Carnegie Mellon University
Software Engineering Institute

Building a Cybersecurity Strategy

Carol Woody, PhD. & Robert Ellison, PhD

October 2020

Introduction

Today's missions rely on highly integrated and complex technology that must be protected from a wide range of adversaries. This technology must operate in a highly dynamic and contested environment. Reliance on operational security controls has provided insufficient protection for quite some time. The role of cybersecurity engineering is growing in importance as adversaries' capabilities increase and systems shift from being "built for purpose" to integrating reused components including legacy, third-party software, open source software, and external services (e.g., Azure Cloud and Platform One).

A cybersecurity strategy is a critical element in ensuring that each component and the composition of these components have sufficient security to address a mission. This strategy does not occur without planning, design, monitoring, and enforcing considerations of cybersecurity at all levels. It is necessary to consider compliance, mandates for an authority to operate, and good cybersecurity hygiene; however, these steps alone are not sufficient to ensure the composition is secure enough. These responsibilities touch on every aspect of the lifecycle. The current approach of programs is to establish silos of excellence separating lifecycle activities; this approach requires a high level of collaboration for cybersecurity that must cross all activities, and that collaboration cannot be assumed.

The owners of the cybersecurity strategy are responsible for defining how a system's cybersecurity performs to meet its mission, even under attack. These responsibilities include activities that achieve the following:

- Plan and design trusted relationships.
- Negotiate appropriate security requirements to ensure confidentiality, integrity, and availability with sufficient monitoring in systems and software.
- Plan and design sufficient resiliency to recognize, resist, and recover from attacks.
- Plan for operational security under all circumstances, including designed-in methods of denying critical information to an adversary to avoid or minimize mission impact.
- Evaluate alternatives to determine the level of accepted cybersecurity risk.

This paper focuses on the elements of the cybersecurity strategy that are critical in the early segments of the lifecycle.

Cybersecurity Engineering Role

Integrating cybersecurity considerations into the early segments of a lifecycle affects acquisition as well as system and software engineering. Program management is ultimately responsible for cybersecurity;[1] however, if they lack that specific expertise, management needs to bring in resources with specialized knowledge and operational expertise, which we describe as "cybersecurity engineering." This concept can be represented in one-to-many resources.

Cybersecurity engineering should focus on the following six key areas. Although these areas are critical for building technology to operate in today's highly contested environments,[2] they are typically given insufficient consideration in cybersecurity:

- risk determination
- defining and monitoring system and component interactions
- trusted dependencies
- attacker response
- coordination of security throughout the life cycle
- measurement for cybersecurity improvement

In *risk determination*, cybersecurity engineering incorporates the effective consideration of threats and mission risk. Perceptions of risk drive assurance decisions, and the lack of cybersecurity expertise in risk analysis can lead to poor assurance choices. Involving individuals with knowledge about successful attacks and how threats can impact the system's operational mission can be critical in the decision-making steps for appropriate prioritization.

For *defining and monitoring system and component interactions*, cybersecurity engineering considers risk to systems from the interaction among technology components and external systems. Highly connected systems require the alignment of cybersecurity risk across all stakeholders, system components, and connected systems; otherwise, critical threats can remain unaddressed (i.e., missed or ignored) at different points of interaction.

The following risk areas should be considered in design and process decisions:

- Interactions must be designed to be assured and segments of the design will be scattered across various interacting components; verification that the pieces are all effectively working together must be part of integration validation.
- There are costs to addressing assurance, and tradeoffs must be made among performance, reliability, usability, maintainability, etc. These costs and tradeoffs must be balanced against the impact of the risks. Then choices must be consistently applied across the range of participating components.

1 Outline and Guidance for Acquisition Program's Cybersecurity Strategies, November 10, 2015, http://www.acqnotes.com/wp-content/uploads/2014/09/DoD-CIO-Cybersecurity-Strategy-Outline-and-Guidance-10-Nov-15-1.pdf

2 "Principles and Measurement Models for Software Assurance." Nancy R. Mead, Dan Shoemaker, & Carol Woody. *2013 IJSSE Special Issue on Cybersecurity Scientific Validation*. January 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=298843

- Interactions occur at many technology levels (e.g., network, security appliances, architecture, applications, data storage) and are supported by a wide range of roles. The choices made at each level must be consistently applied across all levels for effective results.

In *trusted dependencies*, cybersecurity engineering evaluates the dependencies and inherited risk to ensure the appropriate level of trust is established. The following are key dependency considerations where trust is involved:

- Each dependency represents a risk that needs to be shared among interfacing components.
- Dependency decisions should be based on a realistic assessment of the threats, impacts, and opportunities represented by an interaction. Controls placed on the interaction should reflect this analysis.
- Dependencies are not static, and trust relationships should be reviewed to identify changes that warrant reconsideration.
- Using many shared components (e.g., reuse, open source, collaboration environments) to build technology applications and infrastructure increases the dependency on other's assurance decisions that may not meet mission needs.

Planning an *attacker response* is a responsibility that cybersecurity engineering should oversee to ensure that system capabilities are included to allow effective handling of the types of attacks that can be mission critical. A broad community of attackers exists that has growing technology capabilities. This community is able to compromise the confidentiality, integrity, and availability of any and all of a system's technology assets, and the attacker profile is constantly changing.

Since there are no perfect protections and attacker capabilities continue to grow, this coordination must consider the need to recognize, resist, and recover. Ensuring a system is prepared to work, even when under attack, requires extensive planning and coordination across all components and technologies. The attacker uses the same technologies, processes, standards, and practices to craft a compromise (socio-technical responses) as system builders. Attacks are crafted to exploit the ways technology is normally used or designed to contrive exceptional situations where defenses are circumvented.

Coordination of cybersecurity across the lifecycle should be the responsibility of cybersecurity engineering. Each step of the lifecycle should include preparing for the fielded system. Protection must be applied broadly across people, processes, and technology because the attacker will take advantage of all possible entry points. This span of protection includes acquisition decisions about software and services integrated into the system. The role of implementing a cybersecurity strategy requires coordination among system and software engineering, architects and designers, developers, testers, verifiers, and implementers to identify potential gaps and ways of addressing them to ensure the operational mission.

Measurement for cybersecurity improvement needs to be a responsibility of cybersecurity engineering to coordinate data—from the various lifecycle steps, decision-making levels, and system component evaluations—to show that the steps designed to address cybersecurity are delivering expected results. Tools can track vulnerabilities in code, testing can show defects, and architecture analyses can identify design weaknesses. However, until these elements are integrated, the operational risk perspective is missing. All elements of the socio-technical environment (e.g., practices, processes, procedures, products) must tie together and measurements must be consistent.

Cybersecurity engineering should monitor the range of lifecycle activities needed to contribute to assurance. Monitoring these activities demonstrates how the program is progressing toward its compliance and cybersecurity operational goals. Figure 1 shows the range of activities that can be applied at various stages of the lifecycle to address design, coding, and implementation weaknesses that contribute to reduced operational risk. Cybersecurity engineering should assemble data to show that the combination of lifecycle activities produces the results needed to reduce mission impact from undetected weaknesses.

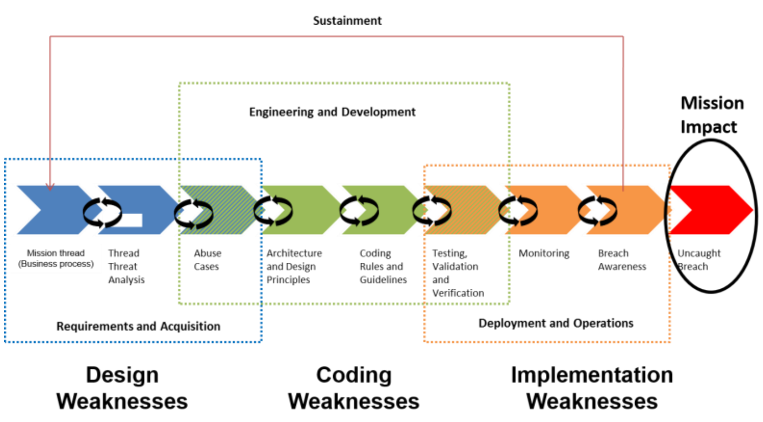
Sustainment
Engineering and Development
Mission Impact
Mission thread (Business process)
Thread Threat Analysis
Abuse Cases
Architecture and Design Principles
Coding Rules and Guidelines
Testing, Validation and Verification
Monitoring
Breach Awareness
Uncaught Breach
Requirements and Acquisition
Deployment and Operations
Design Weaknesses
Coding Weaknesses
Implementation Weaknesses

Figure 1: Addressing Cybersecurity Risk Across the Lifecycle

Software Assurance

Software assurance is the level of confidence that the software (1) is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and (2) functions as intended. Vulnerability requirements lead naturally to a security focus, but the design of a response must be based on the potential consequences. How could a security compromise affect the intended behavior? Often adverse changes in intended behavior affect data management. Has information been disclosed or modified? Except for software-intensive mission-critical systems, the most significant effects of a security compromise likely involve changes in intended behavior related to reliability, safety, and availability.

Too often, an organization becomes aware of a successful security compromise only by observing the consequences. A critical security risk for mission-critical systems is that a security compromise is identified only after encountering a safety or reliability operational fault. Such a risk increases the importance of eliminating security weaknesses during the development lifecycle. The *2005 Department of Defense Guide for Achieving Reliability, Availability, and Maintainability (RAM)* emphasized the importance of systems engineering design analysis over predicting software reliability based on an analysis of faults found during integration [DoD 2005]. Engineering analysis for cybersecurity is equally important.

In particular, threat modeling is essential during detailed systems engineering design to identify potential weaknesses and propose appropriate mitigations. The development of an attack surface guides threat modeling. Threat modeling should also generate the outline of an assurance argument. What are the most consequential threats? How will those threats be sufficiently mitigated? How will the desired level of confidence be demonstrated? Where possible, that confidence should be based on engineering analysis, such as with design reviews and inspections.

Late lifecycle activities, such as testing, static analysis, and dynamic analysis, typically offer only incomplete assurance. The design of the DevSecOps pipeline, for example, should reflect the planned assurance proposed by the cybersecurity engineering analysis done earlier in the development lifecycle. Prioritizing requirements, selecting tools for weakness identification and removal, tracking unaddressed weaknesses, and establishing approval mechanisms for operational release should combine to deliver data that supports growing confidence in the operational assurance of the developing product.

Threat modeling incorporates engineering analysis into the design phases of the lifecycle and supports verification and validation for demonstrating assurance as shown in Figure 2. Threat modeling identifies possible weaknesses that could be exploited and proposes mitigations if they are exploited. That engineering analysis typically provides guidance on how to validate those mitigations, such as with specific unit or integration tests.

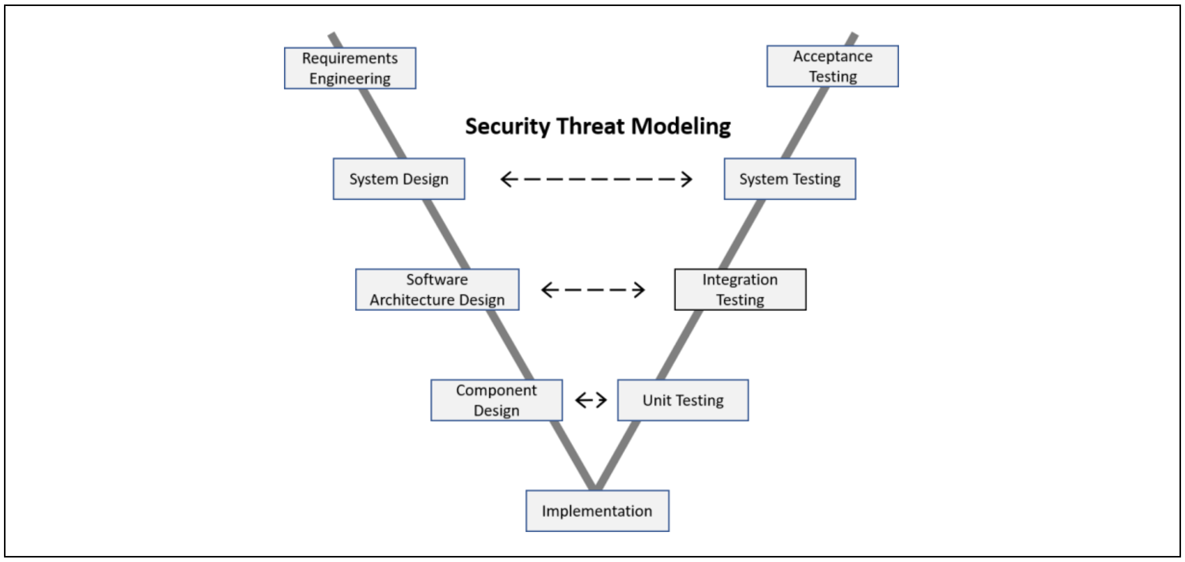

Figure 2: Incorporating Engineering Analysis

Engineering analysis can play an equivalent role for reliability, availability, and maintainability as recommended in the *2005 Department of Defense Guide for Achieving Reliability, Availability, and Maintainability* [DoD 2005].

How can we determine the confidence that a system will behave as expected? It is impossible to examine or test every possible combination of conditions that could affect a system. But achieving that confidence is important to those acquiring, developing, and using the system. Such confidence should be based on concrete evidence and not just on the opinions of developers or reviewers. An assurance case provides the argument mapped to available evidence. The level of confidence in a system should depend on understanding the evidence that leads to an increase in the confidence that a system property holds.

Evaluating an assurance case can also be subjective because evidence can be incomplete and/or inconsistent. One approach for evaluating confidence from a collection of evidence is estimating the likelihood that a claim is false. One approach is to test the argument against the evidence provided by a developer; it might be insufficient to justify the claim. For example, the following would indicate possible unsubstantiated arguments:

- The test plans did not include all of the hazards identified during design.
- The web application developers had limited security experience.
- The acquirer did not provide sufficient data to validate the modeling and simulations.
- Integration testing did not adequately test recovery following component failures.

It is not obvious, but such an approach constructs an alternate assurance case for the same claim. Instead of constructing an argument for the validity of a claim supported by available evidence, we identify the various possibilities for why the claim is false {Goodenough 2015]. An assurance case consists of gathering evidence or performing analysis that removes those possibilities. Each eliminated possibility removes a reason for doubt and thereby increases our confidence in the claim. During a review, the developer should be able to show how to eliminate the doubts that are raised. This approach moves beyond a simplistic focus on whether the software satisfies the requirements to also consider what the software should not do. This approach is especially important for reused and third-party software that was not built to a specification.

When performing an assurance case analysis of a completed design, the outcome is rather black and white—either design artifacts are complete and sufficient, or they are not. Reviewing an in-progress design requires a more nuanced approach—one that reflects relative risk—since the design artifacts are (necessarily) in different stages of completion. For this example, the SEI used a simple and familiar stoplight approach to scoring (named for the red, yellow, and green colors it uses) [Blanchette 2009]. The color red designates a relatively high-risk area, the color yellow designates a relatively medium-risk area, and the color green indicates a relatively low-risk area. The rules for assigning colors are slightly different at the evidence level than they are at the level of the claims, as is shown in Figure 3.

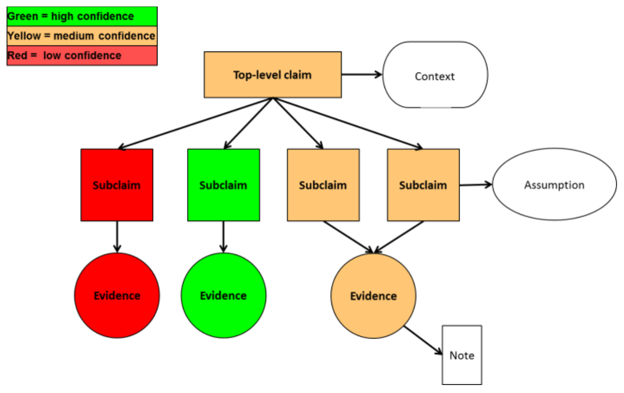

Figure 3: Scoring Legend

When the sub-claims are not all uniformly the same color, an analyst must make a subjective decision about the risk to assign to a node. For example, an analyst might conclude a medium risk given the following doubts raised about the evidence and arguments:

1. Only a subset of information exchanges has been implemented to date.
2. The noted risks are, at best, medium at this time.
3. The security architecture has not been completely propagated across the system.
4. An evaluation of the security architecture revealed some design choices that will prevent system accreditation.
5. Preliminary field tests indicate some information exchanges are exceeding prescribed timelines for completion.

The feasibility of achieving high assurance for a particular system is strongly influenced by early engineering choices. A summary of the results of a number of studies on errors that were introduced in the development lifecycle and when they are discovered is shown in Figure 4 [Feiler 2012]. Requirement and design errors dominate, and there are few tools available to automatically find them.

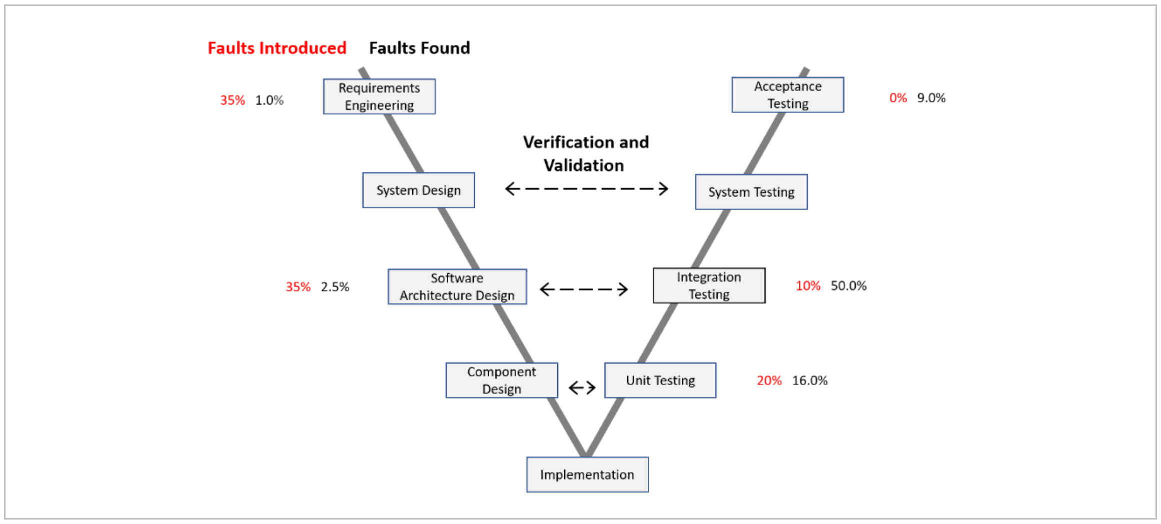

Figure 4: Faults Introduced and Found During Development

Cybersecurity Strategy

Evidence is available at many points along the lifecycle, but structuring it in a way that it is useful for establishing software assurance requires careful planning and analysis. The cybersecurity strategy must integrate the expertise from cybersecurity engineering with evidence of how the product was planned and how it actually performs. As noted in a recent Blog by Phil Venables, a specialist in enterprise risk, information and cybersecurity, and business resilience who sits on the National Institute of Standards (NIST) Information Security and Privacy Advisory Board, cybersecurity must be carefully folded into systems delivery.[3] This cannot be done in a haphazard manner if the results need to meet critical operational requirements.

An assurance case can be used as a framework to connect the various elements of this analysis to show gaps and sufficiency. By mapping evidence that is appropriately generated and selected from the various steps of the lifecycle into an assurance case, considerations for how the system should function and how it should not function can be collected and analyzed. If done properly, this evidence will demonstrate that the system effectively addresses software assurance.

References

[Blanchette 2009]

Blanchette, S. *Assurance Cases for Design Analysis of Complex System of Systems Software.* American Institute for Aeronautics and Astronautics (AIAA) Infotech@Aerospace Conference. Seattle, Washington, U.S.A., April 2009. http://resources.sei.cmu.edu/asset_files/WhitePaper/2009_019_001_29066.pdf

[Feiler 2012]

Feiler, Peter; Goodenough, John; Gurfinkel, Arie; Weinstock, Charles; & Wrage, Lutz. *Reliability Improvement and Validation Framework* (CMU/SEI-2012-SR-013). Software Engineering Institute, Carnegie Mellon University. 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=34069

[Goodenough 2015]

Goodenough, John; Weinstock, Charles; & Klein, Ari. *Eliminative Argumentation: A Basis for Arguing Confidence in System Propertie*s. CMU/SEI-2015-TR-005. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=434805

[DoD 2005]

Department of Defense. *Department of Defense Guide for Achieving Reliability, Availability, and Maintainability.* http://www.acqnotes.com/Attachments/DoD%20Reliability%20Availability%20and%20Maintainability%20(RAM)%20Guide.pdf

[3] https://www.philvenables.com/post/cybersecurity-macro-themes-for-the-2020-s-updated

Contact Us

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

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu