A framework for security requirements engineering

Conference or Workshop Item

A Framework for Security Requirements Engineering

Charles B. Haley
The Open University
Walton Hall
Milton Keynes UK
c.b.haley [at] open.ac.uk

Jonathan D. Moffett
The Open University
Walton Hall
Milton Keynes UK
j.moffett [at] open.ac.uk

Robin Laney
The Open University
Walton Hall
Milton Keynes UK
r.c.laney [at] open.ac.uk

Bashar Nuseibeh
The Open University
Walton Hall
Milton Keynes UK
b.nuseibeh [at] open.ac.uk

ABSTRACT
This paper presents a framework for security requirements elicitation and analysis, based upon the construction of a context for the system and satisfaction arguments for the security of the system. One starts with enumeration of security goals based on assets in the system. These goals are used to derive security requirements in the form of constraints. The system context is described using a problem-centered notation, then this context is validated against the security requirements through construction of a satisfaction argument. The satisfaction argument is in 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.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications.

General Terms
Documentation, Security, Verification.

Keywords
Requirements Engineering, Security Requirements

1. INTRODUCTION
Security is about the prevention of harm caused by the actions of attackers. Attackers are people who gain by exploiting system failures, intentionally or accidentally provoked. This gain usually results in some harm to the system owner. The attacker manipulates some *object* in the system during an attack, to harmful effect. Objects so manipulated have *value* (the inverse of the harm) that must be protected.

Security goals arise when stakeholders establish that objects in the context of the system, be they tangible (e.g. cash) or intangible (e.g. information), have direct or indirect value that the stakeholders wish to preserve. The value is *direct* when the object has an intrinsic value, almost regardless of context. The value is *indirect* when the value arises because of the context in which the object rests. Objects with direct or indirect value are called *assets* [13], and the stakeholders naturally wish to protect themselves from any harm that might come from abuse of these assets. For example, tangible assets might be destroyed, stolen, or modified; in this case, the harm is the loss of the asset itself (direct value). Information assets might be destroyed, revealed, or modified; the harm could be the loss or modification of the asset (direct value) or the consequences of exposing the asset (indirect value). Security requirements operationalize security goals, describing conditions under which the possibility of intentionally caused harm is reduced to an acceptable level.

SESS'06, May 20–21, 2006, Shanghai, China.

This paper is about security requirements, and is targeted at readers interested in early security analysis. There are three contributions. The first is *definition*: what security requirements are. The second is explicit inclusion of *context*: the world within which the system and the potential attackers rest. The third is determining *satisfaction*: whether the system can satisfy the security requirements. The paper can be thought of as a prequel to [11], which first presented our security satisfaction arguments, along with examples.

2. RELATED WORK
This section looks at related work on how security requirements are defined and represented.

2.1 Security Requirements as Security Functions
It is common to express security requirements by describing the security mechanisms to be used. For example, ISO 15408 [13-15], the ISO version of the Common Criteria, 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 …]" [14], 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]" [14]. 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)" [23], 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.

2.2 Security Requirements as NFRs

Devanbu & Stubblebine [7] remark that security requirements are a kind of non-functional requirement. Kotonya and Sommerville [17], when discussing non-functional requirements, in which they include security, define them as "restrictions or constraints" on system services; similar definitions can be found in other text books. Rushby [26] appears to take a similar view, stating "security requirements mostly concern what must not happen". Using the Tropos methodology, Mouratidis et al [22] state that "security constraints define the system's security requirements".

Firesmith in [8] 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%."

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 the system against any eventual chosen constraint 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%" mean? It could mean that if 10,000 attacks are known, the developers can ignore one. Alternatively, it could be a way of saying "all" without actually saying it.

One major problem with percentage-style quantification of security requirements is the binary nature of the majority of security attacks; in most cases, an attack works or it does not. If an attack does not work the first time, it probably will not work the second time unless the parameters of the attack are changed. On the other hand, if the attack works once (the system is penetrated), then the attack will likely continue working until the vulnerability is removed. The result is that successful attacks can (usually) be repeated as often as the attacker wishes, and even shared amongst attackers. It is difficult to know what use to make of the percentage quantification in these cases.

2.3 Other Portrayals of Security Requirements

Many authors implicitly assume that security requirements are identical to high-level security goals. Tettero [29] is 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*.

Some authors identify security requirements with security policies. Devanbu & Stubblebine [7] 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". The difficulty with security policies is their 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 [19] point out the importance of considering security requirements in the development life cycle, but do not define them. Heitmeyer [12] shows how the SCR 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 & Fox [21], followed independently by Sindre & Opdahl [27] and elaborated by Alexander [1], describe abuse and misuse cases, extending the use case paradigm to undesired behavior. Liu, Yu & Mylopoulos [20] 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 [6] fall into this camp with SeDAn, as they focus on attackers and the paths they might take into a system. The problem with these approaches is that they indicate what a system is not to do in specific situations, but not in the general case.

van Lamsweerde [18] describes a process by which security goals are made precise and refined until reaching security requirements. Antón & Earp [4] use the GBRAM method to operationalize security goals for the generation of security policies and requirements, but do not define security requirements.

3. THE FRAMEWORK

The review in Section 2 exposed several problem areas: multiple definitions of security requirements, inconsistent and difficult to understand satisfaction criteria, and a general lack of a clear pathway for deriving security requirements from business goals. The security requirements framework described in this paper addresses these problems, facilitating an understanding of the elicitation, validation, and verification of security requirements and other artifacts by integrating the concepts of the two disciplines of requirements engineering and security engineering. From requirements engineering it takes the concept of functional

goals, which are operationalized into functional requirements while applying appropriate constraints. From security engineering it takes the concept of assets, together with threats of harm to those assets. In the framework:

- Security goals aim to protect assets from harm.
- Security goals are operationalized into security requirements, which take the form of a set of constraints on the functional requirements sufficient to protect the assets from the harms identified previously. Security requirements are, consequently, preventative.
- Feasible realizations of the security requirements may lead to the need for the addition of secondary security goals, which will (eventually) manifest themselves as additional functional and/or security requirements. Secondary security goals may call for detective or preventative measures, a possibility which is discussed further below.
- Security satisfaction arguments show that the system can respect the security requirements.

The framework was developed in order to understand 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, labeling them with the acronym CIA, and more recently another A ([25] and other security textbooks): confidentiality, integrity, accessibility, and accountability. By connecting these general concerns to the assets implicated in a system, and then postulating *actions* that would violate these concerns, one can construct descriptions of possible threats on assets. These *threat descriptions* [10] are phrases of the form *performing action X on/to/with asset Y could cause harm Z.* Threat descriptions permit a form of asset-centered threat modeling, and are represented by a three-element tuple: the asset, the action that will exploit the asset, and the subsequent harm. Threat descriptions are generated by enumerating the assets involved in the system, then for each asset, listing the *actions* that exploit the asset to cause direct or indirect harm. For example, one can imagine *erasing* (the action) the *customer records* (the asset) of a company to cause *loss of revenue* (the harm). Security goals are found by applying *prevent* (also called *avoid*) to threat descriptions.

More security goals can be found by combining management control principles and application business goals. Management control principles include ‘standard’ security principles such as *least privilege* and *separation of duties*. Application business goals will determine the applicability of management control principles to the system, for example by defining those privileges that are needed for the application, and excluding those that are not. Security goals found in this way have a form similar to “signatures of two separate people will be required for any expenditure over £1000” and “privilege to perform operation X shall not imply privilege to perform operation Y”.

Note that other legitimate 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.

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 a zero sum game like football. In football, the goals won by an attacker are exactly the goals lost by the defender. Security is different; there is no exact equivalence between the losses incurred by the asset owner and the gains of the attacker. To see this, look at two examples:

- Robert Morris unleashed the Internet Worm [28], 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.
- Many virus writers today are prepared to expend huge effort in writing a still more ingenious virus, which may cause little damage (screen message "You've got a Virus"). Generally, there is no simple relationship between the gains of a virus writer and the losses incurred by those who are attacked.

The consequences of security not being a zero sum game 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 needs only to determine that harm can be incurred. The second is that the goals of attackers cannot be solely used to arrive at the goals of a defender to prevent harm, i.e. their security goals; further consideration is necessary to determine whether and what harm 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 themselves being functional requirements. As constraints, they are preventative measures.
- 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

There are two related sets of security goals and security requirements. The first, the *primary* goals and requirements, are those derived from the business goals and functional requirements. These goals and requirements are *primary* in the sense that if the resulting system will respect the primary security requirements, then the system will satisfy the primary security goal(s).

Secondary security goals are additional goals that are added for one or both of the following reasons: 1) to enable construction of an acceptable satisfaction argument for the satisfaction of primary

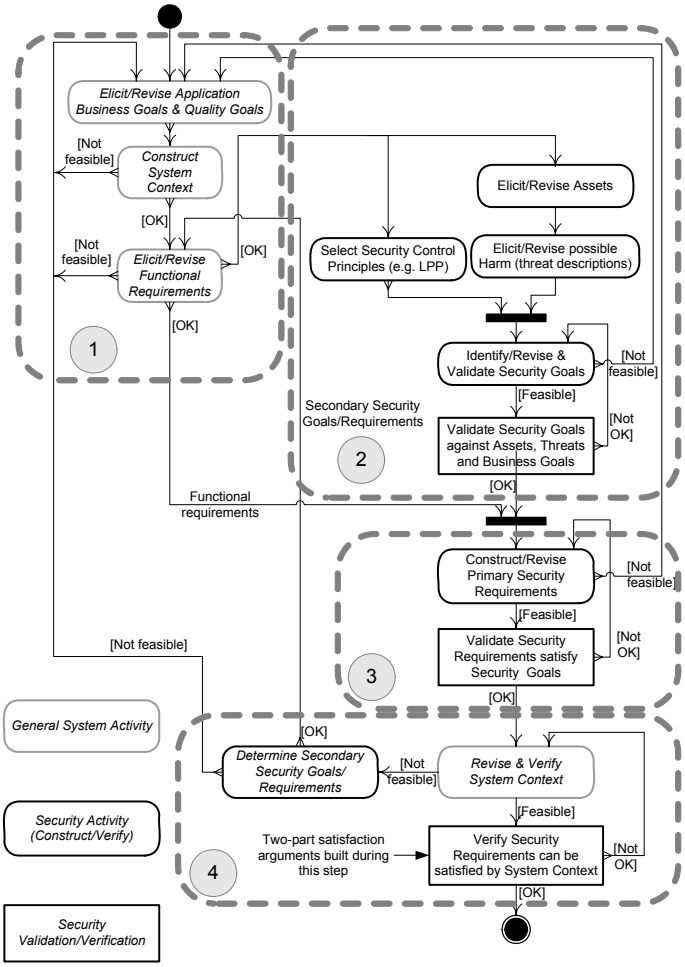

Figure 1 – Process Activity Diagram

security requirements, or 2) to permit an acceptable *feasible realization* of the primary security requirements. Satisfaction arguments are discussed later in this paper.

The term *feasible realization* takes into consideration both technical feasibility and cost/benefit plus risk. In some cases, there may be no known way to respect a constraint and thereby prevent the harm; destroying a computer room with an atomic explosion comes to mind. In other cases, risk analysis might indicate that the cost of respecting a security requirement is excessive. In these cases, the analyst may decide to detect violation after the fact, and then both recover from and repair the breach. Availability requirements are a good example - many such requirements do not *prevent* loss of availability, but instead imply a recovery capability. Analysis of the secondary security goals may lead to the addition of secondary security requirements. This is, of course, a recursive process.

Secondary security goals and security requirements are not secondary in terms of importance, but are instead secondary because they exist to enable satisfaction (to an acceptable level) of hierarchically superior security requirements.

It is very important to note that adding secondary security goals and requirements can supersede the primary security requirement, 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 directly satisfied, as the attack will not be prevented. The choice means that the detection goals (and associated security requirements) are considered suitably equivalent; they 'cover' and 'replace' (but do not delete) the primary security requirement. The same choice, use detection instead of prevention, could also change the behavior specification of the system because of the addition of domains and phenomena to facilitate detection.

3.4 Security Requirements and Context

It is important to reiterate that security requirements are applied in the context within which the system operates, which is larger than the software. A security requirement can be realized in multiple ways, some completely outside the software to be constructed.

We use a variant of Jackson's problem frames [16] to represent the system context; see [11] for examples.

3.5 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 an abstraction hierarchy, progressing from the most abstract to the final concrete working system. At early stages, core artifacts are typically documents 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 counter-example, etc. They are by-products of processes whose aim is to help produce verified and valid core artifacts.

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

Dependencies between Artifacts. In a hierarchy of artifacts, there are dependencies between the artifacts. 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. This kind of dependency is called *hierarchical dependency*.

There is also a reverse kind of dependency: *feasibility*. If it proves impossible to implement a system that sufficiently satisfies a requirements specification, then this will force a change in the goals or requirements. 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 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.

3.6 Activities in the Framework

An ordered set of activities for moving from functional goals to satisfaction arguments is shown in Figure 1. 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 – such as validation of security requirements against security goals – shows that they fail to meet their aims. For example, one might be unable to construct a system context that is both accepted by the stakeholders and permits derivation of functional requirements from the functional goals. Alternatively, one might fail to construct a satisfactory satisfaction argument. Iteration may cascade upwards if the problem cannot be resolved at the preceding step.

There are four general sections in the activity diagram. Although one could describe these sections 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.

3.6.1 Section 1: Identify Functional Requirements

The only requirement the framework places upon the development process is that one output a representation of the context. How the requirements engineer gets to this point is open.

3.6.2 Section 2: Identify Appropriate Security Goals

There are four general steps required to identify the security goals: identify candidate assets, identify harms (generate threat descriptions), apply management principles, 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).

3.6.2.1 Identify Candidate Assets

The goal of this step is to find all the objects in the system context that might have value, direct or indirect. In general, assets consist of all the information objects stored in or accessed by the system-to-be and any tangible objects such as the computers themselves. An object has direct value when the potential harm caused by a threat is to the object itself. An object has indirect value when a threat involving that asset causes harm somewhere else, such as to revenue, to costs, or to reputation. An object can have both direct and indirect value; when money is taken from a bank, the bank loses both the money and its reputation.

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

3.6.2.2 Generate Threat Descriptions

In general, harm is caused by the negation of one or more of the security concerns described in Section 3.1. For information assets, these concerns are *confidentiality*, *integrity*, and *availability*. The concerns are similar for tangible assets: *exposure*, *modification*, and *deprivation* (theft or destruction). These concerns are used to enumerate the threat descriptions. One asks questions of the form "what harm could come from violating the [insert concern here] of [insert asset here]?" Answers to these questions are threat descriptions, which are represented as tuples of the form `{action, asset, harm}`.

3.6.2.3 Apply 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, and data protection (not intended to be an exhaustive list). The sector the system will run in may have standard management principles, such as no outside connections and no removable media. A list of security goals is generated by checking the applicability of 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."

3.6.3 Step 3: Identify Security Requirements

Recall that we define security requirements as constraints on functional requirements that are needed to satisfy applicable security goals. Two operations are needed to establish which goals are applicable to a functional requirement: determining which assets are involved to find the threat descriptions that apply, and settling on appropriate goals derived from management principles. One finds which assets will be implicated in satisfying a particular functional requirement by drawing the context for that functional requirement as a problem diagram. The list of assets implicated in the context leads to a list of threats descriptions that must be mitigated. The list of assets and the function itself will point at the management principle goals to apply. The security requirements are these mitigations or *achieve* goals, constraining the function in ways that will achieve the security goals.

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 only makes sense in the context of the function. One might also impose temporal constraints:

```
The system shall provide Personnel Information
only during normal office hours;
```

and complex constraints on traces:

```
The system shall provide information about an
organization only to any person who has not
previously accessed information about a
competitor organization (the Chinese Wall
Security Policy, [5]).
```

Availability requirements will need to express constraints on response time:

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

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

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

3.6.4 Step 4: Validation of System Context

A key verification step for the framework described in this paper is the ability to show that the system can satisfy the security requirements. We propose the use of structured informal and formal argumentation for this verification step: to convince a reader that a system can satisfy the security requirements laid upon it. These arguments, called *satisfaction arguments* and discussed more completely in [11], are in two parts. The first part of the argument consists of a formal argument to prove that a system can satisfy its security requirements, drawing upon claims about the behavior and properties of domains in a system. The claims about behavior of the domains are *trust assumptions* [9]. The second part of the argument consists of structured informal arguments to support the trust assumptions about system behavior and characteristics made in the formal argument. Building on our understanding of security requirements, the satisfaction arguments assist with identifying security-relevant system properties, and determining how inconsistent and implausible assumptions about them affect the security of a system.

3.7 Iteration

One reason that an analyst may fail to construct a convincing satisfaction argument is that there is not enough information available to justify the using a trust 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 [24], asking the designers to add more detail into the system context so that claims can be justified. These iterations move from step four to steps one and two.

The details added during a Twin Peaks iteration may well require new functions, thus functional requirements. Consider a system where to satisfy a confidentiality requirement, designers choose authentication. Further assume that the designers choose 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 step 1 with a re-analysis 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 step 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 the Twin Peaks 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 to the system, 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 satisfies 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.

Lastly, 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. CONCLUSION

This paper has presented a framework for security requirements engineering where a) asset and security goal analysis are done in the business context of the system, b) the effects of security requirements on the functional requirements are understood, c) design constraints are taken into account, and d) the 'correctness' of security requirements is established through the use of satisfaction arguments. The framework unifies our work on early threat analysis [10], trust assumptions [9], and satisfaction arguments [11].

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.

Acknowledgements:

The authors wish to thank Michael Jackson for his continuous involvement and support. Thanks also to Simon Buckingham Shum for many helpful conversations about argumentation. The financial support of the Leverhulme Trust and the Royal Academy of Engineering is gratefully acknowledged, as is EU support of the ELeGI project, number IST-002205.

References:

[1] Alexander, I.: Misuse Cases in Systems Engineering. Computing and Control Engineering Journal, 14(1) (Feb 2003), 40-45.

[2] Allen, J.H.: CERT System and Network Security Practices. In Proceedings of the Fifth National Colloquium for Information Systems Security Education (NCISSE'01), George Mason University, Fairfax, VA USA, 22-24 May 2001.

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

[4] Antón, A.I., Earp, J.B.: Strategies for Developing Policies and Requirements for Secure E-Commerce Systems. In E-Commerce Security and Privacy, vol. 2, Advances In Information Security, A. K. Ghosh, Ed.: Kluwer Academic Publishers, Jan 15 2001, pp. 29-46.

[5] Brewer, D.F.C., Nash, M.J.: The Chinese Wall security policy. In Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, CA USA: IEEE Computer Society Press, 1-3 May 1989, pp. 206 - 214.

[6] Chivers, H., Fletcher, M.: Applying Security Design Analysis to a service-based system. Software: Practice and Experience, 35(9) (2005), 873-897.

[7] Devanbu, P., Stubblebine, S.: Software Engineering for Security: A Roadmap. In The Future of Software Engineering, A. Finkelstein, Ed.: ACM Press, 2000.

[8] Firesmith, D.: Specifying Reusable Security Requirements. Journal of Object Technology, 3(1) (Jan-Feb 2004), 61-75.

[9] Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B.: Using Trust Assumptions with Security Requirements. Requirements Engineering Journal, 11(2) (April 2006), 138-151.

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

[11] Haley, C.B., Moffett, J.D., Laney, R., Nuseibeh, B.: Arguing Security: Validating Security Requirements Using Structured Argumentation. In Proceedings of the Third Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05), Paris France, 29 Aug 2005.

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

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

[14] ISO/IEC: Information Technology - Security Techniques - Evaluation Criteria for IT Security - Part 2: Security Functional Requirements. International Standard 15408-2, ISO/IEC, Geneva Switzerland, 1 Dec 1999.

[15] ISO/IEC: Information Technology - Security Techniques - Evaluation Criteria for IT Security - Part 3: Security Assurance Requirements. International Standard 15408-3, ISO/IEC, Geneva Switzerland, 1 Dec 1999.

[16] Jackson, M.: Problem Frames. Addison Wesley, 2001.

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

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

[19] Lee, Y., Lee, J., Lee, Z.: Integrating Software Lifecycle Process Standards with Security Engineering. Computers and Security, 21(4) (2002), 345-355.

[20] Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis Within a Social Setting. In Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03), Monterey, CA USA, 8-12 Sept 2003, pp. 151-161.

[21] McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In Proceedings of the 15th Computer Security Applications Conference (ACSAC'99), Phoenix, AZ USA: IEEE Computer Society Press, 6-10 Dec 1999, pp. 55-64.

[22] Mouratidis, H., Giorgini, P., Manson, G.: Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. In Proceedings of the 15th Conference on Advanced Information Systems Engineering (CAiSE'03), Klagenfurt/Velden Austria: Springer-Verlag, 16-20 Jun 2003, pp. 63-78.

[23] NIST: An Introduction to Computer Security: The NIST Handbook. Special Pub SP 800-12, National Institute of Standards and Technology (NIST), Oct 1995.

[24] Nuseibeh, B.: Weaving Together Requirements and Architectures. Computer (IEEE), 34(3) (Mar 2001), 115-117.

[25] Pfleeger, C.P., Pfleeger, S.L.: Security in Computing. Prentice Hall, 2002.

[26] Rushby, J.: Security Requirements Specifications: How and What? In Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS), Indianapolis, IN USA, 5-6 Mar 2001.

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

[28] Spafford, E.H.: The internet worm program: an analysis. ACM SIGCOMM Computer Communication Review, 19(1) (Jan 1989), 17-57.

[29] Tettero, O., Out, D.J., Franken, H.M., Schot, J.: Information security embedded in the design of telematics systems. Computers and Security, 16(2) (1997), 145-164.