Software Engineering Institute | Carnegie Mellon University

Evidence of Assurance: Laying the Foundation for a Credible Security Case

Charles B. Weinstock
Howard F. Lipson

August 2013

ABSTRACT: A security case bears considerable resemblance to a legal case, and demonstrates that security claims about a given system are valid. Persuasive argumentation plays a major role, but the credibility of the arguments and of the security case itself ultimately rests on a foundation of evidence. This article describes and gives examples of several of the kinds of evidence that can contribute to a security case. Our main focus is on how to understand, gather, and generate the kinds of evidence that can build a strong foundation for a credible security case.

ACKNOWLEDGEMENTS: Reviews by Debra Herrmann, Andy Moore, Julian Rrushi, and Melanie Smith are gratefully acknowledged. We also wish to thank Pamela Curtis for her skillful technical editing and Sheila Rosenthal for library services support.

INTRODUCTION

> *"[T]here are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies." (C.A.R. Hoare, 1980 Turing Award Lecture)*

As modern software-intensive systems become more complex and difficult to analyze, there is an increasing tendency to treat them as natural phenomena rather than as artificial constructs that are engineered by humans. Thus we try to assess the security, safety, survivability, or other critical properties of such systems through observation and experiment rather than by direct analysis or an examination of the manner in which the system was constructed. Evaluating the security properties of a system through penetration testing, or by noting the number (or absence) of security-related incidents, or the number and type of vul-

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

nerabilities discovered for a given system over a given period of time[1] can never provide the high assurance needed for mission-critical systems. Nor are observations about the occurrence of security incidents timely enough for critical applications—the system has already been deployed and exposed to whatever damage the reported incident inflicted.[2] An observed absence of security incidents might not be evidence of robust security but instead could be an indication of the inability of the system to detect attacks,[3] or a reflection of the absence of attacks during the period of observation.

While empirical analysis of software-intensive systems has its place and does provide some evidence that a system meets its desired security properties, when used alone it is an incomplete and unsatisfactory approach. Richer sources of evidence can be found elsewhere, particularly through visibility into (and a disciplined examination and "instrumentation" of) the engineering processes used throughout the system development life cycle (SDLC). Generating and gathering evidence that the desired security properties are being "built in" during a system's construction and maintained throughout the system's lifetime of use are crucial steps in demonstrating, with high assurance, that those security properties are present in the current instantiation of the system.

No matter how rich the sources of available evidence are, unstructured "piles" of evidence do not provide assurance—just as piles of evidence alone do not win or lose a legal case. Evidence should be carefully chosen and organized using well-structured arguments that show how the evidence relates to and supports a particular claim. Depending on the claim, some kinds of evidence (and some specific instances of evidence) will be stronger than others. However, even evidence that is relatively weak when considered in isolation may be compelling when combined in argumentation with other evidence. This kind of evidence-based approach has been used for more than a decade in Europe to demonstrate the safety properties of critical systems, and when used for that purpose is known as a safety case, or more generally an assurance case.

[1] Another empirical observation viewed as evidence of security quality is the speed with which vulnerabilities are patched.

[2] Here, early adopters would be a primary testing ground for the security of new system. Of course, the best strategy here is not to be an early adopter.

[3] Such as lack of logging, failure to regularly audit existing logs, or not protecting/isolating logs from erasure or modification by an attacker.

In recent years, there has been increasing interest and activity toward applying the assurance case approach to domains other than safety[4]—and in particular to the security domain. A security assurance case (known more succinctly as a security case) uses a structured set of arguments and a corresponding body of evidence to demonstrate that a system satisfies specific claims with respect to its security properties. Figure 1 is a high-level view of a security case that has a top-level claim called Security Claim 1. The validity of that claim is demonstrated by using arguments that break the top-level claim into subclaims and then using further arguments to ultimately link the top-level claim to a supporting body of evidence. (See the introductory BSI article "Arguing Security – Creating Security Assurance Cases" [Goodenough 2007] for more details on how security cases are created and on the graphical Goal Structuring Notation [Kelly 2004] that is commonly used to present them.)

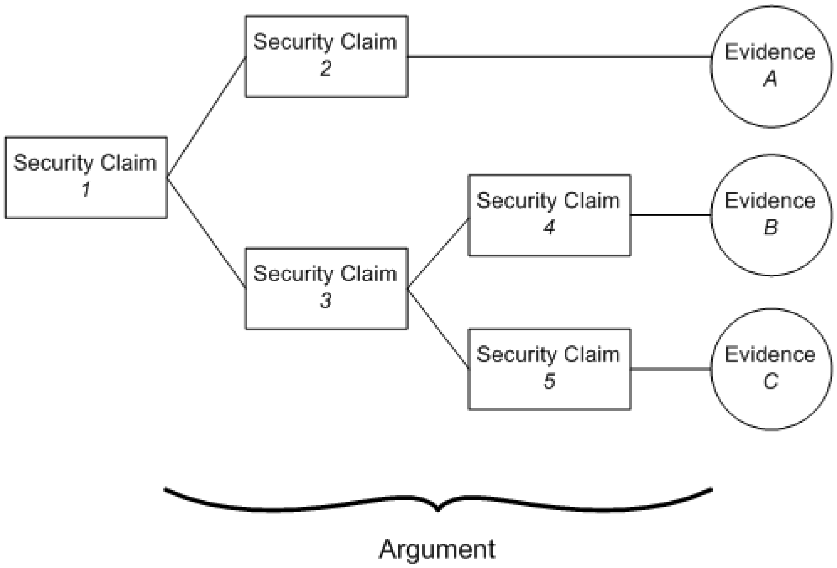

Figure 1. High-level view of a security case

A security case bears considerable resemblance to a legal case, where the "jury" consists of one or more of the stakeholders of the system under scrutiny and may

[4] The application of this approach to the reliability domain is described in the Software Reliability Program Standard issued by the Society of Automotive Engineers [SAE 2004]. The standard defines a framework for managing software reliability, in which the concept of a software reliability case plays a central role.

include third parties (e.g., objective experts, certifiers, or regulators) that represent the interests of some or all of the stakeholders. The stakeholders and/or their representatives review the security case and decide whether the security case is credible.[5] As with a legal case, persuasive argumentation plays a major role, but the credibility of the arguments and of the security case itself ultimately rests on a foundation of evidence.

This article describes and gives examples of several of the kinds of evidence that can contribute to a security case. The quality of evidence is of great importance as well, and we'll provide some discussion along those lines, but a comprehensive treatment of the quality of particular pieces of evidence and the "evidential force" of the combination of various pieces of evidence is an open research area that is beyond the scope of this article. How to make a security case more credible by understanding, gathering, and even generating the kinds of evidence that can contribute to a strong foundation for a security case is the focus of this article.

THE NEED FOR EVIDENCE-BASED ASSURANCE

Dependability is an umbrella term that is meant to encompass a broad range of software quality attributes including reliability, fault tolerance, and security [Avizienis 2000]. A recent National Research Council report [CCDSS 2007] vigorously puts forth the view that claims about high dependability for software systems must be based firmly on evidence and the existence of a credible dependability case.

> *"[D]ifficulty regarding the lack of evidence for system dependability leads to two conclusions, reflected in the committee's findings and recommendations below: (1) that better evidence is needed, so that approaches aimed at improving the dependability of software can be objectively assessed, and (2) that, for now, the pursuit of dependability*

[5] As security cases become more commonplace, similarities between the security case under consideration and previous security cases for known systems with established performance histories may make the decision process easier, in a manner somewhat akin to legal precedent. For example, a security case in which the only substantive difference from a widely accepted security case is a less hostile threat environment should itself be readily accepted. Of course, stakeholders might then ask what cost savings could be garnered by easing off on some of the security controls that may not be needed for a system operating under reduced threat. Analysis of the effects of this tradeoff would require modification of the security case and further review.

> *in software systems should focus on the construction and evaluation of evidence.*
>
> *The committee thus subscribes to the view that software is "guilty until proven innocent," and that the burden of proof falls on the developer to convince the certifier or regulator that the software is dependable. This approach is not novel and is becoming standard in the world of systems safety, in which an explicit safety case (and not merely adherence to good practice) is usually required. Similarly, a software system should be regarded as dependable only if it has a credible dependability case" [Jackson 2007].*

Clearly, this viewpoint confirms the need for a credible security case based on a sound body of evidence to support high-assurance claims of security.

BUILDING IN SECURITY ASSURANCE

Throughout the BSI website we make the point about building in security from the outset rather than attempting the often futile exercise of bolting security on later. Similarly it is difficult, cost-prohibitive, and often impossible to gather sufficient evidence of assurance to support claims of high security for a system that has already been built and deployed or has even reached an advanced stage of design. We believe that a "security case" must be part of any system design from the outset and should be developed alongside the system it supports.

As the initial system design may be highly abstract and incomplete, so will be the security case, and the initial claims and arguments may be fuzzy and incomplete. However, the initial security case will be a framework that gives a preliminary indication of the kinds of evidence that are needed to support the claims. As such, it is a cue to the development team that when the life cycle development processes generate artifacts that can serve as evidence—requirements documents, design rationale, test plans, test results, and results of code reviews—the artifacts should be preserved for use in a security case. If a review of the early security case indicates that an argument is weak and could be bolstered by a piece or pieces of evidence that would typically not be available in due course during development, changes to SDLC processes can be made to explicitly generate the necessary evidence so it will be available to future reviewers of the more mature security case. Moreover, even preliminary versions of the security case may be provided to reviewers, certifiers, or regulators, who may recommend the kinds of supporting evidence they would like to see or would require for future review. Development processes that do not generate the desired forms of evidence may be considered inadequate for the production and certification of high-assurance systems with respect to security.

THE NATURE OF SECURITY AND SURVIVABILITY CLAIMS

Gathering and generating evidence for use in a security case cannot be carried out effectively without a basic understanding of the nature of the claims that the evidence must support, so that only evidence with at least a reasonable (and preferably high) degree of relevance to the claim is considered. Moreover, the quality of a piece of evidence can only be evaluated in light of the claim it is meant to substantiate. For example, all things being equal (that is, absent any specific context), a full fingerprint is considered to be better quality evidence than a partial fingerprint and can support stronger arguments by a prosecutor. However, in support of the claim that John Doe was present at a crime scene, a partial fingerprint at the crime scene matching John Doe with 85% probability is, of course, much stronger evidence than a full fingerprint found 10 miles away even if the full print matches John Doe with 99% probability.

Security claims are of two general types: (1) claims about the requirements-based security properties of a system and (2) claims about the absence of vulnerabilities in the design or implementation that could be exploited to break the system's security model.[6] The first category involves the traditional CIA security properties confidentiality, integrity, and availability, which apply in the general context of the processing, storage, and communication of data and the provision of services. These basic properties are often extended to include three additional properties: authentication (validating user identity), authorization (granting access only to legitimate users), and non-repudiation (ensuring that a party performing a given action will be unable to successfully refute that they were involved). Non-repudiation is sometimes referred to simply as accountability, although all three of the extended properties can be thought of as supporting accountability as well. Since these extended properties can be directly derived as consequences of the basic properties, they are often considered to be implicit in the CIA properties and are often not listed separately.

The second general category of security claim is the absence of vulnerabilities that could be exploited to break the security model, somewhat akin to what a colleague termed the "Indiana Jones attack."[7] In a classic scene from the movie "Raiders of the Lost Ark," the hero, Indiana Jones (Indy), is confronted by a

[6] A security model is an abstract representation of the means by which a system enforces its security requirements within a given threat environment.

[7] Lipson conversation with John McHugh on October 28, 2007, at the ACM Quality of Protection Workshop, Alexandria, VA.

master swordsman menacingly twirling a huge sword in an intimidating demonstration of his skill. It appears that Indy would have little chance of survival against this adversary until, with a look of impatience, Indy just pulls out a gun and shoots him. By doing so, Indy breaks both the security model of his adversary and the movie cliché of the hero always "playing fair" and strictly within the presumed rules of engagement. Vulnerabilities arise when assumptions about the (threat) environment in which a system operates are incorrect or incomplete or when presumed constraints on the behavior of a potential adversary do not reflect reality.

A more comprehensive security-related concept called survivability is a blend of computer security and organizational risk management [Lipson 1999]. It's an umbrella term like dependability, but more sharply focused on security. Survivability is the ability of a system to fulfill its mission, in a timely manner, despite attacks, accident, or subsystem failures. Under stress the system should degrade gracefully, at the very least providing a set of essential services specified in the survivability requirements derived from the system's mission.[8] Claims relevant to survivable systems include the ability to satisfy the 3 Rs, resistance to attack (e.g., security) or other stress, recognition of a survivability problem or crisis and its effect on the system, and recovery, the ability to restore full services after an attack or other crisis. Another type of recovery claim is that in the event essential services are interrupted during a crisis, the system will recover sufficient capability to get those essential services (or an alternate set of comparable services) back up and running quickly enough so that the mission is not jeopardized. (An example would be radar blinking out momentarily but not enough to affect air traffic control.) These kinds of assurance claims are not binary, but rather are always tempered by the notion that the residual risk has been reduced to as low as reasonably practicable (a principle known as "ALARP").[9] A good discussion of ALARP, as well as a pattern for its application, can be found in The Safety of Software – Constructing and Assuring Arguments [Weaver 2003].

[8] Some refer to this general concept as resilience. "An organization that has resilient operations should be able to systematically and transparently cope with disruptive events so that the overall ability of the organization to meet its mission is minimally or not adversely affected" [Caralli 2007].

[9] That is, the cost of further reducing the risk would be prohibitive or disproportionately high when compared to the corresponding benefits. This determination is made in the context of the criticality of the system that is the subject of the security case (i.e., based on the projected consequences of compromise or other system failure).

A FRAMEWORK FOR EVIDENCE OF ASSURANCE OF SECURITY PROPERTIES

The kinds of evidence that can provide assurance about the security properties of a system should be derived not only from the system itself, but also from the entire context in which the system exists. For example, determining whether the security risks to the system's mission have been reduced to as low as reasonably practicable (ALARP) cannot be accomplished by considering (e.g., analyzing or observing) the system's security mechanisms solely in isolation. Of course it is well understood within the security community that, at the very least, evidence about the threat environment within which the system operates must be thoroughly considered.

However, the context in which a system exists is significantly broader than its threat environment alone. Figure 2 shows the basic outline of a framework that we propose for categorizing and organizing evidence of assurance for a system-in-context.

	System Enablers or Threats		
Elements of a System-in-Context	Actors	Processes	Technology
System Owner			
Mission			
Environment			
System			
Life Cycle Capability			

Figure 2. Evidence of assurance framework for a system-in-context

Elements of a System-in-Context

A system-in-context contains these elements:

System Owner – The organization (or individual) that has primary control over the system's construction, operation, and evolution and who determines its mission. Of course, there are large composite systems-of-systems that have no owner, the Internet being a primary example. However, we're primarily interested in creating security cases for business and organizational systems that have a clear owner and a clear mission, even if the system is partially or primarily composed of components and services that are provided by and controlled by others.

Mission – A very high-level abstract description of the mission of the system, from which detailed requirements for security and other quality attributes are derived. The mission should be abstract enough so that, in the event of a crisis, alternate and diverse ways of fulfilling the mission are possible, which allows for a system design that is survivable and resilient even if some portion of the system is compromised by attack or accident. Otherwise security becomes a binary, "all or nothing" condition and leads to a brittle system that will likely fail in the event of an intrusion or compromise of any its components or subsystems.

Environment – This includes the technical, social, political, economic, legal, and regulatory environment in which the system operates. Some salient elements of the environment include•legal mandates and regulation (for security and privacy like SOX, HIPAA)

- security organizations and incident response teams
- colleges and universities with engineering and security programs
- security standards, standards bodies, and trade associations
- commonly accepted certifications, academic credentials, engineering best practices and generally accepted state-of-the-practice (i.e., "due care")
- law enforcement
- international treaties
- From a security standpoint, the threat environment includes
- malicious actors (along with analyses of their motivations)
- attack methodologies and tools
- hacker engineering capabilities
- hacker organizations and communication channels that spread information about vulnerabilities and exploits
- criminal organizations, nation states, and terrorists that provide resources that support the activities of attackers

System – This is the system itself and includes cyber, physical, and human elements. It includes deployed code, components, subsystems, services, hardware, and facilities (e.g., physical plant), whether developed and provided in-house, acquired from vendors, or outsourced. The system also includes operators, policies and procedures (whether manual or automated), staff that use or add value to the cyber services the system provides,[10] and guards for physical security. Any elements or services that are part of the system (cyber, physical, or human) may be outsourced.

Life Cycle Capability – Refers to the ability of the system owner (typically an organization) to construct and maintain the system throughout the system's engineering life cycle. In addition to an organization's in-house engineering capability, this refers to the organization's expertise and risk management evaluation skills with respect to acquisition and outsourcing in a manner that produces a system that meets mission objectives, in particular with respect to the security properties of the system.

When evaluating the security properties of a system that is in the early design phase, artifacts from the life cycle are the primary means of gathering assurance evidence, since there is no running system to test, profile, or analyze. But even after the system is deployed, life cycle evidence can play an important role in system evolution and the ability to react to a changing environment and evolving threats. Typical life cycle phases include the following (an iterative spiral software engineering methodology is implied [Mead 2001]):•Mission Specification

- Threat Analysis
- Concept of Operation (Use and Misuse Scenarios)
- Requirements
- Architecture and Design
- Acquisition
- Implementation and Integration
- Test
- Deployment
- Operations
- Evolution
- Decommissioning

[10] Some users can be classified as part of the system and others as part of the environment. We won't draw that distinction too finely in this article.

Incident and Emergency Response Capability – No modern large-scale, highly distributed system is 100% immune from attack, nor can a system's automated recovery mechanisms handle every possible crisis scenario. An incident response capability is an essential aspect of the survival and resiliency of any mission-critical system, though this capability may rely in whole or part on an external trusted organization, such as a national incident response team or a response team dedicated to a particular industry segment. Note that emergency response (also called disaster planning and recovery) is distinguished from normal incident response because resource demands can be drastically different and may involve outside assistance from government emergency services.

SYSTEM ENABLERS OR THREATS – ACTORS, PROCESSES, AND TECHNOLOGY

Corresponding to each element of a system-in-context are the three classes of contributors to system assurance: Actors (organizations and individuals), Processes, and Technology.[11] Actors, processes, and technology either enable a system to fulfill a mission or can threaten it. Hence, they are the three categories from which evidence of assurance can be generated, gathered, and assessed. Often, a piece of evidence will not fall cleanly into (or be drawn solely from) a single category but will represent some combination, such as actors and processes, processes and technology, or all three. For example:

- Evidence that specific actors have the competence to correctly carry out a particular risk mitigation process (AP).
- Evidence that a given tool correctly implements a security analysis process (PT).
- Evidence that a specific actor followed a prescribed procedure P by applying a security analysis tool to component C, version V, on date D.[12]

[11] An autonomous system-in-context may also play the role of an actor. This makes the three categories a generalization of the oft-cited People, Processes, and Technology. However, within this article we'll only consider actors to be organizations and individuals.

[12] The distinction between individual pieces of evidence and composite evidence, composed of pieces of evidence that are closely related, is not drawn too finely in this article.

Evidential Attributes of Actors, Processes, and Technology

Let's take a more detailed look at the attributes of actors, processes, and technology that relate most strongly to the kinds of evidence that would contribute to compelling assurance arguments and a credible security case. First, we'll look at actors and which attributes of actors convey evidence of assurance, particularly with respect to security. Later, we'll examine the intersection of actors with the elements of a system-in-context.

Actor Attributes

Actor attributes that are relevant to evidence of assurance include

Competence – an actor's skills/expertise for a given task, or more generally, in a specific domain (e.g., credentials are one source of evidence)

Capacity – how much can be accomplished within a given period of time

Trustworthiness – this relates solely to intent: the actor's veracity, honesty, and alignment with the mission of the system[13]

Objectivity – absence of conflicts of interest in a given context

Resources – assets (including economic assets of organizations)

To illustrate how these attributes relate to evidence of assurance, we'll present a few examples. However, since the quality of evidence can only be evaluated within its context of use, it is good practice to present and describe assurance evidence using a template that contains at least the following elements:

Evidence:

Type of Evidence: (classification within evidence framework)

Claim Supported:

Assumptions:

Argument (include caveats):

Also note that actors are not restricted to those within the organization that controls the system-in-context. Actors include the full range of individuals and or-

[13] In general usage, trustworthiness may include competence as well as intent, but here we treat competence separately.

ganizations that can impact the security of your system, including vendors, users, attackers, competitors, governmental regulators, certifiers, standards bodies, service providers, and organizations to which you've outsourced any of your system's functionality or services.

Example 1: Absence of Common Software Defects that Lead to Vulnerabilities

Evidence	All actors on programming team for module X have a certificate showing successful completion of the CERT secure coding course.
Type of Evidence	Actor > Competence & Life Cycle Capability > Implementation > Secure Coding => Credential (Training and Education)
Claim	Module X does not contain any of the common software defects that lead to vulnerabilities.
Assumption	Secure coding course provides sufficient knowledge.[14] Certificate attests to learning, not merely attendance. Credential is sufficiently up-to-date to reflect expertise in the latest secure coding techniques. The programming team has applied the secure coding techniques learned in the course to the code under consideration.
Argument	All programmers working on module X are competent in secure coding techniques.

As with a legal case, combining diverse but complimentary pieces of evidence can bolster the quality of an overall body of evidence (i.e., increase its "evidential force") and therefore strengthen the credibility of a security case [Bloomfield 2003, Pfleeger 2005]. Example 1 provides evidence about preventing the introduction of vulnerabilities into code. This evidence supports an argument that the claim is satisfied but is not, by itself, enough to convince any reasonable reviewer that the claim is true. Additional evidence is required. For instance, code reviews can provide evidence that secure coding best practices have been followed (i.e., the techniques learned during the secure coding course have actually been applied to the code and with demonstrable competence). Complementary evidence about the use of static analysis tools can provide evidence that no common coding errors that lead to vulnerabilities have been detected, or if detected, that the errors have been removed. A formal methods approach (e.g., model check-

[14] Course and program accreditation can support this assumption and therefore bolster the argument.

ing) can provide very strong evidence through formal proof that a given portion of code is free of such defects. Use of a type-checking programming language that prevents the introduction of defects such as buffer overflow vulnerabilities would be further evidence in support of the claim in Example 1.

The assumptions form an important part of the argument. The more “off the wall” the assumptions, the less likely that the argument will be believed. For this reason, the fewer the assumptions, the better. One way to show the validity of an assumption such as “secure coding course provides sufficient knowledge” is through providing an assurance case for the course.

Process Attributes

Process attributes that are relevant to evidence of assurance include the following:

Capability – What can actors accomplish by using the process?

Quality – How good is the process at achieving a desired result, be it analysis of code for a desired security property or the ability to construct a component with a high assurance of security? Quality arguments include conformance to best practices as embodied in recognized standards (or the more general claim of “adherence to industry standard practice “ or “due care”) and the existence of studies that validate the effectiveness of the process, as well as typically weaker arguments about the performance history associated with systems for which this process was used.[15]

Cost/Benefit – How practical is the process, i.e., how much does the process cost with respect to the value obtained?

Context of Use – What is the context in which the process is applicable and achieves valid results?

Repeatability – Within its context of use, is the process readily repeatable over time across different project teams, across different organizations, even across

[15] There is an urgent need for scientific studies that provide evidence that specific processes (measurably) improve specific system quality attributes. In the absence of such evidence, we can only rely on (a consensus of) expert opinion that, for example, a given process improves security, reliability, or safety. Moreover, many best practices and processes describe what should be done in general terms (e.g., “perform a vulnerability analysis”) without providing detailed guidance on how best to carry out the practice or process.

different application domains (i.e., industry segments)? For example, is the process well documented and easy to follow? What organizational and individual resources are needed (e.g., skill sets and tools) so that the process produces the same results each time it is executed, regardless of who executes it?

Reviewable – Does the process document intermediate steps (intermediate inputs and outputs), along with the associated rationale, so that the entire process is reviewable (for example, by an independent third party)?

Schedule, Workflow, and Resources – What actor resources (with what skill sets) and what technology resources are required to carry out the process, and for how long are they needed? What interactions are there among the individuals and workgroups (internal and external) participating in the process? How predictable is the schedule for the process? This attribute is closely related to the cost/benefit attribute above.

Accountability/Attribution – Does the process keep track of the actions of the participating actors, and are the appropriate actors accountable for their actions (i.e., are the results of significant activities attributable to specific actors?) For example, are test results dated, linked to the correct version number of the code, digitally signed, and stored in a database for future use as evidence of assurance?

Example 2: Requirement Document Accurately Reflects Security Concerns

Evidence	A requirements document containing a set of initial security requirements and a description of the decision-making process and rationale for how they were selected. This evidence is the output from applying the nine-step Security Quality Requirements Engineering (SQUARE) methodology [Mead 2005].
Type of Evidence	Process & Life Cycle Capability > Security Requirements => Security Requirements (and Rationale) Document
Claim	The security requirements accurately reflect the high-priority security concerns of the system stakeholders.
Assumption	The process is sound and used valid inputs and techniques to produce its output. The process produced valid results in the context of the domain in which it was applied. Actors participating in the process had sufficient competence to satisfactorily fulfill their role in the process.

Argument	Competent staff applied the SQUARE methodology, a sound requirements elicitation and prioritization process in an appropriate context of use. The output of the process correctly reflects the high-priority security concerns of the system stakeholders because the process is based on business drivers, threat analysis, risk analysis, risk tolerance of the organization, and cost/benefit analysis. The process documents the rationale for the selection of security requirements, along with all intermediate steps, so that the results of the entire process can be reviewed by others (e.g., outside experts).

Technology Attributes

Technology attributes that are relevant to evidence of assurance include the following, several of which are the same as process attributes:[16]

Capability – What can the technology accomplish?

Quality – Which software or system quality attributes (security, reliability, fault tolerance, etc.) are associated with the technology?

Visibility – Are the artifacts of the life cycle processes used to create the technology available to users, customers, certifiers and others, so that the technology's quality attributes can be assessed through analysis of those artifacts?

Cost/Benefit – How affordable is the technology relative to its value?

Context of Use – What is the context in which the technology is applicable and achieves valid results?

Resources – What actor resources (with what skill sets), processes, and other technology resources are required to make effective use of the technology?

Traceability/Accountability/Attribution – Does the technology keep track of the actions of the participating actors, and are the results of significant activities attributable to specific actors? Are significant events logged and available for audit?

Example 3: Formal Verification of a Security Property

[16] For a detailed discussion of the kinds of evidence of assurance that could be used to assess the security and survivability of commercial-off-the-shelf (COTS) products, see [Lipson 2002].

Evidence	Output from a model-checking tool
Type of Evidence	Technology & System => Analysis > Formal verification
Claim	Protocol/Code/Component satisfies a security property[17]
Assumption	The formal model matches reality
Argument	The argument must show how the model-checking language translates faithfully from the real world and that no significant assumption is left out. Often a formal language is arcane, making it difficult for a non-expert observer to verify mapping from real-world to model.

Example 4: Conformance to a Security Standard

Evidence	(1) Document specifying a process *P* based on security standard *S*, and (2) a set of life cycle artifacts showing the implementation of component *C* using process *P*.
Type of Evidence	(1) Process & Life Cycle Capability > Conformance to Standards => Process Specification (2) Process & Life Cycle Capability > Implementation => Artifacts (Code, Test Results, Code Review Results)
Claim	Component *C* constructed using process *P* has the desired security properties.
Assumption	The process *P* faithfully captures all relevant aspects of the standard *S*. The standard *S* embodies best practice. The best practice has been shown to be effective in producing the desired security property.
Argument	Show that the mapping from standard *S* to process *P* is comprehensive. Preferably use studies to show effectiveness of *S*, otherwise argue performance history (less formal than studies) or expert opinion. Can also argue "independence," such as objectiveness of the standards body (e.g., IEEE) versus a quickly constructed "standard" from a trade group.

Example 5: Test Results as Evidence of Security Properties

[17] A risk as low as reasonable practicable (ALARP) is always assumed for such claims [Weaver 2003].

Evidence	Results from running test of the system using attacks of type *A*
Type of Evidence	Technology & System => Test Results
Claim	The system is secure against attacks of type *A*.
Assumption	The tests can adequately exercise the code, and therefore the results are a reasonable measure of code quality.
Argument	Since the level of assurance needed isn't ultra-high, it is possible to run enough tests for long enough to make the results believable. The tests have good coverage of the software under test. The testing methods have been proven to work in other similar systems.

Example 6: Simulation Results as Evidence of Security Properties

Evidence	Results from simulated attack *A* on the system
Type of Evidence	Technology & System => Simulation
Claim	The system is secure against attacks of type *A*.
Assumption	The simulation is an adequate model of the system.
Argument	Show how the simulation can be taken as a representative of the real system. Show that the simulation results are representative of real results.

This simulation example and the previous testing example illustrate two types of evidence that can be mutually supportive. Simulation allows early analysis of system properties, but it assumes a sufficient match between the quality attributes of the simulation model and those of the actual system. A major advantage is that any corrections or improvements can be made early in the development processes, which reduces overall cost. However, any significant deviation between the model and the actual system can be quite problematic. Testing is performed on the actual system software, either pre-release (e.g., unit or integration testing) or deployed (e.g., penetration testing), but any problems that are discovered must be remedied later in the engineering life cycle, where changes are more expensive.

Example 7: Evidence of Security Growth (i.e., continuous improvement)

Evidence	(1) A report summarizing process improvement effort *E* over past two years (2) The number of vulnerability reports for the system has decreased by 50% for each of the last two years
Type of Evidence	Process & Life Cycle Capability > Process Improvement => Report on Process Improvement Effort / Specification of Modified Processes Process & Incident Response > Vulnerability Reports => Statistics
Claim	The security of the system is showing continuous improvement (a.k.a. “security growth”).
Assumption	There is a performance history of security growth associated with process improvement efforts similar to *E,* for this industry sector.
Argument	Growth curves for similar systems apply to this system. The history of security of this system is such that it shows security is increasing (less frequent reports of vulnerabilities).

A Few Basic Steps for Building Assurance In

Take steps to preserve existing evidence and to improve or generate new evidence throughout the system development life cycle. For example, instead of simply collecting raw test results (or not collecting them at all), record what modules and versions they apply to and the date and time run, and have the results signed by a supervisor or other responsible party—ideally with a digital signature that confirms the validity of the entire document. Carefully preserve your evidence within a version control system database. There is additional cost associated with collecting and generating higher quality evidence, but managers should consider the benefits of being able to back up stronger claims about their system’s security. Supplementing raw test results with some additional narrative that interprets the test results in terms of the security requirements (providing traceability back to the requirements specification, or even mapping the results back to the threat analysis) would greatly improve the quality of the evidence, instead of merely preserving raw data that may not be understood or be of value at a future date. You may even wish to invite objective third-party expert reviewers to participate in the very early stages of the development life cycle to help you decide what evidence to build in (based on their experience with what certifiers and regulators usually look for). As a result, the evidence will be guaranteed to be available to certifiers or regulators who will require it once the system is complete.

THE FUTURE OF SECURITY CASES – OPEN ISSUES AND CHALLENGES

Although our main focus has been on security cases, these open issues and challenges apply to assurance cases for any software or system quality attribute.

Dealing with Massive Amounts of Evidence

Tools and methodology improvements are needed to handle the massive amount of evidence needed to demonstrate assurance even for systems of moderate size. Intuitively, the amount and quality of evidence should be proportional to the degree of assurance specified by the top-level security claim, which in turn should be commensurate with the criticality of the system that is the subject of the security case. The trend toward ultra-large-scale systems [Northrop 2006] means that the amount of evidence needed to support security claims for high-assurance systems will be even more massive in the future. Automated tools to support security case analysis as well as artifact analysis will be urgently needed.

Dealing with Complexity

Systems must be built differently. We must generate and gather evidence from the outset and build systems with simplicity in mind [Sha 2001, Jackson 2007] to make the creation of credible assurance cases more practical. The creation and maintenance of the assurance case will become as important a goal for the development team as the system itself, because without the accompanying security case the system would not be permitted to be used in critical applications where public health and safety are at stake.

Don’t Treat Software or System Quality Attributes in Isolation

Assurance cases for each of the software or system quality attributes (SQAs) should not be built in isolation. The security case should be part of a larger survivability or dependability case that includes all SQAs relevant to mission success (e.g., security, reliability, and safety). Tradeoffs among the various SQAs must be explicitly demonstrated in a composite survivability (or dependability) case along with the design rationale so that as the system evolves, these tradeoffs can be maintained. Moreover, to reflect changes in the threat environment, these tradeoffs can be explicitly modified within updated assurance cases that demonstrate that the new SQA tradeoffs have actually been realized.

Training, Education, and Awareness Are Essential

Our society’s critical dependence on highly distributed, networked systems means that accurately assessing the security of such systems is essential. However, security properties are particularly difficult to evaluate when the only evidence we have is glitzy brochures, penetration test results, and testimonials. A higher level of confidence is needed, and security cases can provide that higher degree of assurance.

As we've stated throughout this article, the credibility of a security case ultimately rests on a foundation of evidence. The more familiar the practitioner is with the nature of evidence that can contribute to a security assurance case and how to facilitate the gathering, generation, and preservation of such evidence, the more valuable that practitioner will be to their employer and to the stakeholders of the systems their organization has developed or acquired. Many more scientific studies are needed to provide guidance to practitioners by validating or refuting today's best practices, methodologies, and tools that are the source of much of the currently available evidence of assurance. Training, education, and awareness about the emerging discipline of assurance cases are essential not only for system and software engineering practitioners and their managers, but also for policy makers (including regulators) and for all organizations and individuals who are increasingly dependent on the critical services provided by modern software-intensive systems.

BIBLIOGRAPHY

Ankrum 2005

Ankrum, T. S. & Kromholz, A.H. "Structured assurance cases: three common standards," 99-108. *Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*, 2005.

Avizienis 2000

Avizienis, Algirdas, Laprie, Jean-Claude, & Randell, Brian. "Fundamental Concepts of Dependability." *Proceedings of the Third Information Survivability Workshop (ISW2000).*

Bloomfield 2003

Bloomfield, Robin & Littlewood, Bev. "Multi-legged

Arguments: The Impact of Diversity Upon Confidence in Dependability Arguments." *Proceedings of 2003 International Conference on Dependable Systems and Networks*, San Francisco, California. IEEE Computer Society Press, 2003.

Caralli 2007

Caralli, R., Stevens, J., Wallen, C., White, D., Wilson, W., & Young, L. *Introducing the CERT Resiliency Engineering Framework: Improving the Security and Sustaina-*

bility Processes (CMU/SEI-2007-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2007.

CCDSS 2007

Committee on Certifiably Dependable Software Systems. *Software for Dependable Systems: Sufficient Evidence?* Edited by Daniel Jackson, Martyn Thomas, and Lynette I. Millett. National Research Council, 2007 (ISBN 978-0-309-10394-7).

Despotou 2007

Despotou, Georgios. “Managing the Evolution of Dependability Cases for Systems of Systems.” PhD thesis, High Integrity Systems Research Group, Department of Computer Science, University of York, United Kingdom, April 2007.

Goodenough 2007

Goodenough, J., Lipson, H., & Weinstock, C. “Arguing Security – Creating Security Assurance Cases.” Department of Homeland Security *Build Security In* website, Jan. 2007.

Graydon 2007

Graydon, Patrick J.; Knight, John C.; Strunk, Elisabeth A. “Assurance Based Development of Critical Systems,” 347-356. *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007*, June 2007.

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.

Kornecki 2003

Kornecki, Andrew J. “Assessment of Software Safety Via Catastrophic Events Coverage,” 1139–1144. *Proceedings of the 21st IASTED International Multi-Conference on Applied Informatics*, 2003.

Lautieri 2004

Lautieri, Samantha, Cooper, David, Jackson, David, & Cockram, Trevor. “Assurance Cases: How Assured Are You?” *Supplemental Volume to DSN-2004, The Proceedings of the 2004 International Conference on Dependable Systems and Networks*, 2004.

Lipson 1999

Lipson, Howard & Fisher, David. "Survivability—A New Technical and Business Perspective on Security," 33–39. *Proceedings of the 1999 New Security Paradigms Workshop*. Caledon Hills, Ontario, Canada, Sept. 22–24, 1999. New York: Association for Computing Machinery, 2000.

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.

Littlewood 2007

Littlewood, Bev. "Limits to Dependability Assurance – A Controversy Revisited." Presentation at the 29th International Conference on Software Engineering, ICSE 2007, May 20-26, 2007, Minneapolis, MN. IEEE Computer Society TCSE; ACM SIGSOFT.

Mead 2001

Mead, Nancy R., Linger, Richard C., McHugh, John, & Lipson, Howard F. "Managing Software Development for Survivable Systems." *Annals of Software Engineering*, Volume 11, No. 1, 2001, 45–78.

Mead 2005

Mead, N. R., Hough, E., & Stehney, T. *Security Quality Requirements Engineering (SQUARE) Methodology* (CMU/SEI-2005-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

Moore 2000

Moore, Andrew & Strohmayer, Beth. *Visual NRM User's Manual* (NRL/FR/5540--00-9950). Washington, DC: Naval Research Laboratory, May 31, 2000.

Northrop 2006

Northrop, Linda et al. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, June 2006.

Okun 2007

Okun, Vadim, Guthrie, William F., Gaucher, Romain, & Black, Paul E. "Effect of Static Analysis Tools on Software Security: Preliminary Investigation." *Proceedings* *of the 2007 ACM workshop on Quality of Protection*. ACM Conference on Computer and Communications Security (ACM CCS), 2007.

Ozment 2007

Ozment, Andy "Improving Vulnerability Discovery Models." *Proceedings of the 2007 ACM workshop on Quality of Protection*, ACM CCS, 2007.

Pfleeger 2005

Pfleeger, Shari Lawrence. "Soup or Art? The Role of Evidential Force in Empirical Software Engineering." *IEEE Software*, January/February 2005.

SAE 2004

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

Sha 2001

Sha, Lui. "Using Simplicity to Control Complexity." *IEEE Software*, July/August 2001.

Tsai 1998

Tsai, W.T., Mojdehbakhsh, R., Zhu, F. "Ensuring System and Software Reliability in Safety-Critical Systems," 48-53. *Proceedings of the 1998 IEEE Workshop on Application-Specific Software Engineering and Technology (ASSET-98)*, 1998.

Van Eemeren 1996

Van Eemeren, Frans H., Grootendorst, Rob, Johnson, Ralph H., Plantin, Christian, Willard, Charles A., Zarefsky, David, Blair, J. Anthony, Henkemans, A. Francisca Sn, Krabbe, Erik, Woods, John H. *Fundamentals of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Developments*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 1996.

Weaver 2003

Weaver, Robert A. "The Safety of Software – Constructing and Assuring Arguments." PhD diss., University of York, September 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.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Department of Homeland Security or the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM-0001120