Arguing Security: Validating Security Requirements Using Structured Argumentation

Charles B. Haley Jonathan D. Moffett Robin Laney Bashar Nuseibeh
Department of Computing
The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
{C.B.Haley, J.Moffett, R.C.Laney, B.Nuseibeh} [at] open.ac.uk

Abstract

This paper proposes using both formal and structured informal arguments to show that an eventual realized system can satisfy its security requirements. These arguments, called 'satisfaction arguments', consist of two parts: a formal argument based upon claims about domain properties, and a set of informal arguments that justify the claims. Building on our earlier work on trust assumptions and security requirements, we show how using satisfaction arguments assists in clarifying how a system satisfies its security requirements, in the process identifying those properties of domains that are critical to the requirements.

1. Introduction

Like all requirements, security requirements benefit from early understanding of their completeness and impact on a system, and from validation that the system can respect the requirements. In previous work [16], we (and others – e.g. [13]) have argued that security requirements may usefully be described as *constraints on the functions of a system*, converting them from quality requirements to functional requirements. We have proposed that one begins by eliciting security goals for assets that are implicated in the system. Next, for each function of the system, the analyst determines which assets are involved in that function. The analyst then determines the security requirement(s) to apply to that function in order to satisfy the goal(s).

Key validation steps for such a process are the abilities to show that 1) the proposed security goals adequately express what the stakeholders need, 2) the proposed security requirements adequately satisfy the goals, and 3) the system can satisfy the security requirements.

The contribution of this paper is the use of structured informal and formal argumentation for the third validation step: to convince a reader that a system can satisfy the security requirements laid upon it. The paper proposes a two-part argument structure for security requirement *satisfaction arguments*. The first part is a formal argument to prove a system can satisfy its security requirements, using claims about system behavior, assuming that these claims are true. The second part consists of structured informal arguments to support the claims (and the assumptions behind the claims) made when constructing the formal argument. Building on our earlier work on trust assumptions [8] and security requirements, we show how two-step satisfaction arguments assist with determining security-relevant domain properties, and how inconsistent and implausible assumptions about them affect the security of a system.

The paper is structured as follows. In Section 2 we elaborate on the motivation for our work, drawing upon related contributions from the areas of design rationale, safety cases, and domain analysis. Section 3 summarizes our approach to problem analysis and introduces the two arguments upon which we base the work presented in this paper. Section 4 shows how these arguments are used to construct satisfaction arguments when reasoning from security constraints to problem context diagrams. Section 5 presents a discussion and evaluation. Finally, Section 6 concludes and discusses directions for future work.

2. Motivation and Background

Our work is related to, and builds upon, research on design rationale and argument capture, on safety requirements analysis, and more generally on ideas behind problem domain analysis [10, 25].

2.1 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 [15]. For example, Buckingham Shum [3] focuses on how rationale (argument) is visualized, especially in collaborative environments. Potts and Bruns [21], and later Burge and Brown [4] discuss capturing how decisions were made, which decisions were rejected, and the reasons behind these actions. Mylopoulos et al [18] present a way to represent formally knowledge that was captured in some way, without focusing on the outcome of any decisions. Ramesh and Dhar [22] describe a system for "capturing history in the upstream part of the life cycle." Fischer et al [7] suggest that the explicit process of argumentation can itself feed into and benefit design. Finkelstein and Fuks [6] 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.

When analyzing security requirements, the ultimate goal is to *convince a reader* that the security requirements can be satisfied, and that nothing is omitted that could result in the requirements not being satisfied. The process used is relevant only as it relates to completeness. Optimality is not part of the argument. Of course, we are not saying that there is no use in having the history that lead to the final arguments; this history will certainly be useful if the arguments fail to convince, or if the situation changes.

2.2 Safety cases

Kelly [12] argues that "a safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context." He goes on to show the importance of the distinction between *argument* and *evidence*. An argument calls upon appropriate evidence to convince a reader that the argument holds. Attwood and Kelly use the same principles in [2], where they take the position that argument forms a bridge between requirements and specification, permitting capture of sufficient information to realize rich traceability.

A similar situation exists with regard to security requirements. Combining the two ideas, argument for safety cases and using arguments for traceability, we paraphrase Kelly's quote presented above as: "a security satisfaction argument should communicate a clear, comprehensive and defensible argument that a system is secure enough to operate in its context."

The techniques proposed by Kelly are not directly applicable to security without modification, primarily because the techniques are focused around objective evidence, component failure, and accident, rather than subjective reasoning, subversion, and malicious intent.

2.3 Problem domain analysis

Zave and Jackson in [25], and Jackson in [10], argue that one should construct a *correctness argument* for a system, where the argument is based on known and desired properties of the *domains* involved in the *problem*. To quote Jackson, "Your [correctness] argument must convince yourself and your customer that your proposed machine will ensure that the requirement is satisfied in the problem domain." This position is the same as Kelly's, with the proviso that Kelly's arguments focus equally on all domains, with no special emphasis on the machine.

A similar situation exists with regard to security requirements. Two significant distinctions must be made, however. The first is that it is very difficult to talk about correctness when discussing security. One can convince the reader that the proposed system meets the needs, but it is far more difficult to prove that the system is correct. The distinction between convince and prove (or show) is important. It is not possible to prove the negative – that violation of security goals do not exist – but one can be convincing that sufficient outcomes have been addressed. We propose using argumentation to this end: to convince a reader that the security requirements can be satisfied.

3. Argumentation Driven Problem Analysis

This section summarizes our approach to problem analysis, and describes the two kinds of argument.

3.1. Problem Frames

We use an approximation of Jackson's problem frames diagrams [10] to represent the system context for a given system function. We do not attempt to identify a particular problem class, but instead enter phenomena and requirements into a system problem diagram. We take this approach because we are not attempting to analyze the wider development problem. We are instead looking at the interaction between domains from a security perspective, which requires us to determine which domains can trigger or see which phenomena.

Figure 1 presents an example problem diagram, showing, for a simple Human Resources system, the domains involved and the phenomena exchanged between the domains. As noted above, the diagram is not intended to conform to a known problem class, but it does show the requirement (the function), the constrained domain(s), the inputs, and the phenomena shared between the domains: the domains that are involved in the system within which the machine operates to realize the necessary function. The *behavior* of the system is specified by the sequencing and interplay of phenomena between the domains.

The notion of claim is central to the work described in this paper. To ground the idea of 'claim' in Jackson's problem analysis, system requirements are optative

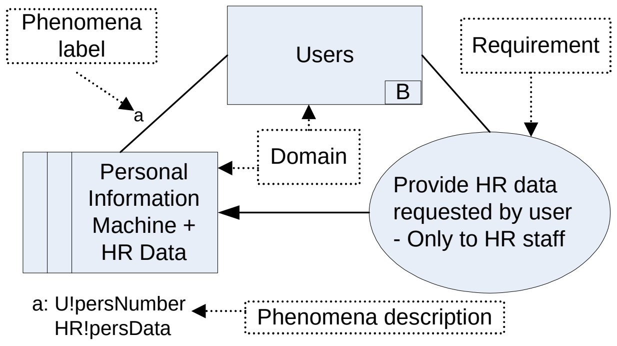
Phenomena label
Users
B
Requirement
a
Domain
Personal Information Machine + HR Data
Provide HR data requested by user - Only to HR staff
a: U!persNumber HR!persData
Phenomena description

Figure 1 – HR data retrieval problem

statements, or statements about what we wish to be true, about the behavior of a system, and therefore are claims about future system behavior that should be argued (and in fact, this is what correctness arguments do). For example, the optative statement "the system shall do X" states a claim that under the conditions described in the problem, the system *will* do X. The correctness argument establishes the validity of this claim.

Indicative statements, or statements that are "objectively true" [10], are used as grounds in a Jackson-style correctness argument. Grounds, "circumstance[s] on which an opinion, inference, argument, statement, or claim is founded,"[1] are used to justify the claims that the optative statements will be true. In the process, one might find that the indicative statements must also be argued, converting them from grounds in an argument to claims made by a sub-argument. The arguments continue recursively, with grounds becoming claims, until the analyst chooses to terminate the recursion.

3.2. Trust Assumptions & Arguments

Our earlier work extended the problem frames approach with *trust assumptions* [8], which are claims about the behavior or the membership of domains included in the system, where the claims are made in order to satisfy a security requirement. These claims represent an analyst's trust that the domains will be as described. Trust assumptions are in the end the analyst's opinion, and therefore assumed to be true. Said another way, trust assumptions are unsubstantiated grounds used in security satisfaction arguments. Because trust assumptions are not argued, they stop the grounds-to-claim recursion.

The trust assumption work has revealed the need to adopt a more structured approach to security satisfaction arguments. We wish to satisfy two goals: 1) that given a collection of domain properties and trust assumptions (accepted as true), one can show that a system is secure, and 2) to create a uniform structure for the satisfaction argument so that the trust assumptions that terminate the recursion are made explicit. We satisfy these goals by splitting the satisfaction argument into two parts: a *formal outer argument* that is first constructed, and *informal structured inner arguments* that are constructed to support the outer argument.

3.2.1 The Outer Argument

The formal outer argument uses claims about the behavior of the system (interplay of phenomena) to demonstrate that the security requirement (the constraint) is satisfied. It is expressed using an appropriate logic, where the premises are formed from domain behavior properties and the conclusion is the satisfaction of the security requirement. We use propositional logic in this paper, resulting in the outer argument being a proof of the form:

```
(domain property premises) ├─ security requirement
```

3.2.2 The Inner Arguments

The inner argument is a set of informal arguments to recursively support the claims used in the outer argument. We propose a form inspired by the work of Toulmin [23], one of the earliest advocates and developers of a formal structure for human reasoning. Toulmin style arguments appear to be well suited for our purpose, since they facilitate the capture of relationships between domain properties (grounds in the formal argument), the trust assumptions that eventually support these grounds, and reasons why the argument may not be valid.

Toulmin et al [24] describe arguments as consisting of:

1. *Claims*, specifying the end point of the argument, or what one wishes to convince the world of.
2. *Grounds*, providing any underlying support for the argument, such as evidence, facts, common knowledge, etc.
3. *Warrants*, connecting and establishing relevancy between the grounds and the claims. A warrant explains how the grounds are related to the claim, not the validity of the grounds themselves.
4. *Backing*, establishing that the warrants are themselves trustworthy. These are, in effect, grounds for believing the warrants.
5. *Modal qualifiers*, establishing within the context of the argument the reliability or strength of the connections between warrants, grounds, and claims.
6. *Rebuttals*, describing what might invalidate any of the grounds, warrants, or backing, thus invalidating the support for the claim.

Toulmin et al summarize the above six items as follows [24; pg 27]: "The *claims* involved in real-life arguments are, accordingly, *well founded* only if sufficient *grounds* of an appropriate and relevant kind can be offered in their support. These grounds must be connected to the claims by reliable, applicable, *warrants*, which are capable in turn of being justified by appeal to sufficient *backing* of

[1] Definition 5C of 'ground' from The Oxford English Dictionary, Second Edition, 1989

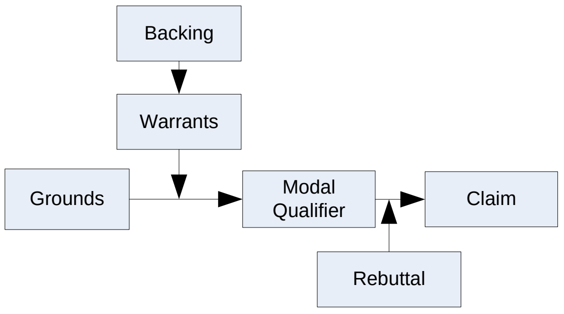

Figure 2 – Generic Toulmin-form argument

the relevant kind. And the entire structure of argument put together out of these elements must be capable of being recognized as having this or that kind and degree of certainty or probability as being dependent for its reliability on the absence of certain particular extraordinary, exceptional, or otherwise *rebutting* circumstances." Toulmin et al propose a diagram for arguments that indicates how the parts fit together. See Figure 2.

Newman and Marshall show in [19] that the 'pure' Toulmin form suffers because the fundamental recursive nature of the argument is obscured. Grounds may need to be argued, making them claims. Warrants may need to be argued, which is the reason for the existence of the backing, but it is not clear how the backing differs from grounds in a normal argument. We agree, and extend Toulmin arguments to make the recursive properties of arguments and the relationships between grounds, warrants, and claims explicit, while keeping the basic connections between the components that Toulmin proposed.

We propose a simple language to represent the structure of these extended Toulmin arguments. This language, expressed in the BNF-like LR(1) grammar[2] shown in Figure 3, captures the essence of Toulmin arguments while facilitating recursion and sub-arguments. Utterances in the language can be seen in Section 4.3.

We chose a textual language because textual utterances are easier to manipulate than tree diagrams, because trees are easily generated from the parser's abstract syntax tree, and because a 'compiler' can assist in dynamic browsing of arguments. Further discussion of the use of the language can be found in Section 5.1.

4. Constructing Satisfaction Arguments

Recall the goal of our work: to construct convincing satisfaction arguments that a system can satisfy its

```
argument      : optional_assignments claim '.'
              | argument optional_assignments claim '.' ;
optional_assignments    : LET assignments ';'
                        | // empty ;
assignments             : assignment
                        | assignments ',' assignment ;
assignment              : IDENTIFIER '=' atom ;
claim : optional_grounds proposition optional_rebuttals;
optional_rebuttals      : REBUTTED BY rebuttals_list
                        | // empty ;
rebuttals_list          : rebuttal
                        | rebuttals_list ',' rebuttal ;
rebuttal      : proposition
              | proposition MITIGATED BY proposition
              | proposition MITIGATED BY '(' claim ')' ;
optional_grounds        : GIVEN GROUNDS grounds_expr
                          optional_warrant THUS CLAIM
                        | // empty ;
optional_warrant        : WARRANTED BY grounds_expr
                        | // empty ;
grounds_expr            : grounds_factor
                        | grounds_expr AND
                          grounds_factor  ;
grounds_factor          : grounds_term
                        | grounds_factor OR
                          grounds_term    ;
grounds_term            : grounds  | NOT grounds ;
grounds                 : proposition | '(' claim ')' ;
proposition             : IDENTIFIER ':' atom
                        | IDENTIFIER | atom  ;
atom                    : STRING   ;
```

Figure 3 – Language Grammar

security requirements. The use of the word "can" instead of the word "will" is important, because we do not know if the eventual system implementation will respect the specifications levied upon it, nor do we know if the system will introduce unintended vulnerabilities.

To construct security satisfaction arguments, one must have security requirements to be satisfied. There are two principal steps involved in determining the security requirements for a system: enumerating the security goals (which assets are to be protected, and why), then determining the security requirements (the constraints) to apply to the system functions to satisfy the goals. These processes are discussed in [16], and are not further elaborated in this paper. This paper is about validating the security requirements, using the collection of domains (including the machine as it will be) to show that the system can respect the security requirements.

A running example is used to illustrate construction of security satisfaction arguments.

4.1. Explanation of the Example

A simple human resources application is used in this section to illustrate our uses of argumentation. We assume one security goal: the data is confidential. One security requirement (constraint) has been derived from this goal: the data must only be provided to HR staff. Figure 1 shows the initial problem diagram for this application. There are two phenomena of interest: the user's request for personnel information (`U!persNumber`) and the information returned by the request (`HR!persData`).

[2] This BNF-like form is the input to a YACC-like [11] parser generator from Bumblebee Software that generates LR parsers in Java and C++. (http://www.bumblebeesoftware.com/)

4.2. The Outer Argument

Starting with the HR problem shown in Figure 1, we first attempt to construct the outer argument that proves the claim: HR data is provided only to HR staff. Recall that this argument will take the form

```
      (domain property premises) |— security requirement
```

There are two domains in the problem: the biddable domain 'users' and the machine (which contains the data). To construct the argument, 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 [17]. In this logic, the behavior of the domains in Figure 1, 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 of the Users domain is not made apparent. No formal argument can be constructed. At this point, we have (at least) three design choices:

1. Introduce some physical restriction, such as a guard, to ensure that the membership of the domain 'users' is restricted to HR staff. Doing so would permit construction of the following proof:
```
M is defined as User ∈ HR
D is defined as phenomenon HR!persData
D → M  (if info is displayed, then user ∈ HR)
D      (info is displayed)
M      (therefore user ∈ 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, even though no information is available to support the assertion.

We choose option 2, and the resulting problem diagram is shown in Figure 4. We now require the user to supply some sort of 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!(UserId, credentials, Payroll#) shall cause
   PIM!Validate(UserId, HR, credentials)
2. if isValid(UserId, credentials)
     PIM!Validate(HR, UserId, credentials)
       shall cause CS!YES
   else
     PIM!Validate(HR, UserId, credentials)
       shall cause CS!NO
3. CS!YES shall cause PIM!PersonInf(Payroll#)
4. CS!NO shall cause PIM!NO
```

The truth or falsity of the isValid predicate 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.

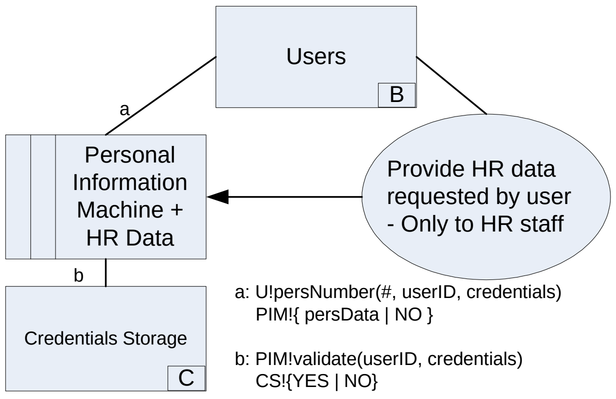
Users
B
a
Personal Information Machine + HR Data
Provide HR data requested by user - Only to HR staff
b
Credentials Storage
C
a: U!persNumber(#, userID, credentials)
PIM!{ persData | NO }
b: PIM!validate(userID, credentials)
CS!{YES | NO}

Figure 4 – New HR staff problem diagram

Symbol	Derived from (see Figure 4)
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)
M: MemberOfHR	Conclusion: user is member of HR

The following predicate 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	$C \rightarrow M$	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 M as the conclusion. Thus we want to show:

(P1, P2, P3, P4, D $\vdash$ M).

A proof is shown in Figure 5.

4.3. The Inner Arguments

Each of the rules used in the outer argument should be examined critically. We begin with the premises P1, P3, & 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 will be correctly implemented";
given grounds G1 thus claim S1.
given grounds G1 thus claim S3.
given grounds G1 thus claim S4.
```

1	$I \rightarrow V$	(Premise)
2	$C \rightarrow M$	(Premise)
3	$Y \rightarrow V \& C$	(Premise)
4	$D \rightarrow Y$	(Premise)
5	D	(Premise)
6	Y	(Detach ($\rightarrow$ elimination), 4, 5)
7	$V \& C$	(Detach, 3, 6)
8	V	(Split (& elimination), 7)
9	C	(Split, 7)
10	M	(Detach, 2, 9)
11	$D \rightarrow M$	(Conclusion, 5)

Figure 5 – Proof: the security requirement is satisfied[3]

Premise P2 is more complex. This premise 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 Figure 6. This argument incorporates 3 trust assumptions: G2, G3, and G4.

The three rebuttals in the argument require some treatment. 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: a dishonest HR member sells credentials. 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 and leaving the employee a very uncomfortable position. 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 email to the HR director.

These functional requirements would then be used as grounds in an argument against the rebuttal R1, shown in Figure 7. C2 would then be 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.

[3] This proof was constructed manually and verified automatically with DC Proof, by Dan Christensen. http://www.dcproof.com/

```
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" .
```

Figure 6 – Argument 1: for premise P2

5. Discussion and Evaluation

In this section we discuss some issues arising from our work, in order to evaluate our approach.

5.1. The Logic Used for the Outer Argument

We have used propositional logic in our example for simplicity. As a side effect, we hid implicit assumptions that ought to be explicit, e.g. the UserId is the same in I and V. For our example to be complete, claims (trust assumptions) should have been added to the inner argument to cover these assumptions. Using predicate or a more powerful logic would have removed this difficulty.

Use of a more powerful, i.e. more fine-grained, logic in the outer argument leads to fewer trust assumptions in the inner argument. On the other hand, more powerful logics are harder to work with.

5.2. The Inner Argument Language

The syntactic and semantic structure of the inner argument language facilitates several kinds of analysis. Because trust assumptions are defined as unsupported grounds, one could easily produce a list of them. One could produce a list of arguments supporting a particular premise in an outer argument, helping verify plausibility. One could negate one or more trust assumptions, seeing which outer arguments no longer hold. One could produce tree diagrams of the arguments, taking closure into account. The naming of grounds, claims, and rebuttals facilitates reuse of arguments.

```
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 probability 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".
```

Figure 7 – Argument against rebuttal R1

5.3. Constructing Inner Arguments

One question that arises is "how does the analyst find rebuttals, grounds, and warrants?" Unfortunately, we cannot propose a recipe. We suggest a method inspired by the how/why questions used in goal-oriented requirements engineering methods (e.g. KAOS [14]). Given a claim, the analyst asks 'why is this claim true?' and 'what happens if it is not true?'

The method we propose is for the analyst first to choose which claim is being argued, 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. The argument is then constructed.

The analyst next asks the question "what can prevent this claim from being true?" The answers are the initial rebuttals. Some of these rebuttals will be challenges of the grounds or warrants; these create the need for sub-arguments where the challenged item is a claim. In other cases, the rebuttal will not be addressed, thereby creating an implicit trust assumption stating that the event(s) described in the rebuttal are not to be considered. A third possibility is to add new grounds to the argument that remove the conditions assumed by the rebuttal.

5.4. Problem vs. Solution Space

A reasonable objection to argumentation as described in this paper 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 [20], 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.

5.5. Goal Hierarchies & Argumentation

The difference between the implicit argument found in a goal hierarchy (e.g. KAOS [14]) and the argumentation proposed in this paper is primarily one of expressiveness. A goal hierarchy has a ready-made structure (it is an 'and/or' tree) whose validity (or invalidity) is immediately apparent. The structure of an argument from domain properties to security requirements is more complex. Because it depends upon the domains and the behavioral specification of the phenomena, it needs an explicitly-crafted outer argument. Finally, the inner arguments make use of grounds that would not normally appear in a goal hierarchy, for example the warrants G3, G4, and G8.

5.6. Security Functional Requirements

Adding functionality to support security requirements creates a traceability problem. This paper provided two situations where this sort of functionality was added: addition of credential verification to permit the outer argument to be constructed, and addition of monitoring and logging functionality to support removal of the dishonest employee rebuttal. Somehow these functions must remain connected with the security requirement they support, because the need for these functions could change or disappear if the security requirement changes.

6. Conclusions & Future Work

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 either through modification of the problem, addition of security functional requirements, or through addition of trust assumptions that explain why the vulnerability can be discounted.

One area that we are actively looking at is tool support for capturing the arguments. The capabilities discussed in Section 5.1 are prime candidates; the 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 [1] for describing and navigating through IBIS-style arguments [5].

Another area of focus is to combine the ideas from this paper together with our earlier work on threat descriptions [9]. The goal is to present a coherent method to elicit security goals through the enumeration of assets and the threats that they are subject to, and then to link the resulting problem context diagrams together to aid consistency checking.

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.

Finally, we continue to investigate industrial case studies in order to test these ideas more thoroughly.

Acknowledgements:

The authors wish to thank Michael Jackson for his continuous support, to Simon Buckingham Shum for many helpful conversations about argumentation, to Clara Mancini for extending the Compendium tool to support problem frames, and to the external reviewers for their helpful comments. The financial support of the Leverhulme Trust is gratefully acknowledged, as is EU support of the E-LeGI project, number IST-002205.

References:

[1] "Compendium Institute." http://www.compendiuminstitute.org/.

[2] K. Attwood, T. Kelly, and J. McDermid, "The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families: Position Paper," Proceedings of the International Workshop on Requirements Reuse in System Family Engineering, Eighth International Conference on Software Reuse. Carlos III University of Madrid, Madrid Spain, 5 Jul 2004, pp. 18-21.

[3] S.J. Buckingham Shum, "The Roots of Computer Supported Argument Visualization," Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making, P. A. Kirschner, et al., Eds. London UK, Springer-Verlag, 2003, pp. 3-24.

[4] J.E. Burge and D.C. Brown, "An Integrated Approach for Software Design Checking Using Design Rationale," Proceedings of the First International Conference on Design Computing and Cognition, J. S. Gero, Ed. Cambridge MA USA, Kluwer Academic Press, 19-21 July 2004, pp. 557-576.

[5] J. Conklin, "Dialog mapping: Reflections on an industrial strength case study," Visualizing argumentation: software tools for collaborative and educational sense-making, P. A. Kirschner, et al., Eds. London UK, Springer-Verlag, 2003, pp. 117 - 136.

[6] A. Finkelstein and H. Fuks, "Multiparty Specification," Proceedings of the 5th International Workshop on Software Specification and Design. Pittsburgh PA USA, 1989, pp. 185-195.

[7] 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., Lawrence Erlbaum and Associates, 1996, pp. 267-293.

[8] C.B. Haley, R.C. Laney, J.D. Moffett, and B. Nuseibeh, "The Effect of Trust Assumptions on the Elaboration of Security Requirements," Proceedings of the 12th International Requirements Engineering Conference (RE'04). Kyoto Japan, IEEE Computer Society Press, 6-10 Sep 2004, pp. 102-111.

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

[10] M. Jackson, Problem Frames, Addison Wesley, 2001.

[11] S.C. Johnson, Yacc - Yet Another Compiler-Compiler, Computer Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, July 1975.

[12] T.P. Kelly, Arguing Safety - A Systematic Approach to Safety Case Management, D.Phil Dissertation, University of York, York, 1999.

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

[14] A. van Lamsweerde, "Goal-oriented Requirements Engineering: A Guided Tour," Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01). Toronto, Canada, IEEE Computer Society Press, 27-31 Aug 2001, pp. 249-263.

[15] J. Lee and K.Y. Lai, "What's in Design Rationale?," Human-Computer Interaction - Special Issue on Design Rationale, vol. 6 no. 3-4, 1991, pp. 251-280.

[16] J.D. Moffett, C.B. Haley, and B. Nuseibeh, Core Security Requirements Artefacts, Technical Report 2004/23, Department of Computing, The Open University, Milton Keynes UK, June 2004.

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

[18] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing Knowledge About Information Systems," ACM Transactions on Information Systems (TOIS), vol. 8 no. 4, October 1990, pp. 325 - 362.

[19] S.E. Newman and C.C. Marshall, Pushing Toulmin Too Far: Learning From an Argument Representation Scheme, Technical Report SSL-92-45, Xerox PARC, Palo Alto CA USA, 1991.

[20] B. Nuseibeh, "Weaving Together Requirements and Architectures," Computer (IEEE), vol. 34 no. 3, Mar 2001, pp. 115-117.

[21] C. Potts and G. Bruns, "Recording the Reasons for Design Decisions," Proceedings of the 10th International Conference on Software Engineering (ICSE'88). Singapore, IEEE Computer Society, 1988, pp. 418-427.

[22] B. Ramesh and V. Dhar, "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," IEEE Transactions on Software Engineering, vol. 18 no. 6, June 1992, pp. 498-510.

[23] S.E. Toulmin, The Uses of Argument, Cambridge: Cambridge University Press, 1958.

[24] S.E. Toulmin, R.D. Rieke, and A. Janik, An Introduction to Reasoning, New York: Macmillan, 1979.

[25] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," Transactions on Software Engineering and Methodology (ACM), vol. 6 no. 1, Jan 1997, pp. 1-30.