Arguing Security – Creating Security Assurance Cases

Charles B. Weinstock
Howard F. Lipson
John Goodenough

January 2007

ABSTRACT: An assurance case is a body of evidence organized into an argument demonstrating that some claim about a system holds, i.e., is assured. An assurance case is needed when it is important to show that a system exhibits some complex property such as safety, security, or reliability. In this article, our objective is to explain an approach to documenting an assurance case for system security, i.e., a security assurance case or, more succinctly, a security case.

ACKNOWLEDGEMENTS: Reviews by Sam Redwine, Andy Moore, Ann Miller, Gary McGraw, Nancy Mead, Bob Ellison, and Pamela Curtis are gratefully acknowledged.

INTRODUCTION

A security assurance case[1] is similar to a legal case. It presents arguments showing how a top-level claim (such as "The system is acceptably secure") is supported by objective evidence. Unlike a typical product certification, a security case considers people and processes as well as technology. A case is developed by showing how the top-level claim is supported by subclaims. For example, part of a security assurance case would typically address various sources of security vulnerabilities. Among them, the case would probably claim that a system has none of the common coding defects that lead to security vulnerabilities, including for example buffer overflow vulnerabilities.[2] A subclaim about the

[1] Assurance cases were originally used to show that systems satisfied their safety-critical properties. In this usage, they were (and are) called safety cases. The notation and approach used in this article has been used for over a decade in Europe to document why a system is sufficiently safe [Kelly 1998, Kelly 2004]. The application of the concept to reliability was documented in an SAE Standard [SAE 2004]. In this article, we extend the concept to cover system security claims.

[2] Buffer overflows have been exploited by attackers more than any other class of vulnerability. Further information about the common coding defects that lead to security vulnerabilities can be found elsewhere on the BSI web site and in the computer security literature [BSI 2007b, BSI 2007c, BSI 2007d, BSI 2007e, Howard 2005, Lipner 2005, McGraw 2006, Seacord 2006, Voas 1997, and Viega 2001].

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

absence of buffer overflow vulnerabilities could be supported by showing that (1) programmers were trained on how to write code that minimizes the possibility of buffer overflow vulnerabilities; (2) experienced programmers reviewed the code to see if any buffer overflow possibilities existed and found none; (3) a static analysis tool scanned the code and found no problems; and (4) the system and its components were tested with invalid arguments and all such inputs were rejected or properly handled as exceptions.

In this example, the "evidence" would consist of programmer training credentials, the results of the code review, the output of the code scanner, and the results of the invalid-input tests. The "argument" is, "Following best coding practice has value in preventing buffer overflow coding defects. Each of the other methods has value in detecting buffer overflow defects; none of them detected such defects (or these defects were corrected[3]), and so, the existing evidence supports the claim that there are no buffer overflow vulnerabilities."[4] Further information could show that this claim is incorrect. Our confidence in the argument (i.e., in the soundness of the claim) depends on how convincing we find the argument and the evidence. Moreover, if we believe that the consequences of an invalid claim are sufficiently serious, we might require that further evidence or other subclaims be developed. The seriousness of a claim would depend on the potential severity of an attack (e.g., projected economic loss, injury, or death) related to that claim and on the significance of the threat of such an attack. Although an in-depth discussion of the relation of threat and impact to security cases is beyond the scope of this article, a comprehensive security case should include, or at least be developed in the context of, analyses of the threats to a system and the projected impact of successful attacks.

The structure for a partially developed security assurance case focusing on buffer overflow coding defects is shown in Figure 1. The case is presented in a graphical notation called Goal Structuring Notation (GSN) [Kelly 2004].

---

[3] The proper response to the detection of programmer errors is not simply to correct the code, but also to keep a record of the defects found and to use that information to improve the process that created the defect. For example, based on the nature of the defects detected, the training of programmers in best coding practices may have to be improved. One might also search other products for similar defects and remind (or retrain) programmers regarding these defects.

[4] Of course, despite our best efforts, this claim might be invalid; the degree of confidence that we have in the argument supporting any given claim is an assertion about the case itself rather than an assertion about what is claimed, that is, when we say a system is "acceptably" secure or that it meets its security requirements, we provide an argument and evidence in support of these claims. The extent to which the case is convincing (or valid) is determined when the case is reviewed.

The case starts with a claim (in the shape of a rectangle) that "The system is acceptably secure." To the right, a box with two rounded sides, labeled "Acceptably Secure," provides context for the claim. This element of the case provides additional information on what it means for the system to be "acceptably" secure. For example, the referenced document might cite HIPAA requirements as they apply to a particular system, or it might classify the kinds of security breaches that would lead to different levels of loss (laying the basis for an expectation that more effort will be spent to prevent the more significant losses).

Under the top-level claim is a parallelogram labeled "SDLC." This element shows the strategy to be used in developing an argument supporting the top-level claim. Explicitly showing the strategy is optional but provides helpful insight to anyone reviewing the case. In this example, the strategy is to address potential security vulnerabilities arising at the different stages of the software development life cycle (SDLC), namely, requirements, design, implementation (coding), and operation.[5] One source of deficiencies is coding defects, which is the topic of one of the four subclaims. The other subclaims cover requirements, design, and operational deficiencies. (The diamond under a claim indicates that further development—i.e., further expansion—is required to fully elaborate the claim-argument-evidence substructure.) The structure of the argument implies that if these four subclaims are satisfied, then the system is acceptably secure.

The strategy for arguing that there are no coding defects involves addressing actions taken both to prevent and detect possible vulnerabilities caused by coding defects.[6] In Figure 1, only one possible coding defect, buffer overflow, is developed. Three types of evidence are developed to increase our confidence that no buffer overflow vulnerabilities are present. These types of evidence are associated with each of three subclaims. The "Code Scanned" subclaim asserts that static analysis of the code has demonstrated the absence of buffer overflow defects. Below that are the subclaims that the tool definitively reported "No Defects" and also that all warnings reported by the tool were all subsequently

---

5 We omit validation or testing (as a development activity) because these will be included within the security case itself.

6 The strategy might also consider actions taken to mitigate possible vulnerabilities caused by coding defects, although we don't illustrate this in our example. Such actions could include the use of tools and techniques that provide runtime protection against buffer overflow exploits [Plakosh 2006] in the event that some buffer overflow vulnerability was neither prevented nor detected prior to release of the code. Although not illustrated in our example, the strategy might also include a formal methods approach [Chaki 2006].

verified as false alarms (i.e., “Warnings OK”). Below these subclaims are two pieces of evidence. The first is the tool output, which is the result of running the static analysis tool. The second is the resolution of each warning message, showing why each was a false alarm.

This is not a complete exposition of GSN. For instance, two other symbols, not shown in Figure 1, include “justification” and “assumption.” As with the context element, these are used to provide additional information helpful in understanding the claim.

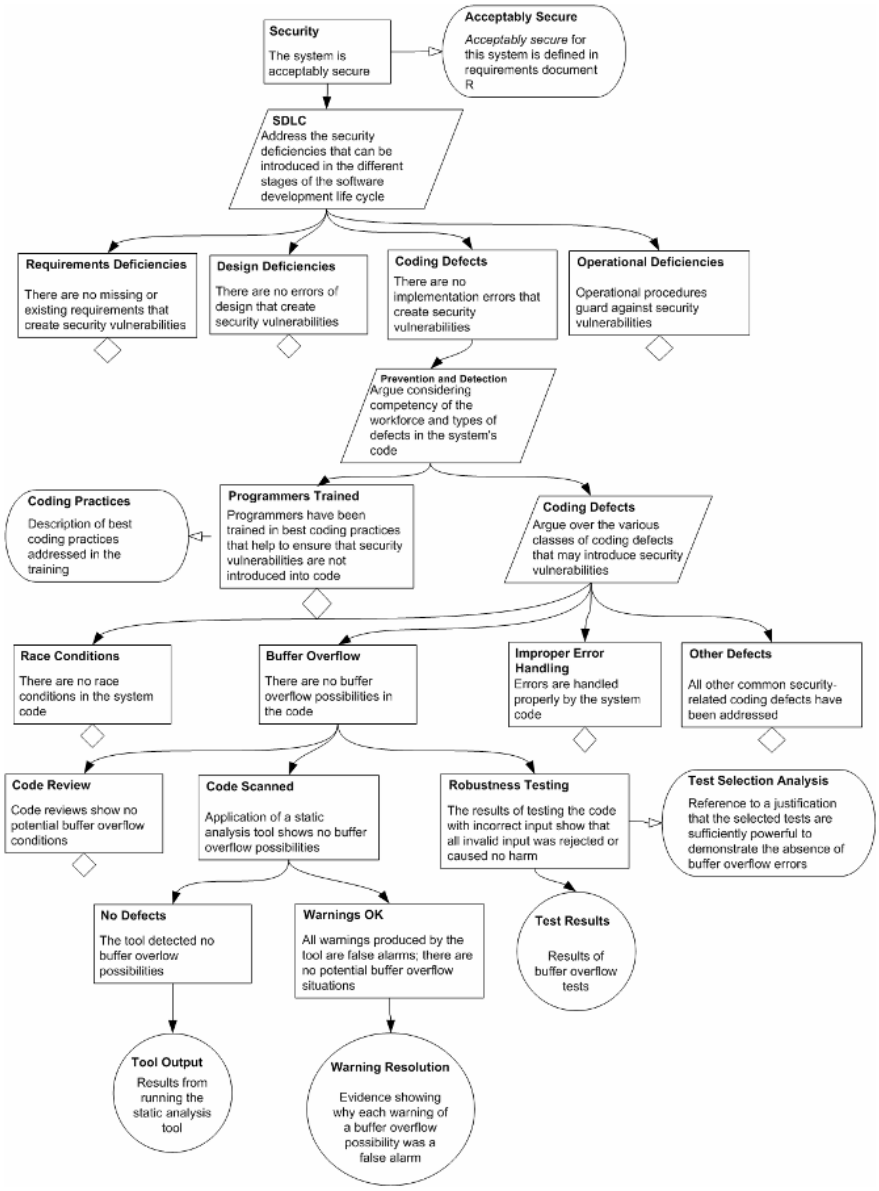


Figure 1. Partially expanded security assurance case that focuses on buffer overflow

The claims in the example are primarily product-focused and technical (i.e., the claims address software engineering issues). An assurance case may also require taking into account legal, regulatory, economic (e.g., insurance), and other non-technical issues [Lipson 2002]. For example, a more complete case might contain claims reflecting the importance of legal or regulatory requirements relating to Sarbanes-Oxley or HIPAA. In addition, an analysis of the threat and consequences of security breaches will affect how much effort is put into developing certain claims or types of argument. If a security breach can lead to a

major regulatory fine, the case may require a higher standard of evidence and argumentation than if a breach has little economic penalty.

Developing a security assurance case is not a trivial matter. In any real system the number of claims involved and the amount of evidence required will be significant. The effort involved is offset by an expected decrease in effort required to find and fix security-related problems at the back end of product development and by a reduced level of security breaches with their attendant costs. Although we believe that the return on investment (ROI) for developing security cases would typically be substantial, work is needed in the community to gather the hard evidence necessary to support this assumption.[7]

Creating and evolving the security case as the system is being developed is highly recommended. Developing even the preliminary outlines of an assurance case as early as possible in the software development life cycle (SDLC) can lead to improvement in the development process by focusing attention on what needs to be assured and what evidence needs to be developed at each subsequent stage of the SDLC. Attempting to gather or generate the necessary security case evidence once development is complete may not only be much more costly, it may be impossible.

In the next section, we present a method for diagramming the overall structure of a security assurance case so it is easier to review for completeness and soundness. We are not so much interested in discussing specific techniques for gathering evidence or making arguments as we are in showing how to document the case so it can be reviewed and evaluated for sound reasoning and in light of the evidence. In a later section, we present the concept of a "security case pattern," which takes advantage of the fact that certain arguments occur again and again when evaluating security claims. We will discuss how to create and use such patterns.

How to Create a Security Assurance Case

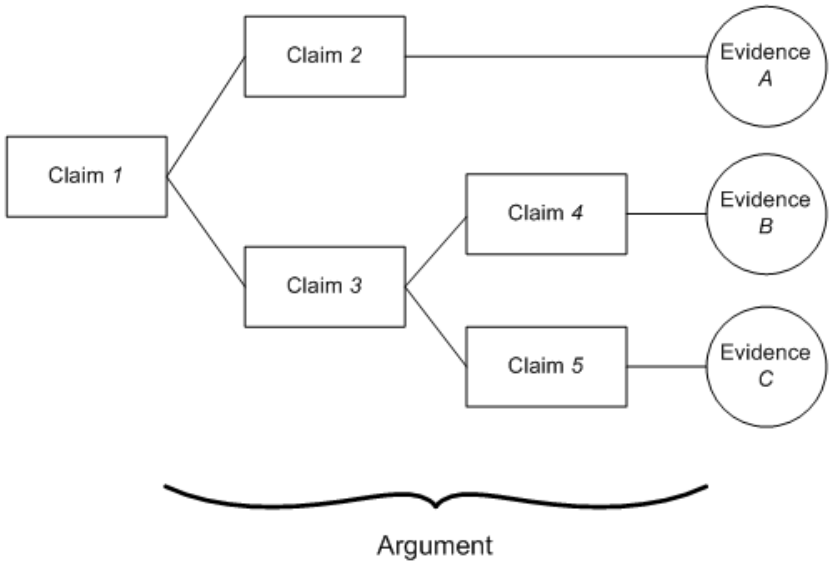
Creating and presenting security assurance cases in a form that facilitates outside review requires some care. While the case can be presented textually, reviewers often find a graphical representation much easier to understand, and we expect that developers and maintainers will find it more useful as well.

---

[7] See the BSI website content areas on Business Case and Measurement for material relevant to making ROI arguments [BSI 2007a, BSI 2007f].

Argument Structure

A security assurance case consists of a structured collection of security-related claims, arguments, and evidence. A claim embodies what is to be shown; an argument tells why to believe a claim has been met, based upon subclaims and evidence such as results of tests, simulations, analysis, etc. Reviewers must be able to understand a security case, so how the evidence supports a claim needs to be clear. Having just a single top-level claim (e.g., "The system does what it's supposed to do") and supporting evidence, without knowing specifically how the evidence relates to the claims, is not appropriate. Instead of requiring a giant leap from top-level claim to evidence, a security assurance case breaks claims into subclaims, each of which is broken into yet another level of subclaims (as in Figure 1) until the step to the actual evidence that supports that subclaim is reasonably small. This structure is shown in Figure 2.



if *Evidence A* & *Evidence B* & *Evidence C* then *Claim 1*
"Argument":
 if *Evidence A* then *Claim 2*
 if *Evidence B* then *Claim 4*
if *Evidence C* then *Claim 5*
 if *Claim 4* & *Claim 5* then *Claim 3*
if *Claim 2* & *Claim 3* then *Claim 1*

Figure 2. The "claim-argument-evidence" structure of an assurance case

Creating a Structured Argument

Claims

Creating a structured argument is a relatively straightforward process, best accomplished in a top-down manner. The process starts by identifying top-level

claims to be made. These top-level claims should provide the "take-home" message for the reviewers and should convince them (if supported) that the required attributes have been satisfied. In general, claims should be straightforward statements that do not consider implementation details. "System X is acceptably secure" might be a good top-level claim while "System X is acceptably secure through the use of a virtual private network" is not. The implementation details should be part of the support of the claim, not part of the claim itself. That said, it is also important that the top-level claim not be over-simplified. "System X is secure" is not as useful a top-level claim as "System X is acceptably secure." The latter claim allows for a richer and more realistic argument structure involving cost/benefit analyses and risk-management tradeoffs. Getting the top-level claim right is important. It is the seed from which the arguments can develop, and if it doesn't contain the right concepts or includes too-specific details its usefulness may be limited.

The following are examples of claims that are properly worded:

- The tool detected no defects.
- The system is acceptably secure.

The following are examples of claims that are poorly worded:

- Hazard Log for System Y (describes an entity)
- Run static analysis tool Z (an action, not a statement).

A claim is properly worded if it is a predicate, i.e., a statement that is either true or false.

Establishing Context

The text that states each claim must be succinct and unambiguous if the argument is to be reviewable. Yet it is often necessary to provide additional information that is not directly part of the argument. For instance the claim may contain terms that are not generally known, or it may refer to a standard or a requirements document, etc. Context is used to provide this additional information. For example: "Context: Acceptably secure for this system is defined in requirements document R." The context is not a part of the actual argument. Justification and assumption cues are used similarly: to provide additional information to make the argument more understandable.

Identifying Strategy

As defined above, the strategy is an additional cue that helps the reader understand the form that an argument is going to take. Instead of being true or false statements, as the claims and subclaims are, the strategy provides

information on how to substantiate the stated claim. The strategy can take many forms, but it is often simply a matter of presentation. The example in Figure 1 shows strategies (i.e., the "SDLC" and "Prevention and Detection" parallelograms) that describe how a claim is to be supported, typically by subclaims that cover all relevant possibilities, each of which is likely to be easier to deal with than the overall claim.

The strategy can be explicit (as in Figure 1) or implicit (as it would be if we hadn't added any strategies to that figure). Sometimes the strategy is obvious from the layout of the subclaims, in which case an implicit strategy is fine. However, if the relationship between a claim and a set of subclaims has any complexity, or if the strategy being used requires additional context, it should be explicit.

Strategies should not contain claims. They should be phrased with respect to the argument, not with respect to the design, testing, or analysis approach. Thus it would be wrong to say "Use Byzantine Agreement protocol." Rather the strategy should be "Argument by appeal to the Byzantine Agreement protocol." Technically, removing a strategy cue does not affect the argument being made, but can affect ease and correctness of understanding. As with claims, it is sometimes helpful to link context, justification, or assumptions to the strategy.

Elaborating the Strategy

A strategy is elaborated by providing a series of subclaims that fulfill the selected approach. For example, for a strategy ranging over all subsystems, claims must be made that cover each individual subsystem. For a strategy involving quantitative results there must be quantitative claims. Elaborating these claims is exactly the same as elaborating the top-level claim. If a good strategy has been chosen and the basis for the strategy is clear, this can be very straightforward.

Evidence

Eventually, as the expansion continues, there will come a point where a claim needs no further refinement and can be directly supported by evidence. For instance, see the "evidence circles" at the bottom of Figure 1, which support the subclaims directly above them. The evidence elements show the results of running a static analysis tool, the results of reviewing any warnings produced by the tool, and the results of testing the code with invalid input.

One caveat: what may seem obvious (and therefore not requiring refinement) to the creator of an assurance case may not be at all obvious to the typical reader of the case. When in doubt, it is best to err on the side of providing too many steps between a claim and its supporting evidence rather than too few.

Any type of evidence can support security assurance arguments, but clearly some types of evidence are better than others. The BSI article, "Evidence of Assurance: Laying the Foundation for a Credible Security Case," [Lipson 2008] presents examples of the kinds of evidence that may be well suited for use in security cases. Details on the kinds of evidence that could support assurance arguments about security properties are also given throughout the Software Assurance Common Body of Knowledge (SwA CBK) [Redwine 2007] (in particular, see Table 5, "Kinds of Evidence").[8]

It is important to realize that the terse descriptions that appear inside the "evidence circles" in our security assurance case example are simply references to the actual evidence. The GSN tool provides a capability for adding hyperlinks so that a security assurance case can serve as an index to a collection of documents representing relevant evidence. More sophisticated tools could allow the case to serve as, or to be an integral part of, an evidence repository.

Tools and Notation

Although this document has shown the development of a security assurance case using the Goal Structuring Notation, other notations are available (e.g., Adelard Safety Case Development – ASCAD [Adelard 2003]). As long as the structure of the argument is clearly presented, the method of exposition of the case and the tools used to develop the case are at the discretion of the case's creator.[9]

Although an assurance case presented in GSN can be created using any general-purpose graphics editor, direct tool support for GSN is available to ease the process. An example of a commercially-available tool that supports GSN (along with some other relevant notations) is Adelard's Assurance and Safety Case Environment.

The Security Assurance Case Throughout the Life Cycle

For maximum utility, a security assurance case is a document that changes as the system it documents changes. The case takes on a different character as a project moves through its life cycle. In the pre-development stage the case focuses on showing that

---

8 There are also discussions of relevant concepts, principles, and practices (and extensive references to the software assurance literature).

9 One of the earliest prototype tools for creating and documenting security assurance arguments was the Visual Network Rating Methodology, developed at the Naval Research Laboratory [Park 2001].

- the plan for a security case is appropriate for the security requirements of the proposed system,
- the technical proposals are appropriate for achieving the security requirements of the proposed system, and
- it will be possible to demonstrate that security has been achieved during the project.

At development time the security assurance case (which is derived from the pre-development case) is

- updated with the results of all activities that contribute to the security evaluation (including evidence and argumentation) so that, by the time of deployment, the case will be complete, and
- presented at design (and other) reviews and the outcomes included in the case.

Using a configuration control mechanism to manage the security case will ensure its integrity as well as help the case always be relevant to the development status.

Security cases provide a structured framework for evaluating the impact of changes to the system and can help ensure that the changes do not adversely impact security. The case should continue to be maintained after deployment of the system, especially whenever the system is modified. Examining the current case can help determine if modifications will invalidate or change arguments and claims and, if so, will help identify the appropriate parts of the case that need to be updated. Further, if parts of the system prove insecure even in the face of a well-developed case, it is important to understand why this particular chain of evidence-argument-claim reasoning was insufficient.

How to Create Security Case Patterns

The Need for Patterns

The process of developing an assurance case is simplified, somewhat, through the introduction of assurance case patterns. Patterns maintain the structure, but not the specific details, of an argument and therefore can be instantiated in multiple situations as appropriate. By building a catalog of patterns (i.e., templates), one can facilitate the process of assurance case creation and documentation. Assurance case patterns offer the benefits of reuse and repeatability of process, as well as providing some notion of coverage or completeness of the evidence.

Our specific focus is on how to develop and use such patterns for security cases. The salient characteristic of a security case is that it is a structured artifact that is

reviewable by a broad range of stakeholders, not limited to security experts. Security case patterns offer the promise of extending the engineering benefits of a repeatable process to the development of security cases for a wide range of software development projects. Reuse of a common argument structure eases the creation of security cases since each case no longer has to be custom-built. A pattern that has been "battle-tested" through repeated use (especially if shared among multiple organizations) is more likely to be complete in its coverage than a new security case developed from scratch, and so, important variations of evidentiary and argument chains are less likely to be overlooked. A repeatable process is more amenable to oversight, training of personnel, detection of defects, and continuous improvement. Moreover, the return on investment of a repeatable process may be easier to determine.

To fully support reusability, a security case pattern should clearly outline the security claim, the profile of the argument to be made, the types of evidence that support that argument, and possibly some measures or weighting of the value of particular (types of) evidence. The context or conditions under which the particular pattern applies should also be fully specified, as well as how the pattern is instantiated, pitfalls in applying the pattern, and when the pattern should not be used. We would expect to see such well-crafted security case patterns become more commonplace as the community gains experience in creating and evaluating security case patterns and their instantiations. Moreover, security case patterns can serve to embody "best practices" for developing secure systems. For example, the application of static code analysis tools is a best practice that is recommended elsewhere in the BSI website (see Code Analysis and Source Code Analysis). In Figure 1, we included this practice in our example security case. A case that includes a larger number of such best practices would presumably be considered more convincing by knowledgeable reviewers.

Turning Our Previous Example Into a Pattern

A very simple and straightforward way to create a security case pattern is to start with a given security case and then parameterize one or more of the content elements. For example, we can turn Figure 1 into a pattern by documenting where choices can be made. The resulting security case pattern is shown in Figure 3, where we have added two black diamond symbols to identify choices. The "at least 1" connector allows a choice among the argument structures for "Code Review," "Code Scanned," and "Robustness Testing." The pattern indicates that the "Coding Defect" claim can be supported either by code review, static analysis, or robustness testing and that at least one of these alternatives should be selected for each type of coding defect. When the consequences of a type of coding defect are potentially more severe, the developer of the assurance

case might want to improve the credibility of the case by selecting two of the three alternatives, or even all three, as we did in Figure 1. On the other hand, certain types of coding defects may not be detectable by an existing static analysis tool, in which case this alternative would be eliminated from the case. When applying a pattern and eliminating an alternative, it is a good idea to insert some justification explaining the absence of that alternative in the context of your particular circumstances in order to show that you have not simply overlooked a common argument structure or type of evidence that could contribute to your case.

The second use of the black diamond is associated with the claim "Code Scanned." This diamond says that one of the two possibilities must be chosen, since they are mutually exclusive: either the tool produces no warnings, in which case nothing further is to be done, or the tool produces warnings, and further analysis is needed to show that the warnings are benign.

The shaded elements of Figure 3 have been further parameterized with text in braces indicating how a particular claim or piece of evidence is to be instantiated. The parameter "Coding Defect Type X" replaces the specific defect "Buffer Overflow" that appeared in several places in Figure 1.

Using Patterns

Security case patterns are claims-argument-evidence structures that can be reused in many different security cases. The security case method offers the opportunity for security and domain experts to codify security knowledge and mitigation strategies in the form of security case patterns. Such patterns can then be shared among the security community and other stakeholder communities and continually built upon, refined, and improved. We envision a growing repository of security case patterns for a variety of domains and operational contexts that not only would provide greater opportunities for reuse and standardization of assurance arguments, but also (with appropriate information sharing) could allow the security community to associate an historical record of security performance (and return on investment) with particular security case patterns. The historical record of systems built with particular security case patterns can itself contribute to the evidence necessary to make a compelling argument that a system satisfies its desired security properties! However, to safely enable widespread sharing of security case patterns, the importance of understanding and clearly specifying the environmental and operational context within which a security case pattern is valid cannot be overemphasized.

Finally, security case patterns that are proven effective and widely shared can encourage worthwhile improvements in organizations' development processes so that the artifacts needed by the security case patterns are created by the

development processes at the appropriate stages of the software development life cycle.

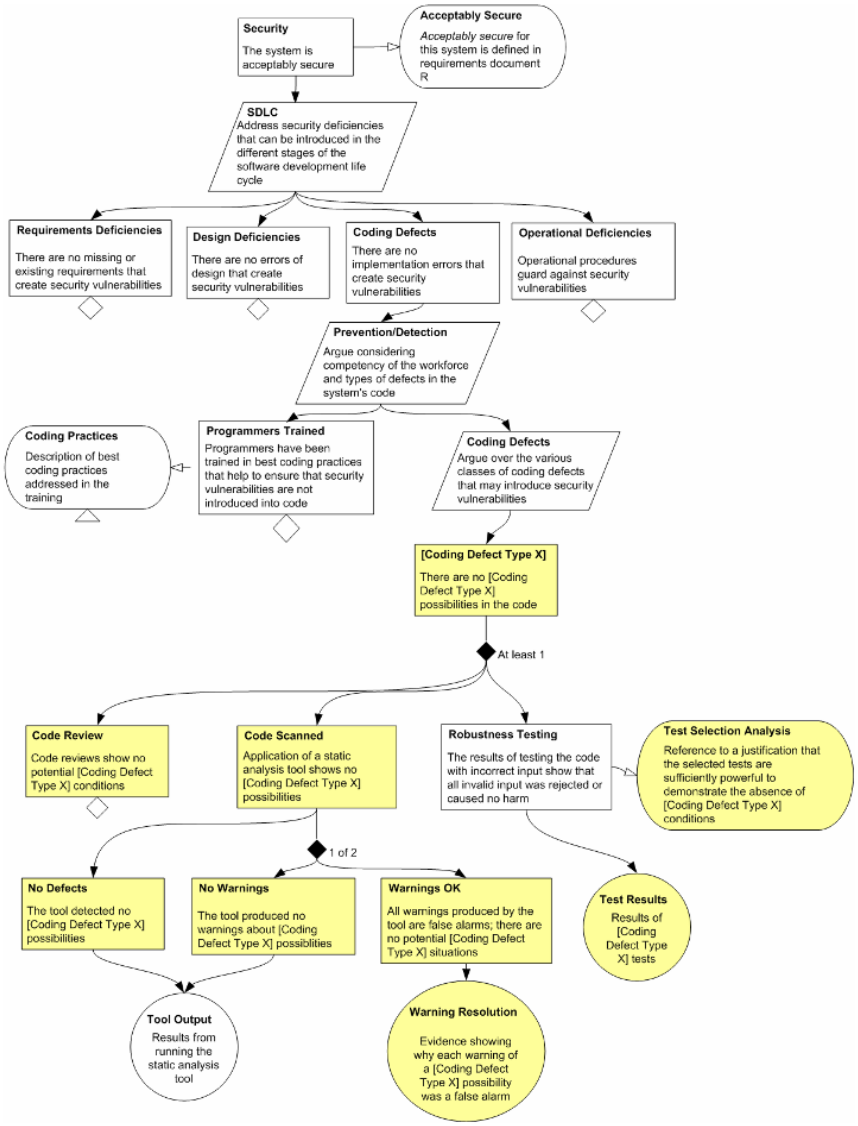


Figure 3. A security case pattern

Related Security Assurance and Compliance Efforts

This section briefly describes how some existing assurance and compliance efforts relate to security cases and patterns.

Security-Privacy Laws and Regulations

Laws and regulations such as Sarbanes-Oxley and HIPAA mandate specific security and privacy requirements. Security assurance cases can be used to argue

that a corporation is in compliance with a given law or regulation. One can envision the development of security case patterns for particular laws or regulations to assist in demonstrating compliance.

Common Criteria

The Common Criteria (CC) [CCMB 2006-7, CCMB 2007] is an internationally recognized standard for evaluating security products and systems. Protection profiles represent sets of security requirements that products can be evaluated and certified against. The results of a CC evaluation include an Evaluation Assurance Level (EAL), which indicates the strength of assurance. Though a CC evaluation has elements that are similar to a security case, the security case is a more general framework, into which the results of CC evaluations can be placed as evidence of assurance.

Anyone creating a product or system meant to satisfy a protection profile needs a way to argue that it in fact meets the profile. Unlike ad hoc approaches to arguing security, the security case method provides an organizing structure and a common "language" that can be used to make assurance arguments about satisfying the set of requirements in a protection profile (at a particular EAL), as well as providing a broader framework that can be used to place CC evaluations in the context of other available evidence of assurance.

The standard format of CC evaluations allows for reuse of some of the basic elements in an assurance argument and hence may be thought of as providing patterns of evaluation. For example, the Common Criteria provides catalogs of standard Security Functional Requirements and Security Assurance Requirements. In contrast, security case patterns allow for the reuse of entire claim-argument-evidence structures and are therefore patterns in a much more general sense. Unlike CC evaluations, a security case is well suited to be maintained over time as a system development artifact, so the assurance case could evolve along with the system, always reflecting the system's current state and configuration.

Community Activities

An international community has begun to work on the topic of assurance cases for security and in the process has formed the International Working Group on Assurance Cases (for Security) [Bloomfield 2006a]. The first organized activity related to this was a workshop entitled "Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities," which was part of the International Conference on Dependable Systems and Networks in Florence, Italy, in June 2004. Its purpose was to promote communication among groups

that were working in the broad area of assurance cases. An important result of the workshop was the decision to take this work further in the security area.

Accordingly, in June 2005, the SEI hosted the Workshop on Assurance Cases for Security in the Washington, DC area. The workshop brought together people working on assuring safety, reliability, and security to envision how assurance cases for security ought to work and how the community might pursue viable technical approaches to realize that vision [Bloomfield 2006b].

In March 2006, the series continued with a workshop entitled "Assurance Cases for Security: Communicating Risks in Infrastructures," which was hosted by the European Commission's Joint Research Centre in Ispra, Italy. The important conclusion from this workshop was the need to support the communication of risks between the different stakeholders involved in critical infrastructures; assurance cases appear to be a viable way of doing this.

A fourth workshop was held in Edinburgh, Scotland on June 27, 2007 at the Dependable Systems and Networks conference. The focus of the workshop was on "Assurance Cases for Security – The Metrics Challenge."

CONCLUSION

Assurance cases for security provide a structured and reviewable set of artifacts that make it possible to demonstrate to interested parties that the system's security requirements have been met to a reasonable degree of certainty.[10] Moreover, the creation of an assurance case can help in the planning and conduct of development. The process of maintaining an assurance case can help to identify new security issues that may arise when changes are made to the system. Developing and maintaining security cases throughout the system development life cycle is an emerging area of best practice for systems with critical security requirements.

A key difference between arguing security and arguing other dependability attributes of a system is the presence of an intelligent adversary. Intelligent adversaries do not follow predictions; rather they try to "attack where you least

---

10 We consider a "reasonable degree of certainty" to mean a "tolerable degree of uncertainty." What is reasonable and tolerable is dependent upon the perceived threat, the consequences of a security breach, and the costs of security measures, including the costs associated with creating and maintaining a security case.

expect." Having an intelligent adversary implies that security threats will evolve and adapt. This means that a security case developed today may have its assumptions unexpectedly violated, or its strength may not be adequate to protect against the attack of tomorrow. This suggests that security assurance cases will need to be revisited more frequently than assurance cases for safety, reliability, or other dependability properties.

In this article, we've introduced the basic concepts associated with security assurance cases and given a detailed example of how to construct a case using Goal Structuring Notation. We then showed how to generalize a portion of a security case into a security case pattern, which can be used as a template to facilitate the process of constructing other security cases.

One should not think of the creation, use, sharing, and evolution of security cases and security case patterns as a method that is in competition with other security certification or evaluation methods, tools, or techniques. Security cases and patterns provide a general framework in which to incorporate and integrate existing and future certification and evaluation methods into a unified argument and evidentiary structure. The security case is particularly valuable as a supporting framework because it allows you to make meta-arguments about the methods, tools, and techniques being used to establish assurance. For example, a security case can argue that a certification method applied by a third-party certifier provides higher assurance than the same method applied by the vendor of the product being certified. This type of meta-argument is outside the scope of the certification method itself.

Though further research and tool development is certainly warranted, organizations can take advantage of the assurance case method right now. There is much to be gained by integrating even rudimentary security cases and security case patterns into the development life cycle for any mission-critical system. Even a basic security case is a far cry above the typical ad hoc arguments and unfounded reassurances in its ability to provide a compelling argument that a desired security property has been built into a system from the outset and has continued to be maintained throughout the system development life cycle.

BIBLIOGRAPHY

[Adelard 2003]

Adelard. *The Adelard Safety Case Development Manual – ASCAD*. Adelard, 2003.

[Bloomfield 2006a]

Bloomfield, Robin E.; Guerra, Sofia; Masera, Marcelo; Miller, Anne; Weinstock, Charles B. "International Working Group on Assurance Cases (for Security)." *IEEE Security & Privacy 4*, 3 (May-June 2006): 66-68.

[Bloomfield 2006b]

Bloomfield, Robin E.; Guerra, Sofia; Masera, Marcelo; Miller, Anne; Saydjari, O. Sami. *Assurance Cases for Security Workshop Report, Version 01c*. Workshop on Assurance Cases for Security, Arlington, VA, June 13-15, 2005 (2006).

[BSI 2007a]

Build Security In. "Business Case Models," 2007.

[BSI 2007b]

Build Security In. "Code Analysis," 2007

[BSI 2007c]

Build Security In. "Coding Practices," 2007.

[BSI 2007d]

Build Security In. "Coding Rules," 2007.

[BSI 2007]

Build Security In. "Measurement," 2007.

[CCMB 2006-7]

Common Criteria Management Board. *Common Criteria for Information Technology Security Evaluation Version 3.1*, *Revision 2*, *Part 1* (CCMB-2006-09-001, September 2006), *Part 2* (CCMB-2007-09-002, September 2007), *Part 3* (CCMB-2007-09-003, September 2007).

[CCMB 2007]

Common Criteria Management Board. *Common Methodology for Information Technology Security Evaluation*, *Version 3.1*, *Revision 2* (CCMB-2007-09-004), September 2007.

[Chaki 2006]

Chaki, Sagar & Hissam, Scott. *Certifying the Absence of Buffer Overflows* (CMU/SEI-2006-TN-030). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.

[Goodenough 2004]

Weinstock, Charles B. & Goodenough, John B. *Dependability Cases* (CMU/SEI-2004-TN-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.

[Howard 2005]

Howard, Michael; LeBlanc, David; & Viega, John. *19 Deadly Sins of Software Security*. Emeryville, CA: McGraw-Hill/Osborne Media, 2005 (ISBN 0-072-26085-8).

[Kelly 1998]

Kelly, Tim P. "Arguing Safety." PhD diss., University of York, 1998.

[Kelly 2004]

Kelly, Tim P. & Weaver, Rob A. "The Goal Structuring Notation –A Safety Argument Notation." *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, July 2004.

[Lipner 2005]

Lipner, Steve & Howard, Michael. *The Trustworthy Computing Security Development Lifecycle* (March 2005).

[Lipson 2002]

Lipson, Howard; Mead, Nancy; & Moore, Andrew. "Can We Ever Build Survivable Systems from COTS Components?" *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE' 02)*. Toronto, Ontario, Canada, May 27-31, 2002. Heidelberg, Germany: Springer-Verlag (LNCS 2348), 2002.

[Lipson 2008]

Lipson, Howard; & Weinstock, Chuck. "Evidence of Assurance: Laying the Foundation for a Credible Security Case." Department of Homeland Security Build Security In Website, May 2008.

[McGraw 2006]

McGraw, Gary. *Software Security: Building Security In*. Boston, MA: Addison-Wesley Professional, 2006 (ISBN 0-321-35670-5), p. 448.

[Moore 1999]

Moore, Andrew P.; Klinker, J. Eric; & Mihelcic, David M. Ch. 13, "How to Construct Formal Arguments that Persuade Certifiers," 285-314. *Industrial-Strength Formal Methods in Practice*. Edited by Michael G. Hinchey & Jonathan P. Bowen. Heidelberg, Germany: Springer-Verlag (FACIT series), 1999.

[Park 2001]

Park, Joon S.; Montrose, Bruce; & Froscher, Judith N. "Tools for Information Security Assurance Arguments," 287-296. *Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX 2001), Volume 1*. Anaheim, California, June 2001.

[Plakosh 2006]

Plakosh, Daniel. "Detection and Recovery." Material excerpted from *Secure Coding in C and C++* [Seacord 2006]. Pearson Education, 2006.

[Redwine 2007]

Redwine, Jr., Samuel T., ed. *Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*. Software Assurance Workforce Education and Training Working Group, U.S. Department of Homeland Security, Draft Version 1.2, October 2007.

[SAE 2004]

SAE. *JA 1002 Software Reliability Program Standard*. Society of Automotive Engineers, January 2004.

[Seacord 2006]

Seacord, R. *Secure Coding in C and C++*. Boston, MA: Addison-Wesley, 2006.

[Viega 2001]

Viega, John. & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley Professional, 2001 (ISBN 0-201-72152-X).

[Voas 1997]

Voas, Jeffrey M. & McGraw, Gary. *Software Fault Injection: Inoculating Programs Against Errors*, 47-48. New York, NY: John Wiley & Sons, 1998 (ISBN 0-471-18381-4), *pg. 416.*

[Weaver 2003]

Weaver, R. A. "The Safety of Software – Constructing and Assuring Arguments." PhD Dissertation, University of York, 2003.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120