

SECURING THE SKIES: IN REQUIREMENTS WE TRUST

- ➔ **Bashar Nuseibeh and Charles B. Haley**, *The Open University*
- ➔ **Craig Foster**, *NATS*

The authors describe their experiences applying a security requirements analysis to an air traffic control project using a framework that offers different forms of structured argumentation. In deploying the framework, they also learned several lessons about security requirements.

We are regularly bombarded with reports of security incidents—confidentiality of information lost, integrity of assets damaged, availability of products and services denied. In response we are quick to identify flaws in our systems, including poor passwords, lack of encryption, flawed access control, and lax organizational procedures. And rightly so—security, after all, is concerned with protecting our systems from harm.

Nonetheless, there appears to be a disproportionate focus on protection—on mechanisms that implement security. Security is as much about understanding the context in which systems operate as it is about the systems themselves. While developing secure software is important, our attention should be directed at the wider systems, of which software is only one part. It is in these larger sociotechnical systems that threats arise and harm can occur.¹

A sociotechnical system comprises hardware, software, and the organizational structures in which these entities function. It also comprises people—the stakeholders—including both developers and users. While computer-based systems are indeed attacked, it is users and their assets that are harmed, whether this harm is physical, financial, informational, or social. Organizations must accordingly look beyond the system to examine what they are trying

to protect, why they are trying to protect it, and the consequences of inadequate protection. The answers to these questions not only inform software design, but more importantly, connect the design to the users, who will either benefit the most or be harmed by those systems.

The collection of activities aimed at answering these questions is termed *security requirements engineering*,^{2,3} and we have developed a framework and associated tools to support these activities.⁴ As part of a feasibility study by the UK's NATS, we used our framework to analyze security concerns about a new technology being considered for integration with existing air-traffic control (ATC) systems. Although NATS understands and fully considers the safety issues raised by potential use of new technologies, the systematic analysis facilitated by our framework exposed hidden assumptions about the behavior of the particular technologies being investigated, revealing potential security problems that should be considered during a future project. Our experience also revealed insights about security requirements and their elicitation.

SECURITY REQUIREMENTS ENGINEERING

Requirements engineering deals with the discovery of stakeholder expectations and their communication to developers responsible for realizing a subset of those

expectations.⁵ There is increasing recognition that RE's focus on understanding stakeholder problems facilitates the design and implementation of effective solutions.

Security requirements pose a set of technical challenges to RE that together make it a distinctive and important area of investigation.⁶ These challenges include the following:

- *Stakeholder identification.* People with malicious intent must be considered.
- *Problem scoping.* The scope of security requirements must include the wider system context, including policies and procedures.
- *Requirements representation.* Security requirements are expressed in terms of prohibition or prevention—what must not happen in a system.
- *Requirements analysis.* Attackers may repeatedly exploit a single flaw in a system's security requirements until it is fixed.

To address these challenges, our framework guides the discovery and articulation of security requirements. The requirements should not be overly general—for example, “the system shall maintain confidentiality of data”—or overly specific, such as “the system shall encrypt all passwords.” Our framework is built around the following premises:

- Security goals aim to protect assets from harm.
- Security goals are operationalized into security requirements. These requirements express constraints on the functional requirements sufficient to protect the assets from harm as well as define a desired quality of service.
- Feasible realizations of security requirements may lead to the addition of secondary security goals, which might manifest themselves as additional functional or security requirements.
- Security satisfaction arguments show that a system can respect the security requirements. Attempting to construct these arguments exposes trust assumptions and oversights within the system that can profoundly affect security.

Although we distinguish security and functional requirements, here for reasons of clarity we avoid the debatable distinction between functional and nonfunctional requirements.⁷


Activities

Our framework comprises a set of activities for moving from functional goals to security satisfaction arguments, as Figure 1 shows.⁴ The activities cover four steps:

1. identifying functional requirements,
2. identifying security goals,
3. identifying security requirements, and
4. constructing security satisfaction arguments.

Here we focus on step 4, where some of the most revealing analysis takes place. This step concentrates on whether the system can satisfy the security requirements by exposing trust assumptions, checking that the system accounts for the risks introduced by these assumptions, and constructing satisfaction arguments to convince a reader that a system can satisfy the security requirements laid upon it.

These arguments are in two parts. The first part, the *outer argument*, is a formal argument that a system can satisfy its security requirements, drawing upon claims about the behavior of domains in a system. The second part, the *inner argument*, consists of structured informal



One reason that an analyst may fail to construct a convincing argument is that there is not enough information available to justify some claim.

arguments to support the claims about system behavior made in the outer argument. This mixture makes it possible to explore the satisfaction of security requirements with varying degrees of rigor.⁸ The most formal arguments constitute a proof, whereas the less formal arguments inevitably rely on trust assumptions that may be challenged and that may require evidence to hold up to such scrutiny.

Iteration

One reason that an analyst may fail to construct a convincing argument is that there is not enough information available to justify some claim. For example, to justify a claim that users are authenticated, something must be exchanged between the user and the rest of the system. Our framework thus assumes that the process is iterative;⁹ when necessary, designers must add more detail to the system context to permit justification of claims. These iterations move from step 4 back to steps 1 or 2, possibly requiring the addition of new function.

The analyst might establish that there is no feasible way to satisfy the security requirement(s). In this case, designers and stakeholders must agree on an acceptable alternative, such as weaker requirements, or simply decide that the system is infeasible.

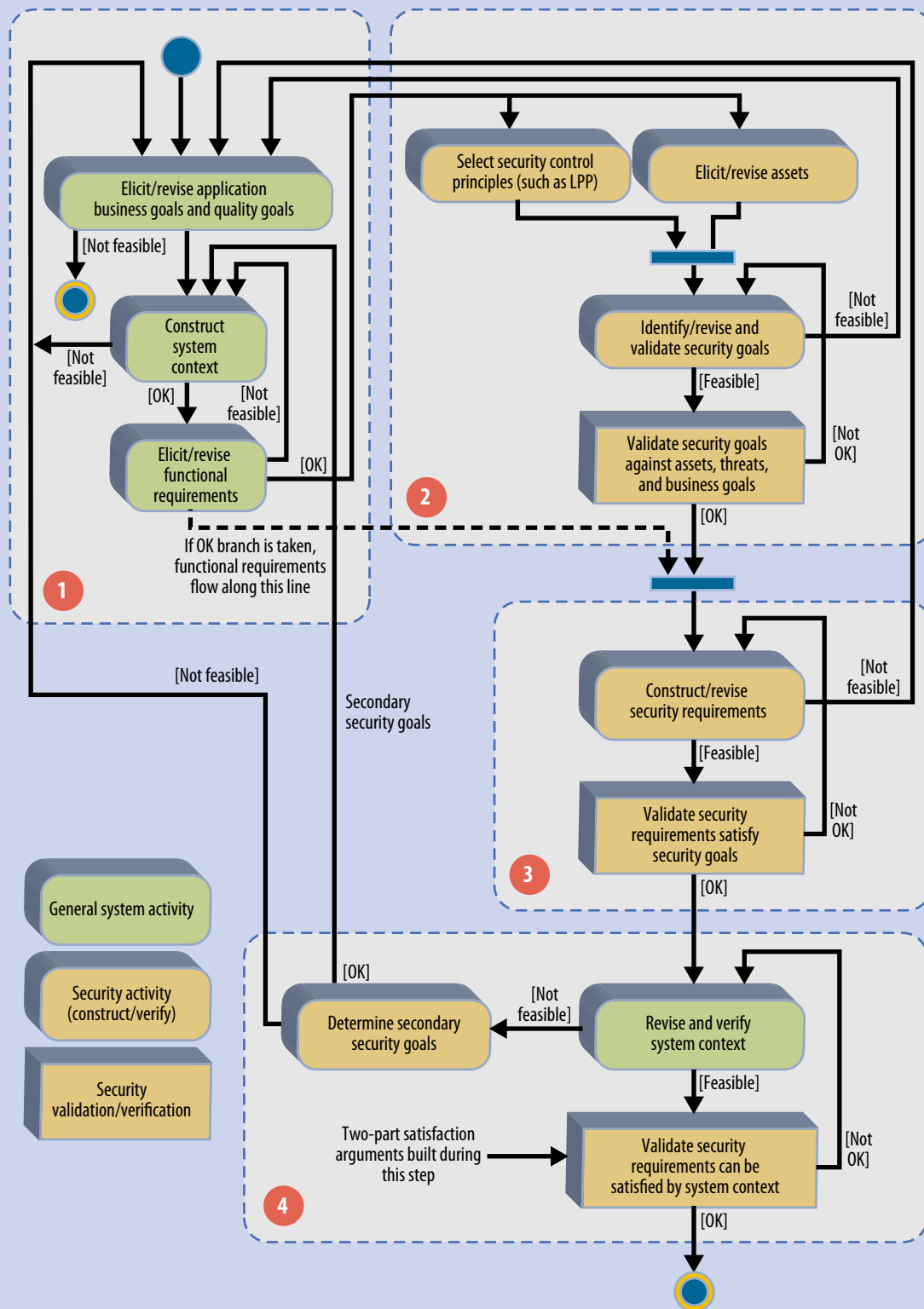


Figure 1. Process activity diagram. Activities cover four steps: (1) identifying functional requirements, (2) identifying security goals, (3) identifying security requirements, and (4) constructing security satisfaction arguments.

AIR TRAFFIC CONTROL PROJECT

We applied our framework to a problem being considered by the CRISTAL UK project, a research initiative managed by NATS for the European Organisation for the Safety of Air Transportation (Eurocontrol) CASCADE program in collaboration with Raytheon Systems Ltd., SITA, and QinetiQ. The project was charged with “determining the role of ‘passive surveillance’ in NATS future surveillance system[s]”—that is, investigating the potential role of emerging surveillance technologies that use GPS for ATC areas where radar is used currently, such as in and around the airspace at busy airports.¹⁰

ATC

ATC 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 safe but also very inefficient. Adding aircraft into the airspace increases efficiency but also increases the risk of a loss of separation. Air traffic controllers try to minimize risk by maintaining safe distances between aircraft. This requires knowing the identity and position of aircraft with a high degree of accuracy, integrity, and assurance.

Controllers maintain safe separation between aircraft while ensuring that they get to their destination efficiently. The minimum separation required between aircraft depends on many factors including the aircrafts’ speed, surveillance accuracy, surveillance system redundancy, and the ability to spot and rectify mistakes.

ATC currently uses active surveillance systems such as radar to determine an aircraft’s position. The project’s primary goal was to examine alternative means for providing this surveillance data that met the same minimum performance requirements. It further examined the impact of improving how often the surveillance system informs a controller where an aircraft actually is and enhancements to the accuracy of this position information.

Active surveillance

Active surveillance determines the position of aircraft independently of the aircraft itself.

Primary radar, which operates by broadcasting directional pulses and listening for reflections, is independent, requiring no specific equipment on the aircraft. It provides only the aircraft’s distance from the radar.

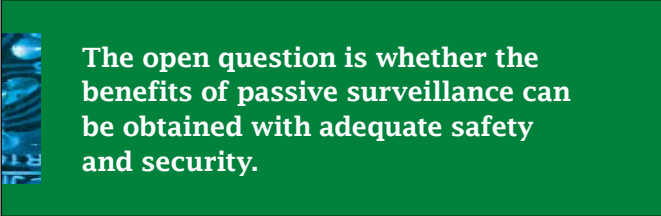
Secondary radar uses highly directional interrogations. It is cooperative in that it expects aircraft to respond to the interrogation in a fixed time. Secondary radar will not see aircraft that do not respond. It calculates an aircraft’s relative distance and bearing in the same way as primary radar, but the response may also contain information such as the aircraft’s identity and altitude.

Passive surveillance

Passive surveillance uses equipment that listens for transmissions from aircraft, then computes the position using that transmission; the surveillance system makes no request of the aircraft for transmission.

Automatic Dependent Surveillance—Broadcast uses on-board satellite navigation technology to determine where the aircraft is, and then broadcasts that position without the need for pilot input or radar interrogation. A surveillance system listening to ADS-B broadcasts depends on aircraft broadcasting accurate positions; it will misplace an aircraft that maliciously or through equipment failure reports an incorrect position.

A second technique, *multilateration*, uses a network of multiple receivers to determine the intersection of the hyperboloids described by the difference in arrival time of the transmission at each receiver—this is used to determine the transmitter’s position. Like secondary radar, position computation depends solely on the timing of the receipt of signals.



The open question is whether the benefits of passive surveillance can be obtained with adequate safety and security.

Passive surveillance has become more attractive to ATC service providers in recent years because aircraft are increasingly being equipped with suitable avionics. In addition to the technologies’ perceived operational benefits, they offer potentially significant cost savings in procurement and through-life maintenance costs over traditional surveillance means. The open question is whether the benefits can be obtained with adequate safety and security.

EXAMPLE SECURITY REQUIREMENTS ANALYSIS

A key issue is whether ADS-B position reports should be used as a primary position source for aircraft. We analyzed the security implications of this question using our framework, stepping through the activities shown in Figure 1. Although our analysis included three iterations, we discuss only one here.

Step 1

We began by identifying the functional goal(s) of the system under analysis, describing the system context, and identifying the functional requirement(s). Three project characteristics dramatically simplified this task.

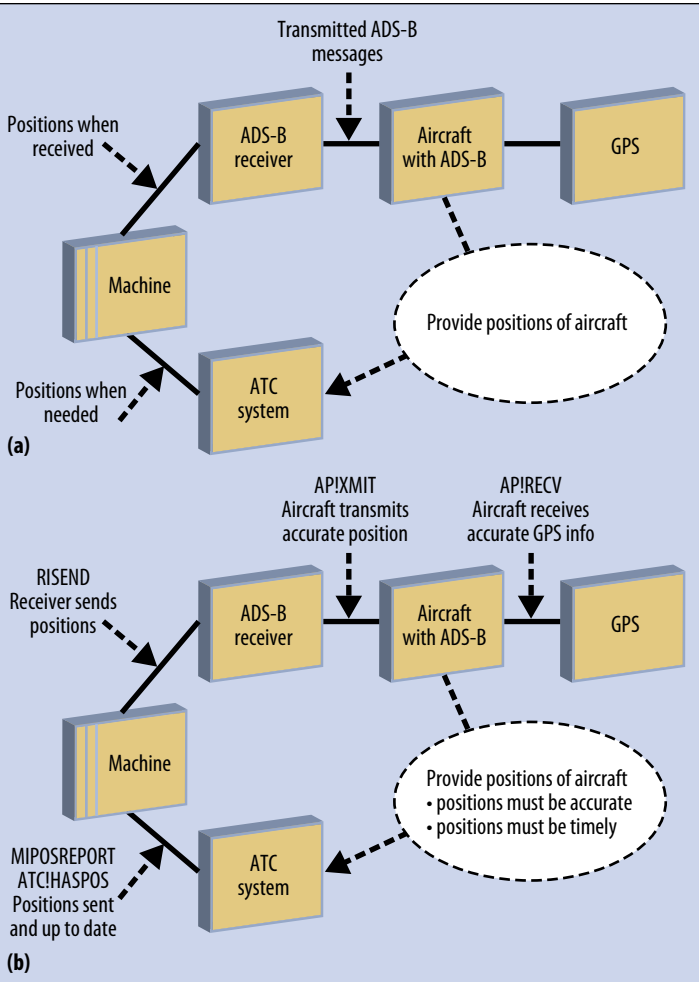


Figure 2. System context: (a) Iteration 1 and (b) with constrained functional requirement.

First, project partners supplied working ADS-B equipment. We treated this equipment as given, simplifying the requirements.

Second, the initial functional goal was given:

FG1: Provide safe and efficient air traffic management.

Third, given this goal and the project’s remit, we were able to summarize the functional requirement as

FR1: Provide positions of aircraft.

The remaining task was to determine the system context, which is shown in Figure 2a. We describe contexts using problem diagrams derived from Problem Frames,¹¹ where a dashed oval denotes the requirements of a machine, which in turn is denoted by a rectangle with two vertical lines on the side. Rectangles indicate domains in the system context. Connecting lines denote shared phe-

nomena such as events between domains (dashed if connected to a requirement), while a dashed line with a solid arrow indicates a requirement’s constraining effect on a domain. The dashed lines with outline arrowheads are our own annotation to problem diagrams, and they mark the names of the shared phenomena.

Step 2

We next determined the assets involved with the system, the harms, and the security goals to avoid those harms. The direct assets are the GPS receivers and signals, aircraft, aircraft positions (broadcast), ground receivers, and the ATC system including the controllers. The indirect assets are the passengers and other aircraft contents; items around the ATC area such as buildings, infrastructure, and the airport; and the aircraft owner’s business—reputation, profitability, and so on.

Using this list of assets, we determined with the help of the project’s domain experts the harms involved in the system, and then identified the threat descriptions in the form “violation of general security goal” on “asset” can cause “harm.” Threat descriptions include the following:

General goal—confidentiality:

- T1: {publicizing, airplanes’ position, facilitating attack in air}
- T2: {publicizing, airplanes’ position, loss of trade secrets}

We decided that threats T1 and T2 were outside the project’s remit and thus did not consider them further.

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 or crash}
- T7: {~available, airplanes’ position, lost revenue due to increased separation}
- T8: {~available, airplanes’ position, lost revenue due to lost confidence}

We determined the system's security goals by avoiding the action in the threat descriptions:

- SG1: Have correct positions (avoids T3, T4, and T5)
- SG2: Report positions as often as needed (avoids T6, T7, T8)

Step 3

In this step we determined the constraints to place on the functional requirements, which in this case was FR1. NATS requires aircraft positions to be highly accurate and timely, with the specific requirements depending on the application. ADS-B can potentially improve upon both of these aspects by an order of magnitude, and the consequences must be studied.

We composed the security goals and the functional requirement, resulting in a constrained functional requirement. The composition produced two security requirements (constraints), the first of which was:

SR1: [FR1: Provide positions of aircraft]: positions shall be accurate. SR1 operationalizes SG1.

The second requirement was:

SR2: [FR1: Provide positions of aircraft]: positions shall be timely. SR2 operationalizes SG2.

Figure 2b shows the system context with the constrained functional requirement.

Step 4

During this step we tested our context, checking if the system could satisfy the security requirements given our assumptions about behavior. By constructing the formal outer argument, we determined which behavior assumptions were important to security. To build the outer argument, we annotated the context with the events exchanged between domains, developed a simplified behavioral specification for the system, and then used the events and behavioral specification as premises in a proof that the system could satisfy the security requirements. The word "could" is important; the proof assumes that any implementation will behave as assumed. We recognize this is a large caveat, but it does not negate the usefulness of understanding what is being secured and why.

Outer argument. We first construct a proof that if certain conditions hold, the system could meet its security requirements. To do this we model the events in the system and the system's behavior, then construct the proof.

As Figure 2b shows, the events exchanged within the system, which use the naming convention "sending domain!message," are:

API!RECV: The aircraft receives GPS broadcasts.

API!XMIT: The aircraft transmits its position.

R!SEND: The receiver sends the 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.



By constructing the formal outer argument, we determined which behavior assumptions were important to security.

We built the behavioral specification using a variant of a causal logic:¹²

API!RECV shall cause API!XMIT

API!XMIT shall cause R!SEND

R!SEND shall cause M!POSREPORT

M!POSREPORT shall cause ATC!HASPOS

We recognized that reception of GPS signals by an aircraft does not in fact cause it to transmit position reports but instead enables them. We chose to accept this slight misstatement instead of adding the complexity of time-based events to the proof.

Logically, we wanted to prove that given the behavior, the air traffic controller would have the aircrafts' positions:

API!RECV → ATC!HASPOS

This would prove that the system could satisfy both SR1 (accuracy) and SR2 (timeliness) assuming that the

- events were described correctly,
- behavior specification was correct and no undocumented conflicting behavior existed, and
- implementation would not introduce any conflicting behavior.

We would challenge these assumptions later when building the informal inner arguments.

We constructed the following proof using propositional logic because we felt that this form of proof would be easier to explain to the project participants:

1. API!RECV → API!XMIT (premise: API!RECV shall cause API!XMIT)
2. API!XMIT → R!SEND (premise: API!XMIT shall cause R!SEND)

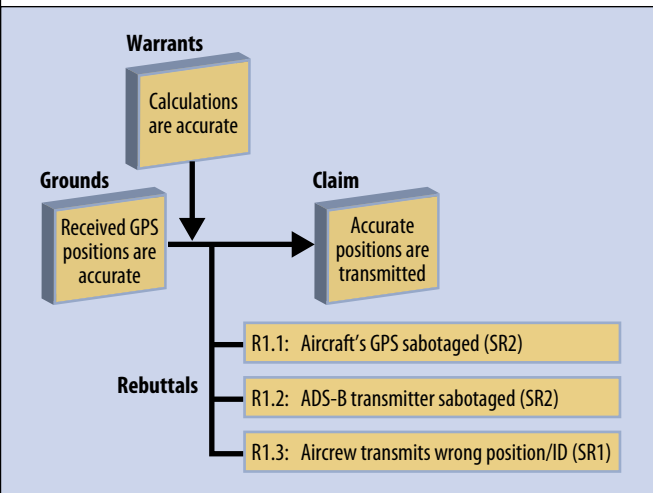


Figure 3. Argument for API!RECV → API!XMIT.

- | | |
|-----------------------------|---|
| 3. RISEND → M!POSREPORT | (premise: RISEND shall cause M!POSREPORT) |
| 4. M!POSREPORT → ATC!HASPOS | (premise: M!POSREPORT shall cause ATC!HASPOS) |
| 5. API!RECV | (assumption) |
| 6. API!XMIT | (detach, 1, 5) |
| 7. RISEND | (detach, 2, 6) |
| 8. M!POSREPORT | (detach, 3, 7) |
| 9. ATC!HASPOS | (detach, 4, 8) |

“Premise” refers to implications derived from the behavior specification, and “detach” refers to the application of logical deduction to the two premises indicated (modus ponens). Further details of the proof’s construction are beyond the scope of this article.

Inner arguments. Steps 1 through 5 of the outer argument must hold for the system to be secure. The purpose of inner arguments is to challenge such assumptions—to establish whether or not they hold in the real world.

We represent inner arguments using a variant of Toulmin’s argumentation diagrams,¹³ although other argumentation representations such as those proposed by T.P. Kelly¹⁴ are available. The general form of a Toulmin-based argument is: *grounds*, justified by *warrants*, lead to *claim*, except when *rebuttals*. The claim is the argument’s conclusion. Grounds are assumptions or evidence used to support the claim. Warrants are connections that justify the use of the grounds, and rebuttals are cases where asserting the claim would not be justified.

We first diagrammed an argument to support one of the claims, then tried to find rebuttals, or reasons why the argument might not hold. Figure 3 shows the argument for the first claim, API!RECV → API!XMIT, along with three

rebuttals. The text in parentheses—for example, SR2—is the security requirement that the rebuttal violates. Other figures, not included here, showed the arguments for premises 2 through 4 (numbers of the lines in the proof) and for the assumption (line 5). In all, there were 12 rebuttals in the arguments, three of which are shown in Figure 3: R1.1, R1.2, and R1.3.

These rebuttals fall into two general categories: sabotage of equipment (R1.1, R1.2), and the intentional transmission of incorrect data (R1.3). We evaluated each one to determine whether it should be mitigated, and if so how. Mitigating a rebuttal requires an iteration within the framework. In this project, we felt that R1.3, aircraft misrepresenting their positions, presented an unacceptable risk of accident or terrorism. We mitigated R1.3 by adding multilateration, a function that computes a transmitter’s position to verify that the aircraft is near to where it says it is. We then subjected the new context to another iteration and found rebuttals to the mitigation; a determined attacker could use specially timed transmissions to confuse multilateration. We considered several mitigations to this new rebuttal.

LESSONS LEARNED

Several lessons about security requirements engineering emerged directly from our experiences in ATC development. They appear to be more widely applicable, although evidence for this is limited to examples of applying our framework on a smaller scale.

Exploit the experts. Identifying security requirements requires a combination of the application of established security engineering practices with deep analysis of the problem domain in which security problems may arise. Our experience suggests that domain knowledge is essential to understanding the subtleties of the security threats and to address their often very technical nature. Indeed, what distinguished our research from similar work⁴ was the addition of such domain expertise to our team.

Exploit the nonexperts. While domain knowledge and expertise are essential for security requirements analysis, experts can often neglect to question assumptions. In fact, we found that it was easy for domain experts to implicitly assume that something behaves in a certain manner because that is how it has always behaved. Domain nonexperts have helped us on projects by asking crucial “why?” questions at unexpected times. Of course, once they asked the questions, a chain of arguments ensued.

Scope the problem. Security problems expand the system context in unexpected ways. For example, the buildings in a city are usually not part of an ATC problem until considering whether an aircraft may fly into one; 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 than that.

FURTHER READING

There is increasing recognition of the importance and role of security requirements. Explicit consideration of security requirements has traditionally fallen in the general areas of risk and threat assessment.^{1,2} The Common Criteria³ mandate the consideration of security requirements explicitly, while recent security methodologies⁴⁻⁶ provide frameworks within which to do so.

Researchers have used abuse cases⁷ and misuse cases⁸ to elicit scenarios describing potentially malicious usage of systems, while other more formal techniques used to check security properties automatically assume a detailed specification is available.⁹ Ross Anderson advocates combining human effort to identify system vulnerabilities.¹⁰

Our work falls within what Jeannette Wing¹¹ calls application-layer security. This includes research examining the social,¹² organizational,^{13,14} and technical¹⁵ subversion of stakeholder goals. John Viega and Gary McGraw first advocated the explicit introduction of trust assumptions during security analysis.¹⁶

References

1. T.R. Peltier, *Information Security Risk Analysis*, 2nd ed., Auerbach, 2005.
2. F. Swiderski and W. Snyder, *Threat Modeling*, Microsoft Press, 2004.
3. Common Criteria sponsoring organizations, "Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Components," v3.1, rev. 2, CCMB-2007-09-003, Sept. 2007, NIST; www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R2.pdf.
4. D.G. Firesmith, "Common Concepts Underlying Safety, Security, and Survivability Engineering," tech. report CMU/SEI-2003-TN-033, Software Eng. Inst., Carnegie Mellon Univ., 2003.
5. N.R. Mead, E.D. Hough, and T.R. Stehney II, "Security Quality Requirements Engineering (SQUARE) Methodology," tech. report CMU/SEI-2005-TR-009, Software Eng. Inst., Carnegie Mellon Univ., 2005.
6. S.T. Redwine Jr., ed., "Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software," v1.05.245, 15 Aug. 2006, Dept. of Homeland Security.
7. J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," *Proc. 15th Ann. Computer Security Applications Conf. (ACSAC 99)*, IEEE CS Press, 1999, pp. 55-64.
8. I. Alexander, "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software*, Jan./Feb. 2003, pp. 58-66.
9. 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 Security (MMM-ACNS 01)*, LNCS 2052, Springer-Verlag, 2001, pp. 84-89.
10. R. Anderson, "How to Cheat at the Lottery (or, Massively Parallel Requirements Engineering)," *Proc. 15th Ann. Computer Security Applications Conf. (ACSAC 99)*, IEEE CS Press, 1999; www.cl.cam.ac.uk/~rja14/lottery/lottery.html.
11. J.M. Wing, "A Symbiotic Relationship between Formal Methods and Security," *Proc. Conf. Computer Security, Dependability, and Assurance: From Needs to Solutions (CSDA 98)*, IEEE CS Press, 1998, pp. 26-38.
12. L. Liu, E. Yu, and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting," *Proc. 11th IEEE Int'l Conf. Requirements Eng. (RE 03)*, IEEE CS Press, 2003, pp. 151-161.
13. R. Crook, D. Ince, and B. Nuseibeh, "On Modelling Access Policies: Relating Roles to Their Organisational Context," *Proc. 13th IEEE Int'l Conf. Requirements Eng. (RE 05)*, IEEE CS Press, 2005, pp. 157-166.
14. P. Giorgini et al., "Modeling Security Requirements through Ownership, Permission and Delegation," *Proc. 13th IEEE Int'l Conf. Requirements Eng. (RE 05)*, IEEE CS Press, 2005, pp. 167-176.
15. R. De Landtsheer and A. van Lamsweerde, "Reasoning about Confidentiality at Requirements Engineering Time," *Proc. 10th European Software Eng. Conf./13th ACM SIGSOFT Int'l Symp. Foundations of Software Eng. (ESEC-FSE 05)*, ACM Press, 2005, pp. 41-49.
16. J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley Professional, 2001.

Issues of problem definition and scoping are well known to RE practitioners and a welcome addition to security analysts.

Iterate to mitigate. Eliciting requirements is often regarded as an activity undertaken early in the development life cycle—not so for a security engineering life cycle. Robust security requirements evolve iteratively as analysts consider mitigations. However, iteration requires careful management to ensure the detection of interactions. We found that the choice to represent mitigations in the context of their rebuttals led naturally to considering them one at a time, when in fact they should be considered together as part of a complete analysis. For example, it makes sense to consider all the jamming scenarios together—for example, ADS-B, clock sync, and

GPS jamming—instead of considering them independently. The composition of local analyses to obtain a more complete analysis remains an open and challenging question, although iteration can sometimes help.

Formalize but argue informally too. Formalization of security requirements and their satisfaction arguments clearly facilitated our analysis. However, we also found that formal (outer) arguments were difficult to construct and explain. One problem was the nature of the proof. The outer argument proves that if the assumptions are valid, the behavior specification is correct, and there are no other behaviors, then the system can be secure. The informal inner arguments help demonstrate the truth of the series of ifs. Security is often as much about being persuaded "beyond reasonable doubt" that a system

is secure than it is about a proof of security, whatever that means.

An important motivator for engaging in security requirements engineering is to discover and articulate security requirements early during development, before substantial investment in design and implementation. Security requirements are often elicited directly from stakeholders, which inevitably increases the likelihood of identifying and protecting their most relevant assets. This is an area of increasingly active research,^{15,16} although some of the more popular approaches, such as misuse cases¹⁷ and threat trees,¹ still assume the existence of “a system” and some known system behavior. The “Further Reading” sidebar highlights classic and recent research on the importance and role of security requirements.

Comprehensive tool support is urgently needed to more easily construct arguments and to better represent and communicate them to people with varied capabilities at handling formality. Of course, articulating clear security requirements and robust satisfaction arguments may be insufficient to achieve the desired level of security. A faulty specification or implementation of the security requirements can introduce security vulnerabilities. Further research is needed to better understand the relationships between security requirements and designs to mitigate against the exploitation of such vulnerabilities. ■

References

1. B. Schneier, *Secrets and Lies: Digital Security in a Networked World*, Wiley, 2000.
2. R. Crook et al., “Security Requirements Engineering: When Anti-Requirements Hit the Fan,” *Proc. 10th Anniversary IEEE Joint Int’l Conf. Requirements Eng.* (RE 02), IEEE CS Press, 2002, pp. 203-205.
3. A.I. Antón, ed., special issue on “Requirements Engineering for Information Security,” *Requirements Eng.*, Dec. 2002, pp. 177-287.
4. C.B. Haley et al., “Security Requirements Engineering: A Framework for Representation and Analysis,” *IEEE Trans. Software Eng.*, Jan. 2008, pp. 133-153.
5. B. Nuseibeh and S. Easterbrook, “Requirements Engineering: A Roadmap,” *Proc. Conf. Future of Software Eng.* (FOSE 00), ACM Press, 2000, pp. 35-46.
6. P.T. Devanbu and S. Stubblebine, “Software Engineering for Security: A Roadmap,” *Proc. Conf. Future of Software Eng.* (FOSE 00), ACM Press, 2000, pp. 227-239.
7. M. Glinz, “On Non-Functional Requirements,” *Proc. 15th IEEE Int’l Requirements Eng. Conf.* (RE 07), IEEE CS Press, 2007, pp. 21-26.
8. D. MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust*, MIT Press, 2001.
9. B. Nuseibeh, “Weaving Together Requirements and Architectures,” *Computer*, Mar. 2001, pp. 115-117.
10. C. Foster and M. Watson, “CRISTAL UK—Final Project Report,” report no. EN-CRISTAL-UK/WP0/FPR/D1.1, 19 Oct. 2007, Eurocontrol; www.eurocontrol.int/cascade/public/site_preferences/display_library_list_public.html.
11. M. Jackson, *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley/ACM Press, 2001.
12. J. Moffett et al., “A Model for a Causal Logic for Requirements Engineering,” *Requirements Eng.*, Mar. 1996, pp. 27-46.
13. S. Toulmin, *The Uses of Argument*, updated ed., Cambridge Univ. Press, 2003.
14. T.P. Kelly, “Arguing Safety—A Systematic Approach to Safety Case Management,” doctoral dissertation, University of York, 1999.
15. A. van Lamsweerde, “Elaborating Security Requirements by Construction of Intentional Anti-Models,” *Proc. 26th Int’l Conf. Software Eng.* (ICSE 04), IEEE CS Press, 2004, pp. 148-157.
16. I.A. Tøndel, M.G. Jaatun, and P.H. Meland, “Security Requirements for the Rest of Us: A Survey,” *IEEE Software*, Jan./Feb. 2008, pp. 20-27.
17. G. Sindre and A.L. Opdahl, “Eliciting Security Requirements by Misuse Cases,” *Proc. 37th Int’l Conf. Technology of Object-Oriented Languages and Systems* (TOOLS-Pacific 00), IEEE CS Press, 2000, pp. 120-131.

Bashar Nuseibeh is a professor of computing at the Open University, Milton Keynes, UK. His research interests are in requirements engineering and design, software process modeling and technology, and technology transfer. Nuseibeh received a PhD in software engineering from Imperial College London. He is an Automated Software Engineering Fellow and a Fellow of the British Computer Society and the Institution of Engineering and Technology. Contact him at b.nuseibeh@open.ac.uk.

Charles B. Haley is a lecturer at the Open University. His research is on the representation of security requirements and their validation through formal and informal argumentation. Haley received a PhD in security requirements from the Open University. Contact him at c.b.haley@open.ac.uk.

Craig Foster is a senior systems engineer in the Communications, Navigation & Surveillance Research Team at NATS, where he manages the portfolio of research projects in the surveillance domain. Foster was project manager of the CRISTAL UK project for the Eurocontrol CASCADE Programme. He received an MSci in mathematics from Imperial College London. Contact him at craig.foster@nats.co.uk.