic Services) for the EUROCONTROL CASCADE Program. This air traffic control project is charged with "determining the role of 'passive surveillance' in NATS future surveillance system[s]" [81] where radar is used currently, such as in and around the airspace at busy airports.

Air Traffic Control is responsible for the safe and efficient movement of aircraft through a given airspace. Air traffic controllers use various surveillance technologies to try to keep the risk low by maintaining safe horizontal and vertical distances (separation) between aircraft. Active technologies do not require cooperation of the aircraft.



```
given grounds
 G5: "uses of HR creds are logged"
  and
 G6: "uses of HR creds from outside are emailed"
warranted by
 G7: "these actions increase the likelihood
      of detecting improper use of creds"
  and
 G8: "the employee does not want to get caught"
thus claim
 C2: "HR members will not sell their credentials".
```

Fig. 8. Argument against rebuttal R1.

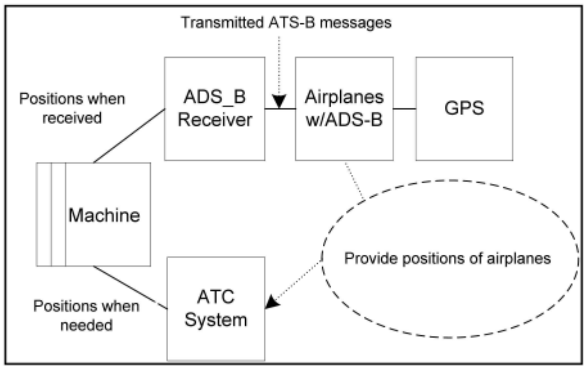


Fig. 9. System context—iteration one.

Passive technologies require the aircraft to broadcast information actively. The passive surveillance method under consideration by the CRISTAL project is ADS-B (Automatic Dependent Surveillance-Broadcast), where satellite navigation technology on board the aircraft determines where the aircraft is in three dimensions, then broadcasts that position. For more background information, see Appendix A.

5.1 The Security Requirements Analysis

The project asks whether ADS-B position reports can (or should) be considered to be a primary position source. We analyze the security implications of this position using our framework by stepping through the activities in Fig. 2. The sections below are numbered using *iteration.stage*, where "stage" comes from Fig. 2. For example, the second stage of the first iteration will be numbered 1.2.

5.2 The First Iteration

During this first iteration, we establish the context for the system, the functional requirements, and the primary security goals and requirements.

Step 1.1—Identify Functional Requirements. In this stage of the activity diagram, we identify the functional goal(s) of the system under analysis, describe the context, and identify the functional requirement(s). This task is dramatically simplified because working ADS-B equipment is being supplied by project partners and the initial functional goal was given. That functional goal was:

FG1: Provide safe and efficient air traffic management.

Given the above goal and project's remit, the functional requirement can be summarized by:

FR1: Provide positions of aircraft.

The only task remaining is to determine the context, which is shown in Fig. 9.

Step 1.2—Identify Security Goals. This step is charged with determining the assets involved directly or indirectly with the system, the harm that the assets can suffer, and, finally, the security goals to avoid those harms.

The direct assets found from the context are GPS receivers and signals, aircraft, positions of the aircraft (broadcast), ground receivers, and the ATC system (including people). The indirect assets are the aircrafts' contents (passengers, etc.), items in the ATC area (e.g., infrastructure, buildings, possibly the airport), and the aircraft owner's business (reputation, profitability, etc.).

Using this list of assets, we can (with the help of the project's domain experts) determine the potential harm and then the *threat descriptions* [31], expressed as *violation of general security goal* on *asset* can cause *harm*. The threat descriptions are:

- General goal: confidentiality:

T1: {publicizing, airplanes' position, facilitating attack in air}
T2: {publicizing, airplanes' position, lost of trade secrets}

The stakeholders made the decision that threats T1 and T2 are outside of the project's remit.

- General goal: integrity:

T3: {~ correct, airplanes' position, lost property due to collision or crash}
T4: {~ correct, airplanes' position, lost revenue due to increased separation}
T5: {~ correct, airplanes' position, lost revenue due to lost confidence}

- General goal: availability:

T6: {~ available, airplanes' position, lost property due to collision/crash}
T7: {~ available, airplanes' position, lost revenue due to increased separation}
T8: {~ available, airplanes' position, lost revenue due to lost confidence}

The security goals are determined by *avoiding* the action in the threat descriptions. Given these threat descriptions, the security goals are:

SG1: Have correct positions (avoids T3, T4, and T5)
SG2: Report positions on a timely basis (avoids T6, T7, T8)

Step 1.3—Identify Security Requirements. In this step, we determine the constraints to place on the functional requirement FR1: Provide positions of airplanes. We do this by composing the security goals and the functional requirement, resulting in a constrained functional requirement.

The composition produces two security requirements (constraints). The first is:

SR1 [FR1: Provide positions of aircraft]: positions shall be accurate.

The NATS requirement for accuracy is that the aircraft be within 300 meters of its reported position at the time the position is received. However, ADS-B can potentially improve on that by an order of magnitude and the consequences of this must be studied. SR1 operationalizes SG1.

The second constraint is:

SR2 [FR1: Provide positions of airplanes]: positions shall be timely.

The NATS requirement for timeliness is that a new position be received within 4 to 6 seconds of the last position report,

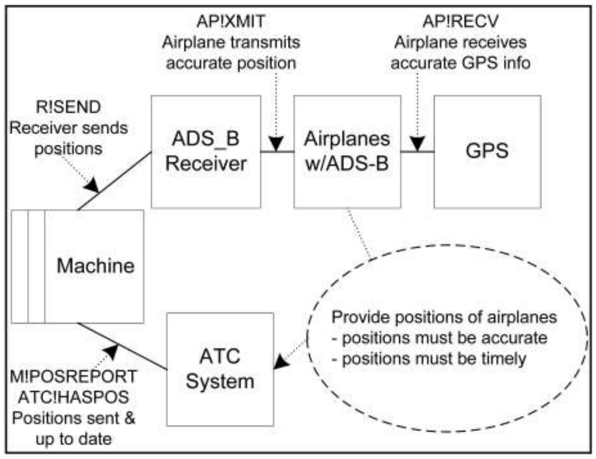

AP!XMIT
Airplane transmits accurate position
AP!RECV
Airplane receives accurate GPS info
R!SEND
Receiver sends positions
ADS_B Receiver
Airplanes w/ADS-B
GPS
Machine
ATC System
Provide positions of airplanes
- positions must be accurate
- positions must be timely
M!POSREPORT
ATC!HASPOS
Positions sent & up to date

Fig. 10. Context with constrained requirement.

or of the aircraft entering controlled airspace. SR2 operationalizes SG2. Fig. 10 shows the context with the constraints.

Step 1.4—Satisfaction Arguments. We begin by constructing the formal outer argument. The steps are 1) annotate the context with the phenomena exchanged between domains, 2) develop a behavioral specification for the system in terms of the phenomena, and then 3) use the phenomena and behavioral specification in a proof that, if they are complete, the system can satisfy the security requirements.

The Phenomena. Fig. 10 shows the phenomena exchanged within the system and used in the behavior specification. The phenomena are:

AP!RECV: The airplane receives GPS broadcasts.
AP!XMIT: The airplane transmits its position.
R!SEND: The receiver sends the received position to the machine.
M!POSREPORT: The machine sends the position to the ATC system.
ATC!HASPOS: The ATC confirms that it has the aircraft's position.

The Behavior Specification. The behavioral specification is built using a variant of the causal logic described in [56] and used in Section 4. For this project's ATC system, the behavioral specification is:

AP!RECV shall cause AP!XMIT
AP!XMIT shall cause R!SEND
R!SEND shall cause M!POSREPORT
M!POSREPORT shall cause ATC!HASPOS

We recognize that reception of GPS signals by the aircraft will not actually cause the aircraft to transmit position reports, but instead enables them. We chose to accept this slight misstatement instead of adding a clock to the context and changing to a temporal logic. As a consequence, AP!RECV shall cause AP!XMIT embeds the assumption that it repeats often enough to satisfy the NATS requirement. We also assume that each processing step in the system will complete in an appropriate amount of time, again to avoid changing to a temporal logic.

```
1. AP!RECV → AP!XMIT     (premise)
2. AP!XMIT → R!SEND      (premise)
3. R!SEND → M!POSREPORT  (premise)
4. M!POSREPORT → ATC!HASPOS (premise)
5. AP!RECV               (assumption)
6.   AP!XMIT             (Detach, 1, 5)
7.   R!SEND              (Detach, 2, 6)
8.   M!POSREPORT         (Detach, 3, 7)
9. ATC!HASPOS            (Detach, 4, 8)
```

Fig. 11. The outer argument (proof).

The Outer (Formal) Argument. There is now enough information to construct the outer argument, a proof that the system can respect the security requirements. We want to prove that

AP!RECV $\vdash$ ATC!HASPOS

If we can prove this, then we have proved that the system can satisfy both SR1 (accuracy) and SR2 (timeliness), given the following assumptions: 1) The context is correct and the implementation introduces no conflicting behavior (the provisos found in Section 3.5.4), and 2) the temporal properties assumed above are not significant. Some of these assumptions will be challenged when we build the inner arguments. A proof is shown in Fig. 11.

The Inner Arguments. The premises and assumptions of the outer argument comprise a set of assumptions that must hold for the system to be secure. The purpose of the inner arguments is to challenge these assumptions in order to establish whether they hold in the real world. In our case, Steps 1 through 5 in Fig. 11 are the assumptions to be challenged.

As explained in Section 4.2, we chose to represent arguments in our framework in a text form because this form handles complex grounds-to-claim graphs and recursion in the arguments more naturally. The argument for the initial premise AP!RECV $\rightarrow$ AP!XMIT in this form is:

Given grounds
  Received GPS positions are accurate (AP!RECV & assumptions)
Warranted by
  Calculations are accurate (assumption)
Thus claim
  Airplanes transmit accurate positions (AP!XMIT)
[Rebutted by . . .]

One of our first lessons learned was that, although it is easy to understand the text representation of an argument when the argument is simple, understanding by project members becomes more difficult as the arguments become more complex. As such, we changed to a modified form of the argument diagrams Toulmin proposed. Fig. 12 shows the argument in this form, along with the newly added rebuttals. The text in parentheses (e.g., SR2) is the security requirement that is violated if the rebuttal is true. Figs. 13, 14, 15, and 16 show the arguments for premises 2 through 4 (numbers of the lines in the proof) and for the assumption (line 5).

There are 12 rebuttals in the arguments. The rebuttals fall into three general categories: sabotage where equipment is sabotaged to break it (R1.1, R1.2, R1.4, and R1.6 through

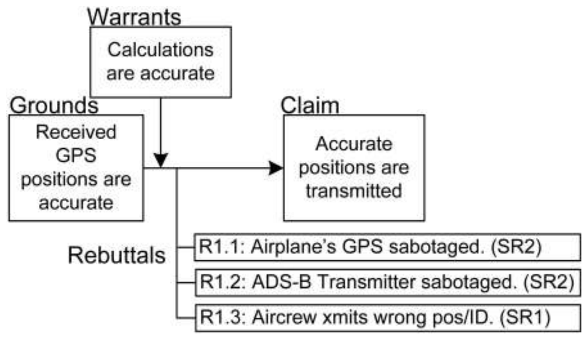

Warrants
Calculations are accurate
Grounds
Received GPS positions are accurate
Claim
Accurate positions are transmitted
Rebuttals
R1.1: Airplane's GPS sabotaged. (SR2)
R1.2: ADS-B Transmitter sabotaged. (SR2)
R1.3: Aircrew xmits wrong pos/ID. (SR1)

Fig. 12. Argument for AP!RECV → AP!XMIT.

R1.11), externally caused denial of service (R1.5 and R1.12), and the intentional transmission of incorrect data (R1.3). Each of these rebuttals should be evaluated to determine whether it should be mitigated and, if so, how. If a rebuttal is to be mitigated, then iteration is required. The project assumed that R1.3 presented an unacceptable risk of terrorism; aircraft believed to be following some track X but really going somewhere else could do a great deal of damage.

Note that rebuttals that are safety concerns are not considered here. For example, the equivalent of "jamming" can be caused by natural phenomena such as multipath and electrical interference. We consider these to be "normal" behavior and therefore to be considered during a safety analysis.

5.3 The Second Iteration

In order to mitigate R1.3, we must find a way to know that the position an aircraft transmits is the true position of the aircraft. We are less concerned with detecting that an aircraft transmitting a correct position is using the wrong identity.

Multilateration can be used to determine the position of a transmitter, computing the position by measuring the difference in a transmission's arrival time at multiple receivers. We choose this approach and change the context appropriately. The new context is shown in Fig. 17.

Stepping through the framework, we see that we do not have any new functional requirements (we put aside administration of the multilateration system). We do have new assets, the multilateration computers, but they did not add any new security goals. As such, our security requirements did not change.

The behavior specification does have a significant change. We must describe the behavior of the new multilateration component in the context. The behavior specification is now:

```
AP!RECV shall cause AP!XMIT
AP!XMIT shall cause R!SEND
R!SEND shall cause MC!SEND
MC!SEND shall cause M!POSREPORT
M!POSREPORT shall cause ATC!HASPOS
```

We now have a new premise in our proof, corresponding to the new and altered components of the behavior specification (marked by the box).

We learned another lesson at this point. It was easier to describe the effects of the iteration using a graphical

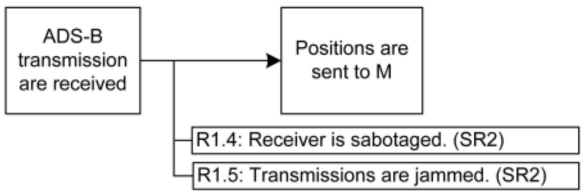

ADS-B transmission are received
Positions are sent to M
R1.4: Receiver is sabotaged. (SR2)
R1.5: Transmissions are jammed. (SR2)

Fig. 13. Argument for AP!XMIT → R!SEND.

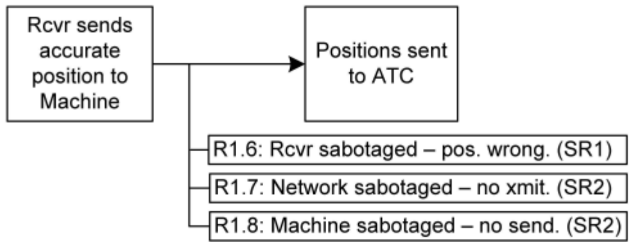

Rcvr sends accurate position to Machine
Positions sent to ATC
R1.6: Rcvr sabotaged – pos. wrong. (SR1)
R1.7: Network sabotaged – no xmit. (SR2)
R1.8: Machine sabotaged – no send. (SR2)

Fig. 14. Argument for R!SEND → M!POSREPORT.

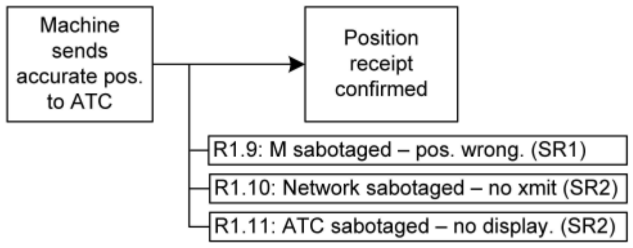

Machine sends accurate pos. to ATC
Position receipt confirmed
R1.9: M sabotaged – pos. wrong. (SR1)
R1.10: Network sabotaged – no xmit (SR2)
R1.11: ATC sabotaged – no display. (SR2)

Fig. 15. Argument for M!POSREPORT → ATC!HASPOS.

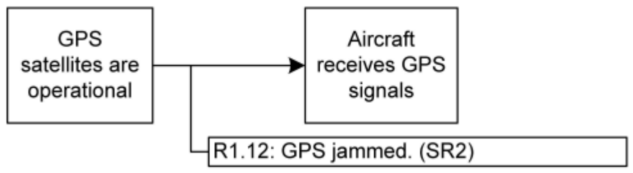


Fig. 16. Argument for AP!RECV.

"subargument" technique, rather than expressing the arguments again. This technique applies the mitigation (M2.1) directly to the rebuttal developed during the first iteration. We use that technique here. Fig. 18 shows the resulting argument and mitigation. The figure also shows the rebuttals of the mitigation, described in the next paragraph.

The first rebuttal challenges the assumption that the transmitter is actually in the airplane it says it is in, or is even in an airplane. One could have a small airplane accompanying a large one. The small airplane broadcasts the position, which would permit the large airplane to divert. Alternatively, one could have a series of transmitters in cars pretending to be the airplane. The second rebuttal challenges the assumption that there is a transmitter where multilateration says it is. It is possible to use multiple transmitters and vary the timing to create "virtual transmitters" at any position [10]. The third rebuttal challenges the assumption that the clocks in the receivers are synchronized. It is possible to perturb the clock at particular receivers, which would cause the position calculation to be offset. More detail on this rebuttal would require looking at specific multilateration time-synchronization solutions.

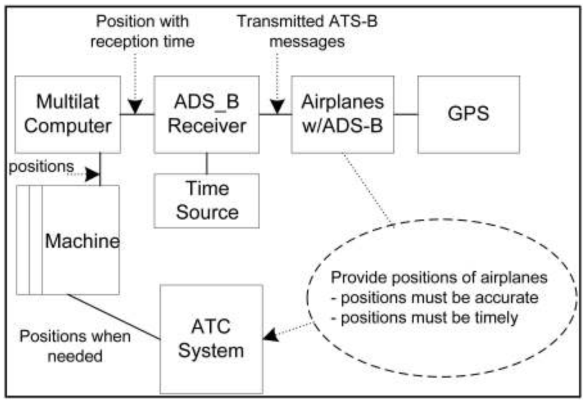

Position with reception time
Transmitted ATS-B messages
Multilat Computer
ADS_B Receiver
Airplanes w/ADS-B
GPS
positions
Time Source
Machine
Provide positions of airplanes
- positions must be accurate
- positions must be timely
Positions when needed
ATC System

Fig. 17. Context diagram, iteration two.

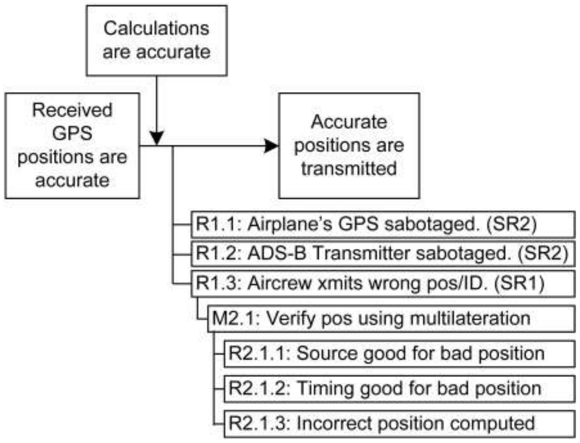

Calculations are accurate
Received GPS positions are accurate
Accurate positions are transmitted
R1.1: Airplane's GPS sabotaged. (SR2)
R1.2: ADS-B Transmitter sabotaged. (SR2)
R1.3: Aircrew xmits wrong pos/ID. (SR1)
M2.1: Verify pos using multilateration
R2.1.1: Source good for bad position
R2.1.2: Timing good for bad position
R2.1.3: Incorrect position computed

Fig. 18. Arguments for the second iteration.

5.4 The Third Iteration

A third iteration is required to deal with rebuttals R2.1.*, assuming that the risks are determined significant, which, at first glance, they appear to be. For example, primary radar mitigates all of them because it gives a reliable indication that something really is at the position reported and that there is not something elsewhere. R2.1.2 could possibly be mitigated by using antennae that provide an approximation of the "angle off of horizontal" of a transmission. R2.1.3 can be mitigated by use of a secure clock synchronization technology.

6 Discussion

This section presents several issues arising from the application of our framework.

6.1 The Logic Used for the Outer Argument

We used propositional logic in our example for simplicity and because our partners were comfortable with propositional clauses. As a side effect, in the example in Section 4 we hid implicit assumptions that ought to be explicit, e.g., the UserID is the same in I and V. For that example to be complete, claims (trust assumptions) should have been added to the inner argument to cover these assumptions. Using predicate logic would have removed this difficulty. Using a temporal logic would also have removed some difficulties described in Section 5.2. We theorize that use of a more fine-grained logic in the outer argument may lead to fewer trust assumptions in the inner argument. On the other hand, more expressive logics come at the expense of tractability of reasoning and of potential decidability problems.

One problem we encountered during the CRISTAL project concerned the outer argument. The outer argument proves that *if the assumptions are valid* and *if the behavior specification is correct*, and *if there are no other behaviors*, then *the system can be secure*. It does not prove that a system *will be* secure and, in fact, cannot do so. Given this distinction, some people did not see the utility of outer arguments and wanted to proceed directly to the inner arguments. However, it is the outer argument that provides the assumptions to be tested in the inner arguments, so skipping this step is not appropriate. We need to find a better way to motivate, capture, and represent the outer arguments.

Note that motivating the need to test the assumptions flowing from the outer arguments (the premises) did not present a problem. People seemed to enjoy constructing the inner arguments.

6.2 The Inner Argument Language

The project members were happier using a graphical representation of the inner arguments, even though the representation has less expressive power than text representation. This, plus the desire to bypass the outer arguments, led to us using the *rebuttal, mitigation* graphical argument form. The graphical form is less expressive because there are arguments that are not easy to express in this form, such as when a mitigation requires a warrant or covers several rebuttals. Tool support for converting between the text and graphical forms and for "drawing" summary arguments would be very helpful.

6.3 Constructing Inner Arguments

One question that arises is "how does the analyst find rebuttals, grounds, and warrants?" We cannot propose a recipe, but did find useful a method inspired by the how/why questions used in goal-oriented requirements engineering methods (e.g., KAOS [47]). Given a claim, the analyst asks "why is this claim true?" and "what happens if it is not true?" The analyst first chooses which claim to argue and then use the "why" question to gather the grounds that are pertinent to the claim along with the warrants that connect the grounds to the claim.

It is interesting to note that, although domain knowledge is certainly required, our project experience showed that domain expertise could lead people not to question assumptions because the experts assumed that something behaves in manner X because that is how it has always done. We found that having domain nonexperts in a project helped; it seemed that someone from outside was more likely to ask "why is that?" at odd times. Note that once the questions were asked, we had no problem having lively and productive discussions.

6.4 Iteration

The project confirmed that the iteration provided by the framework is required, especially when considering mitigations. However, iteration requires careful management to ensure that interactions are detected. The choice to use the graphical notation, representing mitigations in the context of their rebuttals, led to considering them one at a time when they should be considered together as part of a complete analysis. For example, it makes sense to consider all of the jamming scenarios together (e.g., ADS-B jamming, clock sync jamming, GPS jamming) instead of considering them independently.

6.5 Problem versus Solution Space

A reasonable objection to our framework is that we are designing the system in order to determine its requirements. To some extent, this is true; the domains included in the system are being more finely described iteratively. However, we argue that the part of the system being constructed is the *machine* and we are not designing that. By applying an iterative process that interleaves requirements and design [63], we are specifying the environment (or context) that the machine lives within. These specifications include additional domains that need to exist (perhaps inside the machine) and additional phenomena required to make use of these domains.

We also found that security problems expand the system context in unexpected ways. For example, cars and roads are (usually) not considered part of an ATC problem until considering whether they can be used to move transmitters that create virtual positions for aircraft. Neither are GPS satellite signals until GPS jammers are considered. The challenge we faced was to expand the context as much as necessary, but no more so.

7 Related Work

In this section, we look at related work on how security requirements are defined and represented, contrasting the work to our own. In addition, we discuss related research on safety requirements analysis, and on design rationale and argument capture.

7.1 Security Requirements as Security Functions

It is common to express security requirements by describing the security mechanisms to be used. For example, ISO 15408 [38], [39], [40], the ISO version of the Common Criteria [16], [17], [18], provides examples of security requirements of the general form "The [...] Security Function (TSF) shall explicitly deny access of subjects to objects based on the [rules ...]" [39, p. 48], where "rules" appear to be a mechanism. Regarding encryption, one finds "The TSF shall distribute cryptographic keys in accordance with a [specified cryptographic key distribution method] that meets the following: [list of standards]" [39, p. 39]. Again, a mechanism is being described. In addition, both examples say what the function is to do, not why it is to do it.

The NIST Computer Security Handbook states that "These [security] requirements can be expressed as technical features (e.g., access controls), assurances (e.g., background checks for system developers), or operational practices (e.g., awareness and training)" [62, p. 80], in effect defining security requirements in terms of functions and practices. Other security guides imply that recommendations such as "Acquire Firewall Hardware and Software" (e.g., [2]) are requirements.

Defining requirements in terms of function leaves out key information: *what* objects need protecting and, more importantly, *why* the objects need protecting. Although both the ISO and NIST documents say that the underlying reasons why objects are to be protected come from the functionality of the system, they provide little guidance on how to connect the functionality to the security needs. Instead of describing when and why objects are to be protected, they describe how the objects are to be protected. Our framework provides the when and why, leaving the how to the designers.

7.2 Security Requirements as NFRs

Kotonya and Sommerville [46], when discussing NFRs, in which they include security, define them as "restrictions or constraints" on system services; similar definitions can be found in other text books. Rushby [68] appears to take a similar view, stating "security requirements mostly concern what must not happen". Using the Tropos methodology, Mouratidis et al. [58] state that "security constraints define the system's security requirements."

Firesmith, in [26], [27], defines security requirements as "a quality requirement that specifies a required amount of security [...] in terms of a system-specific criterion and a minimum level [...] that is necessary to meet one or more security policies." This appears to be a form of constraint, an impression reinforced by an example he provides: "The [application] shall protect the buyer-related data [...] it transmits from corruption [...] due to unsophisticated attack [when] [...] Buyer Buys Item at Direct Sale [to a level of] 99.99 percent."

The problem with these definitions is their lack of specificity and guidance for the designers. What "system services" are being constrained? What is the constraint and what effect will it have on the functionality of the system? How does one validate some chosen constraint within the system context to ensure that it accurately reflects the stakeholders' wishes? Referring to Firesmith's example, what is an "unsophisticated attack?" What does the measure "99.99 percent" mean? Our framework avoids these problems by expressing security requirements as specific constraints on specific functions in the system.

7.3 Security Requirements from Privacy and Trust

Some researchers approach security from a privacy point of view, asserting that if information "owned" by an agent is kept private, then security goals will be met. De Landtsheer and van Lamsweerde propose modeling which properties agents, authorized or not, can know [22]. The Tropos project, e.g., [29], takes a similar view, but extended to include agents' intentions and explicit trust delegation. Breaux et al. [6] extract privacy rights and obligation information from "policy documents" to assist with development of security requirements.

These approaches work well for problems dominated by privacy concerns. They are less effective when considering vulnerabilities in a system context, primarily because they are focused on early requirements [44] and do not develop a model of the real-world system context. They are also less effective in applications where privacy (c.f. confidentiality) is not the dominate concern. The example in Section 5 is one such case. Air traffic control is dominated by integrity and availability concerns; high confidence is needed that airplanes are where they say they are. We believe these privacy-focused approaches to be complementary to our approach and that they can be integrated into our framework at stages 1 and 2. Integrating at stage 1 would inform development of functional requirements. Integration at stage 2 would generate privacy-focused security goals that would be taken into account in the later stages and during iteration.

7.4 Other Portrayals of Security Requirements

Many authors assume that security requirements are identical to high-level security goals. Tettero et al. [75] are explicit about this, defining security requirements as the confidentiality, integrity, and availability of the entity for which protection is needed. While this is a clear definition, in some cases it may not result in precise enough requirements. Consider an example in health care: Both doctors and administrators would probably agree on the importance of confidentiality, integrity, and availability of the clinical information, but they could disagree on the concrete security requirements that express those goals. The requirements need to be more explicit about *who* can do *what when*. Our framework provides this explicitness.

Some authors identify security requirements with security policies. Devanbu and Stubblebine [23] define a security requirement as "a manifestation of a high-level organizational policy into the detailed requirements of a specific system. [... We] loosely (ab)use the term 'security policy' [...] to refer to both 'policy' and 'requirement'." Anderson [3] is less direct; he states that a security policy is "a document that expresses [...] what [...] protection mechanisms are to achieve" and that "the process of developing a security policy [...] is the process of requirements engineering." Redwine [67] reports that the "software system security policy is part of software system requirements placing constraints on system behavior." The difficulty with "security policies" is its chameleon-like meaning. The term can be used for anything from a high-level aspiration to an implementation. Therefore, without accompanying detailed explanation, it is not satisfactory to define security requirements as security policies.

Lee et al. [51] point out the importance of considering security requirements in the development life cycle, but do not define them. Heitmeyer [37] shows how the Software Cost Reduction method can be used to specify and analyze security properties, without giving the criteria for distinguishing them from other system properties.

A number of papers have focused on security requirements by describing how they may be violated. For example, McDermott and Fox [54], followed independently by Sindre and Opdahl [70] and elaborated upon by Alexander [1], describe abuse and misuse cases, extending the use case paradigm to undesired behavior. Liu et al. [53] describe a method of analyzing 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. One could argue that Chivers and Fletcher [14] fall into this camp with SeDAn, as they focus on attackers and the paths they might take into a system. The difficulty with these approaches is that they indicate what a system is not to permit in specific situations, but not in the general case.

Van Lamsweerde [48] describes a process by which security goals are made precise and refined until reaching security requirements. Antón and Earp [4] use the GBRAM method to operationalize security goals for the generation of security policies and requirements, but do not define security requirements. Like the privacy-based approaches discussed earlier, both of these can be integrated into our framework at stages 1 and 2, informing the construction of the system context and the generation of security requirements (constraints) within that context.

Mead et al. in the SQUARE methodology [55] describe security requirements as being at the system level or the software level. They do not define what requirements are, beyond saying that "Requirements are concerned with what the system should do." They also introduce the notion of "architectural constraints" that specify "how it should be done," leaving open how one distinguishes between a constraint that a system use an existing authentication system and a requirement that the system support authentication in a given context. Our framework fits well within SQUARE, providing a pathway from goals to requirements and making the requirements (or constraints) implied by the context clear.

7.5 Safety and Security

There is a very close relationship between security and safety requirements. Both deal with system failures that lead to harm. Both deal with analysis of the context to look for evidence about how failures might occur. Both add requirements to reduce the possibility of, or to mitigate the effects of, these failures. However, they are not identical. We and other authors consider one difference between security and safety to be *intention* (e.g., [26], [43], [52]). Safety concerns harm caused by accident, while security concerns harm caused by an attacker. Security failures can lead to safety concerns; consider placing a bomb on an airliner. Equally, safety failures can lead to security concerns; consider an accident involving a truck carrying unencrypted backup tapes.

The use of intention as a discriminator is not universally agreed. For example, [5] defines security as "the absence of unauthorized access to, or handling of, system state." The paper discusses the role of intention, but does not give it any particular emphasis. The SafSec methodology [49] combines safety and security, without introducing intention. The ITSEC defines security as "the combination of confidentiality, integrity and availability [of information]" [69, p. 115], a view of security that does not include intention.

Although we recognize that using intention as the differentiator between safety and security is sometimes uncomfortable, we feel that the distinction being made between intention and accident is helpful. It assists with setting bounds on both the context and the mitigations. Consider the possibility of failure of some component in an aircraft, potentially causing the aircraft to crash. Under our definition, this is a safety problem and therefore not considered in our analysis. However, if the component could be provoked to fail, then we have a security problem: preventing the (intentional) actions that could provoke the failure.

7.6 Design Rationale and Argument Capture

Design rationale is principally concerned with capturing how one arrived at a decision, alternate decisions, or the parameters that went into making the decision [50]. For example, Buckingham Shum [8] focuses on how rationale (argument) is visualized, especially in collaborative environments. Potts and Bruns [65] and, later, Burge and Brown [9] discuss capturing how decisions were made, which decisions were rejected, and the reasons behind these actions. Mylopoulos et al. [59] present a way to formally represent knowledge that was captured in some way, without focusing on the outcome of any decisions. Ramesh and Dhar [66] describe a system for "capturing history in the upstream part of the life cycle." Fischer et al. [28] suggest that the explicit process of argumentation can itself feed into and benefit design. Finkelstein and Fuks [25] suggest that the development of specifications by multiple stakeholders who hold disparate views may be achieved through an explicit dialogue that captures speech acts, such as assertions, questions, denials, challenges, etc. The representation of the dialogue is then a rationale for the specifications constructed. The common element in all of the above work is the capture over time of the thoughts and reasons behind decisions. Whether the decisions satisfy the needs is not the primary question.

Our goal for our security satisfaction arguments was to *convince a reader* that the security requirements can be satisfied and that nothing was omitted that could result in the requirements not being satisfied. The goal was not to capture how a decision was made. Our position is not unique. Kelly took a similar position with regard to safety arguments [45]. *Argumentation goals* serve a similar function in the NFR Framework, supporting (or not) refinements of goals [60]. Of course, we are not saying that there is no use in having a trace of what led to the final arguments; the history will certainly be useful if the arguments fail to convince or if the situation changes.

8 Conclusions

This paper has presented a framework for security requirements engineering where

1. asset and security goal analysis are done in the business context of the system,
2. the effects of security requirements on the functional requirements are understood,
3. design constraints are taken into account, and
4. the correctness of security requirements is established through the use of satisfaction arguments.

As noted in the introduction, there are three contributions in this paper. The first is *definition*: a coherent definition of what security requirements are. The second is explicit recognition of the importance of *context*: the world within which the system and the potential attackers exist. The third is a structure for *satisfaction arguments* for validating whether the system can satisfy the security requirements. We have shown how satisfaction arguments facilitate showing that a system can meet its security requirements. The structure behind the arguments assists in finding system-level vulnerabilities. By first requiring the construction of the formal argument based on domain properties, one discovers which domain properties are critical for security. Constructing the informal argument showing that these domain properties can be trusted helps point the analyst toward vulnerabilities; the rebuttal is an important part of this process. Vulnerabilities found in this way are removed through either modification of the problem, addition of security functional requirements, or addition of trust assumptions that explain why the vulnerability can be discounted.

Recalling our experience with NATS, we had two goals for the project: to gain experience with the application of our framework to validate its utility and to discover security requirements in our chosen problem domain. As we used the framework to produce security requirements, rebuttals, and mitigations that had not previously been considered, we consider that we succeeded with both goals.

One area that we are actively looking at is tool support for capturing the arguments. One approach we are considering is "compiling" the abstract syntax tree built by the parser, decorating the tree with appropriate semantic information and symbol table references. We are also looking at a tool constructed around problem context diagrams by experimenting with adapting the argument capture tool Compendium [19] for describing and navigating through IBIS-style arguments [20].

Another area for future work is risk analysis and partial satisfaction. Our plan is to add multivalued indications of risk and satisfaction to security requirements (via the inner arguments). After defining an arithmetic for comparing and combining risk and satisfaction values, we would compute an overall risk and satisfaction level for the system.

It seems that there might be a close correspondence between the "defense in depth" principle and completing different outer arguments that depend on different domain properties. We wish to investigate this idea in more detail.

Two particular future work items deserve mention. Our work with NATS showed the need for more tool support for representing outer arguments and we are adding this task to our near-term future work list. The effort also showed the need for work on better enabling construction and understanding of the outer (formal) arguments by people who do not normally use formality, which is a longer-term research question.

Appendix A

Active and Passive Air Traffic Control Surveillance

Air Traffic Control is responsible for the safe and efficient movement of aircraft through a given airspace. Unfortunately, "safe" and "efficient" are at odds with each other. An empty airspace is a safe one—no loss of life or property due to problems with aircraft is possible—but it is also a very inefficient one. One increases efficiency by adding aircraft into the airspace, which increases risk that an accident (or an intentional act leading to loss) will occur. Air traffic controllers try to keep the risk low by maintaining safe horizontal and vertical distances (separation) between aircraft. To do so, air traffic controllers must know the identity and position of aircraft with a high degree of accuracy, integrity, and assurance.

A.1 Separation

The most important job of an air traffic controller is to maintain a safe separation between aircraft while ensuring that the aircraft get to where they want to go. The minimum separation between aircraft at a given time is dependent on many factors, including speed of aircraft, surveillance accuracy, ability to communicate with aircraft and between controllers, redundancy of surveillance systems, and the ability to spot and rectify mistakes.

Most of the factors are strongly influenced by how often the controller is told where an aircraft actually is, as opposed to where it is supposed to be. The more often accurate positions are reported, the more accurate the controller's "picture" of the airspace is. The controller determines aircrafts' positions using active and passive surveillance.

A.2 Active versus Passive Surveillance

Active surveillance describes a process to determine the position of aircraft independently of where the aircraft thinks it is. There are two systems in use: primary radar and secondary radar. Primary radar operates by broadcasting directional pulses and listening for pulses reflected off aircraft. This system is independent because no help is required from the aircraft to be "seen" by the radar. Primary radar can only provide the position of the aircraft. Secondary radar operates by using highly directional transmissions of enquiries. Aircraft are expected to respond to the query in a fixed time. The position of the aircraft is determined from the position of the antenna and the time required to hear a response from an aircraft. The response can (and does) contain information, such as the aircraft's identity and its altitude. Where primary radar is considered independent, secondary radar can be considered to be "cooperative" surveillance.

As secondary radar depends upon the aircraft responding to an enquiry, it will not "see" aircraft that do not respond. Typically, primary and secondary radar antennae are installed together on the same rotating mount and used together to complement one another. If the primary radar detects something that is not responding to secondary radar enquiries, the air traffic controller can take appropriate action.

Passive surveillance consists of equipment that listens for transmissions from aircraft and then computes the position using that transmission; the surveillance system makes no request of the aircraft for transmission. There are two general techniques in use:

- The aircraft broadcasts its identity and position information, which is used as is.
- The surveillance system uses a network of multiple receivers and multilateration (intersection of the hyperboloids described by the difference in arrival time of the transmission at each receiver) to determine the position of the transmitter.

The first technique is known as ADS-B (Automatic Dependent Surveillance-Broadcast). This uses satellite navigation technology on board the aircraft to determine where the aircraft is and then to broadcast that position to other users without the need for any pilot input or radar interrogation. This technique depends upon the aircraft knowing its accurate position. An aircraft that either maliciously or through equipment failure reports an incorrect position will be misplaced; the only sanity check available is to check if a position report makes sense (is credible). Receiving credible but erroneous information is a key problem to be addressed.

While ADS-B can be used by ground users as a replacement for traditional surveillance techniques like radar, it is also seen as an enabling technology for new methods of air traffic control. The broadcast of surveillance data that can be received by all users, including other aircraft, may permit tasks normally undertaken by a controller to be delegated to the pilot. These ideas are encompassed in the concept of Airborne Separation Assistance Systems (ASAS) [13].

The second technique has similar characteristics to secondary radar; the computation of the position depends solely upon the timing of receipt of signals.

Neither secondary radar nor passive surveillance detect noncooperating aircraft.

A.3 Increasing Use of Passive Surveillance

The use of passive surveillance has become more attractive to Air Traffic Control Service Providers (ANSPs) in recent years because aircraft are increasingly being equipped with suitable avionics. In addition to the perceived operational benefits of these technologies, there are potentially significant cost savings in procurement and through-life maintenance costs of these technologies over traditional surveillance means. The open question, and the reason for the existence of many projects, including CRISTAL UK, is whether these benefits can be obtained with adequate safety and security.

A.4 Using ADS-B to Achieve the Benefits

In order to obtain the majority of the benefits of passive surveillance, there must be aircraft-based equipment available that reports the required information about the aircraft. The ADS-B standard and complying equipment will meet this need.

ADS-B-equipped aircraft broadcast information approximately once per second. These transmissions will include information about the position and status of the aircraft. The information is broadcast in various messages that include airborne position, surface position, aircraft identification and type, airborne velocity, and aircraft operational status messages [11]. This information is collected by ADS-B receivers and then passed to air traffic control processing systems to be displayed to the controller, either on existing displays (preferred) or on some new display. The information broadcast by an ADS-B system is derived both from the avionic systems in the aircraft (e.g., air speed, barometric altitude, aircraft status) and from satellite navigation equipment (e.g., surface position, geometric altitude, and ground speed). ADS-B messages are not "signed" in any fashion; one cannot verify that a message actually comes from the aircraft identified in the contents of the message.

Acknowledgments

The financial support of the Leverhulme Trust, the Royal Academy of Engineering, the European Union, and the EPSRC is gratefully acknowledged.

References

[1] I. Alexander, "Misuse Cases in Systems Engineering," *Computing and Control Eng. J.*, vol. 14, no. 1, pp. 40-45, Feb. 2003.

[2] J.H. Allen, "CERT System and Network Security Practices," *Proc. Fifth Nat'l Colloquium Information Systems Security Education,* 2001.

[3] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems,* 2001.

[4] A.I. Antón and J.B. Earp, "Strategies for Developing Policies and Requirements for Secure E-Commerce Systems," *E-Commerce Security and Privacy,* vol. 2, Advances In Information Security, A.K. Ghosh, eds., pp. 29-46, Kluwer Academic, 2001.

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004.

[6] T.D. Breaux, M.W. Vail, and A.I. Antón, "Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations," *Proc. 14th IEEE Int'l Requirements Eng. Conf.,* pp. 46-55, 2006.

[7] D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy," *Proc. 1989 IEEE Symp. Security and Privacy,* pp. 206- 214, 1989.

[8] S.J. Buckingham Shum, "The Roots of Computer Supported Argument Visualization," *Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making,* P.A. Kirschner, S.J. Buckingham Shum, and C.S. Carr, eds., pp. 3-24, Springer-Verlag, 2003.

[9] J.E. Burge and D.C. Brown, "An Integrated Approach for Software Design Checking Using Design Rationale," *Proc. First Int'l Conf. Design Computing and Cognition,* J.S. Gero, ed., pp. 557-576, 2004.

[10] S. Capkun and J.-P. Hubaux, "Securing Position and Distance Verification in Wireless Networks," Technical Report EPFL/IC/200443, Swiss Federal Inst. of Technology Lausanne, May 2004.

[11] "Australian Technical Standard Order: Airborne Stand-Alone Extended Squitter, Automatic Dependent Surveillance-Broadcast (ADS-B), Transmit Only Equipment," Australian Civil Aviation Safety Authority, Standard ATSO-C1005, CASA, Dec. 2004.

[12] "CERT/CC Statistics 1988-2005," Pittsburgh, CERT CC, http://www.cert.org/stats/cert_stats.html, Feb. 2006.

[13] F. Cervo, "Airborne Separation Assistance Systems," EUROCONTROL, 2005, Newsletter, http://www.eurocontrol.int/mil/public/standard_page/newsletter0605art2.html, Sept. 2006.

[14] H. Chivers and M. Fletcher, "Applying Security Design Analysis to a Service-Based System," *Software: Practice and Experience,* vol. 35, no. 9, pp. 873-897, 2005.

[15] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering.* Kluwer Academic, 2000.

[16] Common Criteria Sponsoring Organizations, "Common Criteria for Information Technology Security Evaluation Part 1: Introduction and General Model, Version 3.1 Rev 1," Nat'l Inst. of Standards and Technology CCMB-2006-09-001, Sept. 2006.

[17] Common Criteria Sponsoring Organizations, "Common Criteria for Information Technology Security Evaluation Part 2: Security Functional Components, Version 3.1 Rev 1," Nat'l Inst. of Standards and Technology CCMB-2006-09-002, Sept. 2006.

[18] Common Criteria Sponsoring Organizations, "Common Criteria for Information Technology Security Evaluation Part 3: Security Assurance Components, Version 3.1 Rev 1," Nat'l Inst. Standards and Technology CCMB-2006-09-003, Sept. 2006.

[19] "Compendium," http://www.compendiuminstitute.org/, Compendium Inst., 2005.

[20] J. Conklin, "Dialog Mapping: Reflections on an Industrial Strength Case Study," *Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making,* P.A. Kirschner, S.J. Buckingham Shum, and C.S. Carr, eds., pp. 117-136, Springer-Verlag, 2003.

[21] E. Dash, "Weakness in the Data Chain," *New York Times,* 2005.

[22] R. De Landtsheer and A. van Lamsweerde, "Reasoning About Confidentiality at Requirements Engineering Time," *Proc. 10th European Software Eng. Conf. (ESEC-FSE'05) with 13th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.,* pp. 41-49, 2005.

[23] P. Devanbu and S. Stubblebine, "Software Engineering for Security: A Roadmap," *The Future of Software Eng.,* A. Finkelstein, ed., ACM Press, 2000.

[24] "Matter of CardSystems Solutions Inc.," Washington, D.C., Federal Trade Commission, 2006.

[25] A. Finkelstein and H. Fuks, "Multiparty Specification," *Proc. Fifth Int'l Workshop Software Specification and Design,* pp. 185-195, 1989.

[26] D.G. Firesmith, "Common Concepts Underlying Safety, Security, and Survivability Engineering," Technical Report CMU/SEI-2003-TN-033, Software Eng. Inst., Carnegie Mellon Univ., Dec. 2003.

[27] D.G. Firesmith, "Specifying Reusable Security Requirements," *J. Object Technology,* vol. 3, no. 1, pp. 61-75, Jan.-Feb. 2004.

[28] G. Fischer, A.C. Lemke, R. McCall, and A. Morch, "Making Argumentation Serve Design," *Design Rationale Concepts, Techniques, and Use,* T. Moran and J. Carroll, eds., pp. 267-293, Lawrence Erlbaum and Assoc., 1996.

[29] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements through Ownership, Permission and Delegation," *Proc. 13th IEEE Int'l Conf. Requirements Eng.,* pp. 167-176, 2005.

[30] M. Glinz, "Rethinking the Notion of Non-Functional Requirements," *Proc. Third World Congress for Software Quality,* vol. II, pp. 55-64, 2005.

[31] C.B. Haley, R.C. Laney, and B. Nuseibeh, "Deriving Security Requirements from Crosscutting Threat Descriptions," *Proc. Third Int'l Conf. Aspect-Oriented Software Development,* pp. 112-121, 2004.

[32] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "The Effect of Trust Assumptions on the Elaboration of Security Requirements," *Proc. 12th Int'l Requirements Eng. Conf.,* pp. 102-111, 2004.

[33] C.B. Haley, J.D. Moffett, R. Laney, and B. Nuseibeh, "Arguing Security: Validating Security Requirements Using Structured Argumentation," *Proc. Third Symp. Requirements Eng. for Information Security with the 13th Int'l Requirements Eng. Conf.,* 2005.

[34] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "Arguing Satisfaction of Security Requirements," *Integrating Security and Software Eng.: Advances and Future Vision,* H. Mouratidis and P. Giorgini, eds., pp. 16-43, Idea Group, 2006.

[35] C.B. Haley, J.D. Moffett, R. Laney, and B. Nuseibeh, "A Framework for Security Requirements Engineering," *Proc. 2006 Software Eng. for Secure Systems Workshop with the 28th Int'l Conf. Software Eng.,* pp. 35-41, 2006.

[36] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "Using Trust Assumptions with Security Requirements," *Requirements Eng. J.,* vol. 11, no. 2, pp. 138-151, Apr. 2006.

[37] C.L. Heitmeyer, "Applying "Practical" Formal Methods to the Specification and Analysis of Security Properties," *Proc. Int'l Workshop Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Computer Security,* pp. 84-89, 2001.

[38] ISO/IEC, "Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 1: Introduction and General Model," ISO/IEC, Geneva, Switzerland, Int'l Standard 15408-1, Dec. 1999.

[39] ISO/IEC, "Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 2: Security Functional Requirements," ISO/IEC, Geneva, Switzerland, Int'l Standard 15408-2, Dec. 1999.

[40] ISO/IEC, "Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 3: Security Assurance Requirements," ISO/IEC, Geneva, Switzerland, Int'l Standard 15408-3, Dec. 1999.

[41] M. Jackson, *Software Requirements and Specifications.* Addison Wesley, 1995.

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

[43] E. Jonsson, "An Integrated Framework for Security and Dependability," *Proc. 1998 Workshop New Security Paradigms,* pp. 22-29, 1998.

[44] E. Kavakli, "Goal-Oriented Requirements Engineering: A Unifying Framework," *Requirements Eng. J.,* vol. 6, no. 4, pp. 237-251, Jan. 2002.

[45] T.P. Kelly, "Arguing Safety—A Systematic Approach to Safety Case Management," D.Phil dissertation, Univ. of York, 1999.

[46] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques.* John Wiley and Sons, 1998.

[47] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," *Proc. Fifth IEEE Int'l Symp. Requirements Eng.,* pp. 249-263, 2001.

[48] A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," *Proc.e 26th Int'l Conf. Software Eng.,* pp. 148-157, 2004.

[49] S. Lautieri, D. Cooper, and D. Jackson, "SafSec: Commonalities Between Safety and Security Assurance," *Constituents of Modern System-Safety Thinking: Proc. 13th Safety-Critical Systems Symp.,* F. Redmill and T. Anderson, eds., pp. 65-78, 2005.

[50] J. Lee and K.-Y. Lai, "What's in Design Rationale?" *Human-Computer Interaction,* vol. 6, nos. 3-4, pp. 251-280, 1991.

[51] Y. Lee, J. Lee, and Z. Lee, "Integrating Software Lifecycle Process Standards with Security Engineering," *Computers and Security,* vol. 21, no. 4, pp. 345-355, 2002.

[52] N.G. Leveson, "Software Safety: Why, What, and How," *ACM Computing Surveys,* vol. 18, no. 2, pp. 125-163, June 1986.

[53] L. Liu, E. Yu, and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting," *Proc. 11th IEEE Int'l Requirements Eng. Conf.,* pp. 151-161, 2003.

[54] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," *Proc. 15th Computer Security Applications Conf.,* pp. 55-64, 1999.

[55] N.R. Mead, E.D. Hough, and T.R. Stehney II, "Security Quality Requirements Engineering (SQUARE) Methodology," CMU/SEI, Technical Report CMU/SEI-2005-TR-009, ESC-TR-2005-009, Nov. 2005.

[56] J.D. Moffett, J.G. Hall, A. Coombes, and J.A. McDermid, "A Model for a Causal Logic for Requirements Engineering," *Requirements Eng.,* vol. 1, no. 1, pp. 27-46, Mar. 1996.

[57] J.D. Moffett, C.B. Haley, and B. Nuseibeh, "Core Security Requirements Artefacts," Technical Report 2004/23, Dept. of Computing, The Open Univ., June 2004.

[58] H. Mouratidis, P. Giorgini, and G. Manson, "Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems," *Proc. 15th Conf. Advanced Information Systems Eng.,* pp. 63-78, 2003.

[59] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing Knowledge about Information Systems," *ACM Trans. Information Systems,* vol. 8, no. 4, pp. 325-362, Oct. 1990.

[60] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach," *IEEE Trans. Software Eng.,* vol. 18, no. 6, pp. 483-497, June 1992.

[61] S.E. Newman and C.C. Marshall, "Pushing Toulmin Too Far: Learning from an Argument Representation Scheme," Technical Report SSL-92-45, Xerox PARC, 1991.

[62] NIST, *An Introduction to Computer Security: The NIST Handbook,* Nat'l Inst. of Standards and Technology (NIST), special publication SP 800-12, Oct. 1995.

[63] B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer,* vol. 34, no. 3, pp. 115-117, Mar. 2001.

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

[65] C. Potts and G. Bruns, "Recording the Reasons for Design Decisions," *Proc. 10th Int'l Conf. Software Eng.,* pp. 418-427, 1988.

[66] B. Ramesh and V. Dhar, "Supporting Systems Development by Capturing Deliberations during Requirements Engineering," *IEEE Trans. Software Eng.,* vol. 18, no. 6, pp. 498-510, June 1992.

[67] "Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software," version 1.05.245, S.T. Redwine Jr., ed., Dept. of Homeland Security, Aug. 2006.

[68] J. Rushby, "Security Requirements Specifications: How and What," *Proc. Symp. Requirements Eng. for Information Security,* 2001.

[69] Senior Officials Group-Information Systems Security, "Information Technology Security Evaluation Criteria (ITSEC)," version 1.2, Dept. of Trade and Industry, June 1991.

[70] G. Sindre and A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases," *Proc. 37th Int'l Conf. Technology of Object-Oriented Languages and Systems,* pp. 120-131, 2000.

[71] E.H. Spafford, "The Internet Worm Program: An Analysis," *ACM SIGCOMM Computer Comm. Rev.,* vol. 19, no. 1, pp. 17-57, Jan. 1989.

[72] "The Chaos Report," research report, Standish Group, 1995.

[73] "Chaos: A Recipe for Success," research report, Standish Group, 1999.

[74] "Extreme Chaos," research report, Standish Group, 2001.

[75] O. Tettero, D.J. Out, H.M. Franken, and J. Schot, "Information Security Embedded in the Design of Telematics Systems," *Computers and Security,* vol. 16, no. 2, pp. 145-164, 1997.

[76] K. Thompson, "Reflections on Trusting Trust," *Comm. ACM,* vol. 27, no. 8, pp. 761-763, Aug. 1984.

[77] S.E. Toulmin, *The Uses of Argument.* Cambridge Univ. Press, 1958.

[78] S.E. Toulmin, R.D. Rieke, and A. Janik, *An Introduction to Reasoning.* Macmillan, 1979.

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

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

[81] M. Watson, *UK ADS-B in a Radar Environment,* EUROCONTROL, 2006, presentation slides, http://www.eurocontrol.int/cascade/gallery/content/public/documents/Presentations/Session%202%20-%20Trials%20and%20Implementations/Watson%20-%20UK%20ADS-B%20in%20a%20radar%20environment.pdf, 2007.

[82] L. Zhuang, F. Zhou, and J.D. Tygar, "Keyboard Acoustic Emanations Revisited," *Proc. 12th ACM Conf. Computer and Comm. Security,* pp. 373-382, 2005.

Charles B. Haley received the PhD degree in security requirements from the Open University and the MS and BA degrees from the University of California at Berkeley. He is a lecturer in the Department of Computing at the Open University. Before reentering the academic community in 1999, he worked for 25 years in the software industry at companies including Bell Laboratories, Rational Software, Bell Northern Research, and Sun Microsystems, holding positions ranging from software engineer to director of development. He joined the Open University in 2003, where his research is in the representation of security requirements and their validation through formal and informal argumentation.

Robin Laney received the First Class Honours BSc degree in microcomputers and applications from Westfield College, University of London, and the PhD degree in computing from King's College, University of London. He is a senior lecturer in the Department of Computing at the Open University. His research interests include requirements engineering, flexible approaches to software architecture, and music computing. His research mission is to focus on activities that bridge the gap between theoretical advances and the experience and problems of working software practitioners, in both directions. He has industrial experience as a software engineer working on programming language technology and graphics.

Jonathan D. Moffett received the bachelor's degree in mathematics and theoretical physics in 1961 from Trinity College, Cambridge, and the PhD degree in computing in 1990 from Imperial College, University of London. He is a visiting senior research fellow in the Department of Computing at the Open University, and was a senior lecturer in the Computer Science Department at the University of York, England, until his retirement. He acted as Advanced MSc Course Organiser there from 1994-2001. He was a member of the Distributed Systems Engineering Group at Imperial College, London, from 1986-1992. His previous experience has been as a systems consultant on large commercial systems, including acting as computer controls and security adviser at Esso Europe Inc. and as a consultant to one of the London Clearing Banks, for whom he wrote their computer security standards. He is a Chartered Engineer, a fellow of the Association of Certified Accountants, a member of the British Computer Society, and a member of the IEEE.

Bashar Nuseibeh received the MSc and PhD degrees in software engineering from Imperial College, London, and the First Class Honours BSc degree in computer systems engineering from the University of Sussex, United Kingdom. He is a professor and director of Research in Computing at the Open University (OU) and a visiting professor at Imperial College, London, and the National Institute of Informatics, Japan. Previously, he was a reader at Imperial College, London, and head of its Software Engineering Laboratory. His research interests are in software requirements engineering and design, software process modeling and technology, and technology transfer. He has published more than 100 refereed papers and consulted widely with industry, working with organizations such as the UK National Air Traffic Services (NATS), Texas Instruments, Praxis Critical Systems, Philips Research Labs, and NASA. He has also served as principal investigator on a number of research projects on software engineering, security engineering, and learning technologies. He is editor-in-chief of the *Automated Software Engineering Journal* and an associate editor of the *IEEE Transactions on Software Engineering*, the *Requirements Engineering Journal*, and a number of other international journals. He was a founder and first chairman of the BCS Requirements Engineering Specialist Group (1994-2004) and is currently chair of IFIP Working Group 2.9 (Software Requirements Engineering) and chair of the Steering Committee of the International Conference on Software Engineering (ICSE). He has served as program chair of major conferences in his field, including ASE '98, RE '01, and ICSE '05. He received a Philip Leverhulme Prize (2002), an ICSE "Most Influential Paper" award (2003), a "Best Application Paper" award from the 18th International Conference on Logic Programming (ICLP '02), and a number of other best paper and service awards. He held a Senior Research Fellowship of the Royal Academy of Engineering and The Leverhulme Trust between 2005-2007. He is a fellow of the British Computer Society (FBCS) and the Institution of Engineering and Technology (FIET), a Chartered Engineer (C.Eng.), and a member of the IEEE Computer Society.