Carnegie Mellon University
Software Engineering Institute

SOFTWARE ASSURANCE GUIDANCE AND EVALUATION (SAGE) TOOL

Luiz L. Antunes, Ebonie McNeil, Robert B. Schiela, and Hasan Yasar

May 2021

Overview

Introduction

The Software Assurance (SwA) Evaluation was developed by the Carnegie Mellon University Software Engineering Institute (SEI) to assess systems development and operations practices and to identify potential vulnerabilities and opportunities to improve and secure processes.

The creation of the Software Assurance Guidance and Evaluation (SAGE) tool required a thorough analysis of the most popular standards and frameworks for software assurance, secure coding, Agile, and secure DevOps, used both in industry and government settings. As a result of this analysis, both the questions and the provided guidance draw from modern practices used in software design, development, test, and operation. The appendix contains a list of some of the standards and frameworks used in the elaboration of this tool.

The evaluation questions are distributed across the software production and operation phases. To achieve the best results, the questionnaire should be applied as a survey to people working in different stages of the production/operation line. Due to overlaps in responsibilities, participants will find that many questions apply to their work, even if they initially appear to be outside of their areas. The multiple evaluation responses, from participants in different roles, will provide different perceptions of the same processes, contributing to the richness of the answers.

How to Apply the SAGE Tool

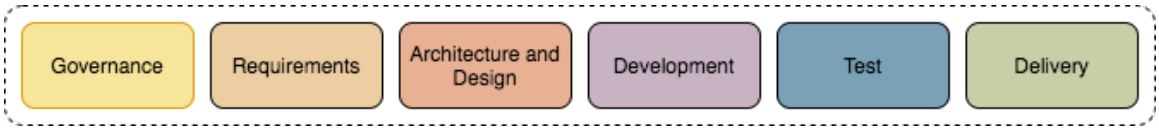
This document is composed of two complementary parts: a questionnaire section, which can be distributed to different teams across the software development and operations areas in your organization, and a guidance section, containing a compendium of best practices executed today in both industry and government settings.

In order to use this evaluation, its parts can be printed separately and used at different stages of the assessment. The questionnaire should be distributed to participants along the processes of the Software Development Lifecycle (SDLC). Team members should try to answer the questions in the most accurate possible way in order to capture as much information from the processes as possible.

The guidance section can be used by technical management or project teams along with the questionnaire section to assess the current state of the practices used and suggest improvements to add to the current processes performed in the SDLC.

Secure Development Phases

The SwA Evaluation is comprised of six Secure Software Development Lifecycle (S-SDLC) phases. For each phase, a list of recommended security activities has been identified to promote awareness of modern best practices, some linked to specific guidance within this document. The evaluation has been categorized to closely relate to the widely used SDLC; this alignment can help organizations that develop software within a waterfall model transition to a more iterative and Agile approach to software development. The guidance and recommendations may also be used when acquiring software. Each phase, shown in the diagram below, is briefly explained and followed by a list of related security activities.



1. Governance

The Governance Phase includes activities that are known as setup or inception activities. Although these activities are usually completed at the beginning of the development lifecycle, this document proposes that these activities are ongoing and should be revisited throughout development and sustainment as needed. The governance questions highlight important documentation and language that should be used up front to reduce risks and establish an understanding of security importance within the project. Some practices that promote security in the project at this phase are

- strategy and metrics
- policy and governance
- education and security guidance
- organizational risk factors assessment
- threat assessment

2. Requirements

Requirements generation is one of the most critical activities during development. Requirements set stakeholder expectations of define software functionality. Requirements are most often developed with use cases in mind. The recommendations in the evaluation emphasize the inclusion of abuse cases or prohibited behaviors when generating requirements. The utilization of both abuse and use cases provides a more robust software application and better test coverage in later phases of the lifecycle. The following practices, used during the elaboration of the systems requirements, focus on security:

- security requirements (security functional requirements [SFR]/security assurance requirements [SAR]) generation
- risk assessment
- abuse case development
- threat modeling
- security stories

- development tool screening
- securing/hardening environments

3. Architecture and Design

The Architecture and Design Phases integrate requirements into diagrams and implementation concepts. Architects and designers should consider security practices to identify vulnerabilities and potential attacks from internal and external sources. Initial designs should be approved and continuously updated and validated against actual development activities. Threat assessments and modeling should be considered in the Architecture and Design Phases. Security aspects can be considered during these phases through the following activities:

- security architecture design
- architectural risk analysis
- security design requirements generation
- attack surface analysis
- vulnerability analysis and flow hypothesis
- security design review
- dependency lists and open-source libraries analysis

4. Development

Software assurance is most notably related to the Development Phase. Software should be developed using proper design principles and security techniques that will not hinder mission assurance. Common best practices should be used by developers including, but not limited to, secure coding, static and dynamic analysis, and code reviews. Below are some security activities that can be implemented during the system development:

- secure coding practices
- security-focused code review
- unsafe function deprecation
- security unit testing
- static code analysis
- traceability analysis

5. Test

Traditionally, organizations focus their security efforts later in the lifecycle, during the Test Phase. This evaluation moves away from that notion, instead placing security up front and early. The evaluation highlights the importance of software-related security activities in the Test Phase that have significant impact on the overall security posture of the system. The evaluation aims to emphasize the use of continuous integration and automated testing methods to foster more efficiency in an iterative environment. Below are some security activities that provide a significant contribution to testing software:

- security test planning
- security testing

- fuzz testing
- risk-based security testing
- dynamic analysis
- penetration testing
- verification of security implementation
- verification of process and procedures
- dependency monitoring

6. Delivery

The Delivery Phase correlates to the build, deployment, sustainment, and transition phases of other software development and acquisition paradigms. While stakeholder participation is mentioned throughout the evaluation, this phase draws attention to the significance of stakeholder inclusion. In order to create a feedback loop to the other phases in the lifecycle, and to catch dormant or regressive issues as early as possible, stakeholder participation is indispensable. Here are some security activities that help us achieve that goal:

- container security practices
- final security review
- certify, release, and archive
- security acceptance testing
- transition incident response planning
- application security monitoring
- secure deployment process
- secure environment practices
- secure operational enablement
- configuration management

Assessment Value and Takeaways

This assessment has two primary goals: The first goal is to provide situational awareness of what security-focused practices have already been put in place to reduce the risks of software failure throughout the SDLC. The second goal is to provide useful guidance on what software assurance practices are currently used both in industry and government settings. This guidance may be used to inform a program SwA assessment team on how to proceed in cases mentioned in this document that are not covered in a given system and to justify any corrective actions that need to take place during the SDLC.

Here is a brief summary of what can be learned or expected at the completion of the questionnaire for each phase:

1. **Governance**: At this phase, the questionnaire will assess what policies and governance have been considered for the execution of the project, what standards have been examined, what security strategies have been proposed, and which metrics allow stakeholders to verify the efficacy of

those security strategies. The guidance section provides significant information on current best practices to cover all these topics.

2. **Requirements**: Within the Requirements Phase, questions will evaluate how system risks and abuse cases have been considered/developed and how security stories have been composed based on those risks. The guidance contains suggestions on how to consistently consider those stories and the issues represented by them throughout the SDLC, with the objective of arriving at the end of each iteration with a more secure product.
3. **Architecture and Design**: Throughout the Architecture and Design Phases, requirements captured in the initial stages of the project start taking shape as high-level definitions that already consider technical constraints and risks to the system. The questions and guidance in this section will ensure that security considerations properly influence the design, leading to a more secure Development Phase and providing consistency with security requirements in the Test Phase.
4. **Development**: During development, product features previously designed at a high level will materialize as code, which must be carefully crafted to avoid introducing any security gaps. The questions and guidance at this phase ensure that secure coding best practices, DevOps procedures, and unit testing are being properly applied to guarantee a more secure code base.
5. **Test**: Testing is fundamental to the product lifecycle. Through testing, the team can verify if the features coming from security requirements function properly and, if not, dispatch reports to require corrective action. At this phase, testing becomes more thorough and distributed across the whole product code and environment, as opposed to localized testing that has already been performed during the Development Phase. The questions and guidance presented at this stage focus on the types of testing that can be applied as well as the frequency to execute them. There is significant guidance about continuous integration and how testing automation is responsible for one of the largest impacts in the whole SDLC, introducing consistency and agility to it.
6. **Delivery**: The questions about the Delivery Phase will help your team evaluate how the system will make it to production and establish a strategy for handling any latent issues that may appear after deployment. The guidance mentions parts of policies that define actions to be taken in dealing with contractors after delivery and also best practices on how to turn new findings into additional security requirements, as well as how to initiate another iteration to resolve these issues.

Section 1: Software Assurance Assessment Questionnaire

1. Governance

1.1. What is the process to develop a new security policy for the project? How are tailored security policies reviewed and evaluated for relevancy and accuracy prior to approval? Once approved, how often are they reviewed?

1.2. How does your risk management plan address security?

1.3. Does a software assurance plan exist? How is it maintained for accuracy and relevancy?

1.4. Has a security data classification scheme, like a security classification guide, been developed? How is it communicated?

1.5. How is software security and compliance language incorporated into vendor contracts to follow the organization's security policies? How is vendor compliance to internal policies enforced?

1.6. What software security training is required for team members? How is participation enforced? Is there organizational support to encourage job-specific training and certifications for team members?

1.7. How is continuance of Authorization to Operate (ATO) ensured in the project?

2. Requirements

2.1. Have security stakeholders been identified for different phases of the lifecycle? What roles are assigned?

2.2. Will all stakeholders be involved and present throughout the SDLC stages?

2.3. How are potential security concerns documented and maintained? How are potential security concerns turned into security requirements?

2.4. How are threats and abuse cases defined, modeled, prioritized, and incorporated into requirements?

2.5. How are security requirements identified, prioritized, assigned, captured, and evaluated?

2.6. How are security requirements tracked to verify and validate that they are satisfied through the phases of the lifecycle?

3. Architecture and Design

3.1. How are security principles considered and incorporated into the system's design and architecture?

3.2. How is your software architecture analyzed against security features and attack models?

3.3. Have software vulnerabilities and risks been identified, prioritized, categorized, and mitigated in the software design analysis (architecture review)?

3.4. How are trust boundaries clearly identified and documented?

3.5. How do system components and subsystems incorporate isolation and defense-in-depth principles?

3.6. When subject to failure and/or compromise, what processes ensure that the systems gracefully degrade and change states from full functionality to minimum essential functionality?

3.7. How does the embedded system design include cryptographic chip design methodologies? If they are not included, what are the tradeoffs and risks identified, and how are they mitigated in the event of a failure?

3.8. Have all stakeholders reviewed, modified, and finalized (with signed approval) the design documentation for the target systems?

4. Development

4.1. Is software developed in compliance with mandated regulations and standards?

4.2. How is software developed and verified against the design documents?

4.3. What policies are used to define third-party software and tool usage? How are third-party components considered and evaluated for potential threats?

4.4. Is there a mitigation or monitoring strategy that focuses on third-party software without vendor support?

4.5. How is instrumentation added during the Development Phase to ensure continuous monitoring capabilities in the system?

4.6. How is stakeholder feedback tracked and integrated into software development?

4.7. What metrics are collected to provide historical security trends? How are metrics used to improve the overall lifecycle?

4.8. What development practices are being used to assure quality of the code and software?

4.9. Does the project perform informal and formal code reviews/audits? What categories or security features do code reviews identify? How often are they performed? How are weaknesses categorized and prioritized? How are code reviews enforced? What is the process to approve code review results?

4.10. How are static and dynamic analysis tools used throughout development? How is code coverage analysis used to increase security?

4.11. How are compiler warnings addressed and tracked? Are security tools and compilers set to use the highest or strictest security settings available?

4.12. How is input validation handled? How is compliance enforced?

4.13. Are all data exchanges sanitized to prevent disclosure, ensure isolation, and protect against malformed inputs (intentional and unintentional) between systems? If so, how is compliance enforced?

4.14. How are secure coding standards used? How are standards enforced throughout development?

4.15. How are unit tests developed to check use and abuse cases (edge, boundary, and fuzz testing)? How often are they executed?

4.16. How are security trust boundaries and the principle of least privilege enforced in code development?

4.17. What is the process for identifying risks introduced by software reuse? How are risks incorporated into requirements and mitigated throughout the lifecycle?

4.18. Are all components and access authorizations implemented with default deny, requiring explicit permissions?

4.19. How are Agile practices incorporated into the project?

5. Test

5.1. How are test cases verified and validated against security requirements? How is this process automated and documented?

5.2. How are software and security-related items protected and re-evaluated for changes in protection based on configuration?

5.3. How are security techniques like fault tolerance and redundancy enforced in the system?

5.4. How are fuzz and penetration tests used to close security gaps in the project?

5.5. What techniques are used to ensure integrity of the software? What tests are used to verify that such techniques do not degrade system or mission performance?

5.6. How are third-party software and libraries or tools tested prior to use?

5.7. How are security updates managed for third-party software components?

5.8. How is environment parity ensured throughout the system lifecycle?

5.9. As new integration cycles occur with updated code, are security-related tests repeated?

5.10. What is the process to ensure that tests are repeated and analyzed on future iterations or deployments?

5.11. If your organization uses an embedded software pipeline, how is the software tested?

5.12. How is code execution being monitored on the target system to identify potential security requirements?

5.13. How are security flaws in the Test Phase documented and tracked through resolution? Is there a reporting mechanism for end users to identify any concerns, errors, or problems?

5.14. Has a select group of end users been allowed to use the deployed software on the target system? Are they encouraged to use the system in an incorrect manner to identify flaws? How are new security concerns integrated into system requirements and addressed?

5.15. Are external or independent test teams utilized in the project for verifying and validating the system and its security? How do they report findings and issue resolution to the rest of the team?

6. Delivery

6.1. Were stakeholders briefed on all addressed security requirements?

6.2. Have the customers been made aware of who to contact in the event of discovering a security concern? If the customer reports a security concern, is there a formal process to review why the concern was not identified or addressed in earlier phases?

6.3. How are latent defects handled when discovered? What is the process for handling software errors whose cause cannot be determined?

6.4. How are security-related lessons learned captured and reviewed? Has a post-transition meeting been scheduled to improve future performance?

6.5. Is training provided for users based on operation and instruction manuals? Who is responsible for delivering the training?

6.6. Is there a formal process for reviewing manuals and system instructions for accuracy? How are feature updates and changes communicated to end users?

6.7. What is the process for implementing continuous monitoring? How are continuous monitoring strategies evaluated for effectiveness and usefulness in operations? How are anomalies handled?

6.8. What is the process for incident response? How are stakeholders informed of security-related incidents?

6.9. What is the program strategy for software decommissioning during the disposal phase of the system development lifecycle?

Section 2: Questionnaire Guidance

1. Governance

No.	Guidance	Security Activities
1.1	Security policies define the behaviors necessary to maintain a secure system, organization, or environment and ensure that procedures are put in place to deter and stop attackers. They should be tailored to fit a program's development and operational environment. Develop new security policies by identifying existing similar organizational policies and other program policies that may be available, along with mandatory regulations and standards the program is required to follow. BSIMM 8 CP1.3, "Create Policy," highlights the importance of establishing a security policy to ensure a "unified approach for satisfying the (potentially lengthy) list of security drivers at the governance	Policy and governance

No.	Guidance	Security Activities
	level."[1] Specific details to create tailored policies can be found in OpenSAMM PC 2.A, "Build policies and standards for security compliance."[2] Program managers should consider all industry and government regulations and standards related to the program. Sometimes conflicts may arise between regulations and their implementation procedures. Conflicts should be identified, documented, and resolved through an adjudication process. BSIMM8 CP1.1, "Unify regulatory pressures," gives details on the importance of a formal approach to identifying compliance policies.[1] All stakeholders should be briefed on tradeoffs and risks that will be accepted by conforming to regulations and standards levied on the program. A thorough review of the security policies should be conducted to guarantee policies cover both acceptable and unacceptable behaviors that are relevant to each program. Once a security policy is reviewed and approved, a schedule to periodically review and update the policy as needed should be established. Security policies should also be communicated to all personnel, enforced through an agreement, and kept in a central tracking system that is accessible to all personnel for reference.	
1.2	All programs should have a risk management plan. Department of Defense Instruction (DoDI) 5000.02, *Operation of the Defense Acquisition System*, requires program managers to address risk management in a way that is "proactive and should be focused on the actions that will be taken and resources that will be allocated to reduce both the likelihood and consequences of risks being realized. Effective risk management is not just risk identification and tracking."[3] Proactive measures can be taken to reduce risk and potential harm to a program. A common mandated approach in acquisition is the Risk Management Framework (RMF).[4] RMF seeks to effectively manage risk with six steps in a cycle. As the government begins to shift in development paradigms to a more iterative approach, programs should consider utilizing the *Continuous Risk Management Guidebook*,[5] developed by the SEI, which provides a reference for lessons learned and implementation guidance for continuous risk management. Items to consider when developing a risk management plan include the following: • How are risks prioritized, categorized, tracked, and mitigated?	Organizational risk factors assessment

No.	Guidance	Security Activities
	• Are risks assessed periodically throughout the SDLC? How are risks found in later lifecycle phases prioritized, categorized, tracked, and mitigated? • What is the process for performing a risk assessment on third-party software? Does the assessment include consequences and likelihood associated with common issues such as obsolescence or unintentional malware? • Do service-level agreements (SLAs), contract agreements, and other related documents incorporate requirements for meeting criteria set in the risk management plan?	
1.3	A Software Assurance Plan identifies steps the team will take to ensure "the level of confidence that software functions as intended and is free from vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the lifecycle" [Definition of Software Assurance, P.L. 112-239 § 933, 2013].[6] Sometimes, the Software Assurance Plan is a section in the overarching Program Protection Plan (PPP) document. Whether it is a section inside of the PPP or its own document, the Software Assurance Plan should include countermeasures for preventing vulnerabilities from persisting within the systems. Security-critical systems and components should be identified and categorized based on a clear definition that establishes what makes a system or component security critical. Evaluations of those components and systems should be conducted against Common Vulnerability Enumeration (CVE®),[7] Common Weakness Enumeration (CWE™),[8] Common Attack Pattern Enumeration and Classification (CAPEC™),[9] and CERT Rules and Recommendations.[10] Resolutions or mitigation strategies for security components and systems that have weaknesses present should be documented in the Software Assurance Plan. Throughout the lifecycle, it is important to reference the Software Assurance Plan to provide confidence that the steps documented are being implemented, are accurate and relevant to appropriately avoid vulnerabilities, and do not negatively affect mission or system performance.	Policy and governance

® CVE is a registered trademark of The MITRE Corporation. The CVE is a list of entries for publicly known cybersecurity vulnerabilities.

™ CWE is a registered trademark of The MITRE Corporation. The CWE is a community-developed list of common software security weaknesses.

™ CAPEC is a registered trademark of The MITRE Corporation. The CAPEC provides a catalog of common attack patterns that helps users understand how adversaries exploit weaknesses in cyber-enabled capabilities.

No.	Guidance	Security Activities
1.4	Security Classification Guides (SCGs) are established to standardize the protection of information about a program, system, or other entity. Executive Order 13526 specifies that agencies should have controls in place to ensure data is secure.[11] SCGs should provide guidance on the aggregation of seemingly harmless data that may result in a higher classification assignment. SCGs should also establish declassification timelines. Similar controls should be in place for defining and communicating requirements for sensitive but unclassified information (e.g., Controlled Unclassified Information [CUI], Covered Defense Information). Both DFARS 252.204-7012, "Safeguarding Covered Defense Information and Cyber Incident Reporting,"[12] and NIST 800-171 Rev. 2, "Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations,"[13] explain rules that could aid in developing a classification guide. SCGs should be accessible to all stakeholders, known throughout the program, and communicated with external partners to prevent security breaches or information leaks.	Education and security guidance
1.5	Compliance with organizational security policies should be passed down to contractors and subcontractors. Contracts with vendors and other service providers should include language that compels the contractors and subcontractors to meet the program's software assurance goals and requirements. Such language should include measures taken to reduce risk, avoid malicious software, ensure valid testing, and plan to manage oversight of software assurance-related processes. Software assurance should be considered, at minimum, in Request for Proposal (RFP) preparation, contract negotiations, software and data rights[14] discussions, and acceptance of contract deliverables. Templates with standard software security language should be developed to ensure that the basic minimum information necessary to acquire, develop, or maintain a system and its components have been met. These templates will serve as a baseline in contracts or other related documents for all programs with similar software security needs or functions. Security language templates can be tailored to include more specific security requirements and provide a minimum standard of software assurance across the entire product baseline and various similar programs. USAF SSE Acquisition Language Guidebook Section 2.5.1 describes security-related sample statements that should be included in contracts to ensure that contractors meet the SwA goals and requirements.[15]	Policy and governance

No.	Guidance	Security Activities
1.6	Training is an important part of the SDLC. Team members should be trained and/or experienced in the techniques that are necessary to implement security protections and controls. Security training should be encouraged beyond the DoD Annual Cyber requirements and include role-specific training.[16] Team members should develop a grasp of how to use tools available through training offerings. Process improvements and more secure and resilient systems are a few of the benefits to the program when members are encouraged to seek training and apply the information learned to the system or environment. Incentives should be provided to members to obtain and maintain training and certifications that are relevant to their position.	Education and security guidance
1.7	In order to adopt Agile or DevOps in the DoD, one must ensure that continuance of ATO is granted so teams can go through sprints more easily. The latest efforts that have been granted continuance of ATO (e.g., Kessel Run Lab) show some common traits—described below—that make it easier for an authorizing official (AO) to approve different iterations of development. While evaluating the ATO process for any given project, it is fundamental to understand the environment under which it is issued. Ensuring that existing dependencies or constraints do not change across iterations, or at least keeping the variability of that environment to a minimum, will help speed up the approval of each phase. A contingency plan should be put in place, in case disruptive but necessary changes need to be implemented. In this case, teams must be able to work with the AO to describe in detail how the changes affect the overall system design and interactions with other components. By isolating the effects of the change and providing a rationale to the AO, teams can provide the AO with situational awareness and speed up the ATO process.	Policy and governance

2. Requirements

No.	Guidance	Security Activities
2.1	Stakeholders should be identified to take responsibility for specific roles on the team and in the project. Traditionally, programs don't consider executive and user stakeholders for all phases of the software	Security stories

No.	Guidance	Security Activities
	development lifecycle. These stakeholders should minimally have responsibility for verifying that their requirements have been met. For each lifecycle phase, a team member should take ownership of ensuring that security requirements and controls are implemented and any changes or issues are resolved and documented.	
2.2	Stakeholders should have a vested interest in the program and be informed and available throughout the lifecycle. Information should be shared up front and early when it is available so that stakeholders can have a clear understanding of the security posture of the systems, components, and environment. Meetings should be scheduled with stakeholders to address the progress of fulfilling and validating security requirements.	Security stories
2.3	A process should exist to identify and address security concerns that may arise throughout the SDLC. The process to identify security concerns should include documenting the concern along with the areas the security concern affects. Every stakeholder for the program should be able to highlight potential security concerns to be addressed within the development of the system. Security concerns should be documented and maintained to ensure traceability. The architecture of the target system should be reviewed at each iteration for new potential security concerns, and those concerns should be turned into requirements. Developers should have a documented formal process to update or introduce new security requirements during development to ensure that any security concerns identified, even in later stages of the lifecycle, are properly addressed in the system. Security concerns should be documented, turned into security requirements, and entered into the cycle of requirements management.	Security requirements (SFR/SAR) generation
2.4	Identifying potential threats and abuse cases is essential to ensuring that the systems, components, and operating environment are secure. The SEI has studied multiple threat modeling methods and encourages a hybrid approach to identify potential threats.[17] An assessment should be completed that identifies, prioritizes, and models threats and abuse cases. A standard criterion for prioritizing threats should be established and documented to give insight into the most important models or cases. Attack patterns and surfaces should also be incorporated into security requirements to ensure that protections are in place to prevent the potential threats from becoming real. Additional mission thread	Threat modeling Abuse case development

No.	Guidance	Security Activities
	analysis should be performed, and the results should be used to influence security requirements after prioritization.	
2.5	Requirements gathering is an important step in the acquisition and development lifecycles. The requirements are the first step in determining the needs and functionality of a system or service. Regulations and standards should be incorporated into requirements. Initial use cases and any threat assessments should influence security requirements. Software security requirements and policies should be present in the requirements documentation and address things including but not limited to • encryption • specific protocols to use • auditing • access control • management and user-separated accessibility • proper handling of personally identifiable information (PII), CUI, and other sensitive/classified data Security requirements should be prioritized and assigned to a team member to ensure there is accountability and verification that the requirements have been met. A mechanism, such as a Requirements Management System, should be used to provide requirements traceability throughout the lifecycle. Any tradeoff decisions or changes in implementation of the requirement should be easily traced back to the original requirement. Be sure to incorporate organization security policies in security requirements. Security requirements that apply to contractors or vendors should also be identified, prioritized, and captured for tracking and verification. Once a final draft of requirements is completed, all team members and customers should have a chance to review and evaluate the requirements. Requirements should be approved by stakeholders to ensure support and show consensus on what needs to be done up front and early. The results of the evaluation, whether approved or rejected, should be tracked in the requirements management system. There should be a formal process for future change requests to the requirements after they have been approved.	Security requirements (SFR/SAR) generation

No.	Guidance	Security Activities
2.6	Each security requirement should have a stakeholder that takes ownership for verification and validation (V&V) throughout the lifecycle. Requirements should be integrated into test cases, and test cases should be approved by all stakeholders prior to implementing tests. Techniques and resources used to satisfy the requirements should be documented and tested to ensure that the requirements are met. All successes and failures in V&V should be documented and communicated to the team and stakeholders to promote awareness. All security requirements should be met by vendors and contractors, and a plan for V&V should be completed to make sure requirements are satisfied.	Security requirements (SFR/SAR) generation

3. Architecture and Design

No.	Guidance	Security Activities
3.1	Architecture and design are the bridges between requirements and implementation. The software architecture of a program or computing system is a depiction of the system that aids in understanding how the system will behave.[18] Security should be incorporated into the design and architecture of the system and development environment. Considerations for hardware, software, and interface components should include protections against intentional and unintentional defects that could possibly be introduced. An analysis of cybersecurity trade-offs should be completed against the architecture and design concepts to help identify weaknesses and implement more robust countermeasures.	Security architecture design
3.2	Threat assessments are important when developing the software architecture and design. Attack models should be developed using the requirements. These models should influence the selection of a secure design environment, the software architecture, and the best practices used for software development. Mitigations for the threats in the models should be analyzed for efficiency and effectiveness and incorporated into the design. Software architects (a contractor or internal team member) should use threat modeling to ensure that cybersecurity efforts consider threats and attack patterns when making design decisions.	Attack surface analysis
3.3	Architectural risks are potential vulnerabilities of the system and its environment that may arise when the system is implemented. Risk	Architectural risk analysis

No.	Guidance	Security Activities
	analysis should be performed on the architecture and design to identify vulnerabilities and plan for mitigation or use of alternative solutions. Risks need to be prioritized and categorized by severity and affected areas. The categorization should help in identifying root causes or resolutions to mitigate such risks. Throughout the lifecycle, the architecture and design should be evaluated and validated for effectiveness and efficiency. Software architecture analysis (SAA) is the process of identifying flaws in the software design and its development environment prior to actual development. An architectural analysis should be performed to identify vulnerabilities in the design. Vulnerabilities should be categorized based on criteria such as affected area or severity. Each category of vulnerability should be prioritized and addressed to reduce the risk and prevent the vulnerability. Failure Mode, Effects, and Criticality Analysis (FMECA) is a method of analysis based on failure analysis and identification of root causes and threads that lead to failure modes. Programs should strongly encourage the use of FMECA as one method for design analysis. Performing FMECA for embedded systems is essential in ensuring security of components. MIL Handbook 502A provides guidance for implementing FMECA within the DoD.[19] The handbook helps provide better insight into the cost and performance of software and hardware products.	
3.4	Trust boundaries are the areas within the architecture where some level of trust has been established. The boundary is often placed between trusted components (subsystems) and an untrusted source. DoDI 8540.01 (2015) establishes a cross-domain policy, which "assigns responsibilities, and identifies procedures for the interconnection of information systems."[20] Software trust is usually centered on data confidentiality and integrity and includes different levels of classification. Trust boundaries should be identified and clearly marked in design documentation.	Security design requirements generation
3.5	Defense in depth is the concept of having multiple layers of security so that if one security control fails there is already another security measure in place to prevent an attack. Isolation reduces the attack surface by limiting access to critical components through untrusted means. Isolation is the practice of separating duties or data control flows to only use the parts of the system required to complete a particular function.	Security design requirements generation

No.	Guidance	Security Activities
	Both techniques, when implemented together and incorporated into the design of the software and architecture, will provide greater protection of the system. Programs should use virtualization or containers where appropriate to provide redundancy and fast recovery, should an attack occur.	
3.6	In case of an attack or system failure, the system will often need to be operational despite being exploited. A recovery plan should be developed that includes the process to ensure the system is able to run in a mode that provides minimum essential functionality. Abuse cases as well as threat modeling should be integrated into the development of how to successfully degrade the system to its appropriate level of functionality. If the system has implemented isolation, those measures should be referenced when developing the recovery plan. Events that are not related to an attack or exploitation may also occur and will need to be considered when developing the plan. Events such as a component failure or powering down, and possible changes in operational environment affecting the system, should also be considered when the program identifies the process for graceful degradation of the system.	Attack surface analysis
3.7	The embedded systems design should include cryptographic functionality to ensure that components are secure. For example, single-chip cryptographic (SCC) design is a methodology used to isolate functionality in processor components where security cannot be guaranteed. In embedded systems, the SCC design ensures security of authentication keys and related mission-critical information. The design documentation should include the use of SCC design to provide an additional layer of security in the event of failure.[16] SCC design should be included in the architecture analysis. Tradeoffs and risks should be identified and mitigated prior to implementation.	Security design requirements generation
3.8	The program should set up a formal process for approval of design documentation and its related analysis assessments. The process should address what is done when items are rejected and accepted. All stakeholders should review, modify, and finalize all architecture and design documentation prior to formal reviews, such as the Preliminary Design Review (PDR) and the Critical Design Review (CDR). In a more iterative approach, stakeholder awareness is important so decisions can be made early and up front, prior to more critical points in the lifecycle. Exposures, vulnerabilities, and threats (EVTs) should be re-	Security design review

No.	Guidance	Security Activities
	evaluated when changes to the architecture and design are made. Each configuration should periodically evaluate EVTs for changes in requirements, architecture and design, and development and operational environments.[16] In addition, the approval process should be documented to ensure uniform implementation and include how modifications after initial approval will be addressed.	

4. Development

No.	Guidance	Security Activities
4.1	Mandated regulations and standards are rules that must be followed and implemented within the program. Identifying all applicable regulations and standards is an important step that must be completed prior to implementation. The software design should ensure all regulations are verified so that the software complies with each rule in the standards and regulations. The rules outlined in the standards and regulations should be included in the requirements documentation and flowed into the design documents. Software should be developed to fulfill the requirements and tie closely to the design documentation. Changes that may arise in the software design should be fed back into the requirements process to provide traceability and also ensure that compliance is still met for the changes made.	Traceability analysis
4.2	Software should be developed to implement specifications listed in the design documents. Any code written should be verifiable against those designs. Code reviews and test cases can be used to verify the software against the design. Secure design practices should include security controls and analysis of threat models and abuse cases.[16] Throughout the SDLC, as new functionality is introduced or fixed, software should be verified to implement secure design principles and controls. System components and subsystems should be verified and validated in isolation against the design prior to integration and V&V against the complete software architecture.	Traceability analysis
4.3	Any third-party software used as a tool to develop a system, or as a dependency or library to the application, should be verified and validated to prevent software vulnerabilities and unnecessary redundancy.	Dependency monitoring

No.	Guidance	Security Activities
	Third-party software should always be evaluated to ensure information security and should be periodically tracked to ensure continual relevancy to the system and its functionality. Integrated circuit suppliers should be accredited prior to use. All vendors and suppliers should be trusted foundry/trusted suppliers approved or accredited by the Defense Microelectronics Agency (DMEA) or National Security Agency (NSA) for acquisition or development of any information and communications technology components (e.g., application-specific integrated circuits). Deliverables should be certified via the National Information Assurance Partnership (NIAP) Common Criteria Evaluation and Validation Scheme (CCEVS) process. There should be a documented process for assigning evaluation assurance levels on commercial off-the-shelf (COTS) deliverables. COTS products should be validated via CCEVS for version updates or modifications. Vendors should be required to submit Certificates of Conformance ensuring that parts are not counterfeit. V&V tests for acceptance should be approved by the government if software components are vendor- or contractor-produced and methods for acceptance should be documented.[16]	
4.4	Third-party software carries an additional risk of obsolescence. There should be a formal process for ensuring that software will be vendor-supported when it is considered for use. There should also be a plan for identifying software components that may not be supported currently or in the future. If third-party software will be used despite not being vendor-supported, there should be an additional risk assessment performed and a formal stakeholder acceptance record to the risks and mitigations posed. Programs should strive to be proactive toward obsolescence, consider data rights where appropriate, and keep track of software versions or competitive alternatives. Throughout the lifecycle, developers and program managers should maintain a clear traceability of software versions, licensing, and software release trends for third-party software components.	Dependency monitoring
4.5	In order to extract useful runtime information from the code during debugging, it is necessary to introduce instrumentation in the code at the same time it is generated and while the ideas about what is important about that specific piece are still fresh within the team. Once that instrumentation is in place, enabling the output can be easily done	Traceability analysis

No.	Guidance	Security Activities
	via configuration by any of the parties executing the testing. This information—when properly accumulated and displayed in a monitoring environment—can quickly provide meaningful insights that, in case of failure, will allow for a faster recovery.	
4.6	Throughout the phases of the software development lifecycle, stakeholder feedback should be collected and integrated into software development. Often, security requirements may change as new technologies and changes in operational environments are introduced. There should be a formal process for reviewing, prioritizing, and incorporating stakeholder feedback for iterations of the software. Whenever the feedback introduces changes to the overall system design, these modifications should be properly captured into new versions of the requirements, so they are properly traced and tested further on in the SDLC.	Security stories
4.7	In Agile, metrics are used to express the operational performance of the software and, consequently, how to improve it. Metrics such as the examples below are measurable aspects of the SDLC that provide insights about the overall system security: • Software assurance requirements findings count • attack vector details (IP, stack trace, time, rate of attack, etc.) • time to approval • time to patch vulnerabilities • mean time to recovery • mean time to detection Metrics should be carefully chosen, as numbers that are representative to what is happening throughout the SDLC. As an example, metrics related to development can show areas of more activity in the codebase, which can indicate issues that are tougher to resolve or features that ended up taking longer to implement. Heightened activity can be an indicator of where most of the new code is, so testing can focus more on those areas from an early stage. Also, lessons learned from previous increments, sprints, or cycles should be documented and referenced so future iterations can implement improvements and verify their efficacy based on the metrics being analyzed.	Traceability analysis

No.	Guidance	Security Activities
4.8	During development, some practices can help a team achieve better code/software quality. Among them are • documentation: Comments and documentation linking code to requirements explain the purpose of each section of the software and must state what is expected as input and output, with details on how that is achieved and boundary conditions that may affect it. • code reviews: Explained in detail in Item 4.4 • continuous integration: A set of practices with the goal of promoting better integration of parts of software written by a development team. These practices include – maintaining a code repository – automating the build – making the build self-testing – building the software at every commit to the baseline – keeping the build fast – testing in a clone of the production environment – making it easy to obtain the latest deliverable • quality assurance: Executes testing that is tightly connected to the software requirements. Each requirement should have a corresponding test that gets executed at the end of each software build to make sure that it is correctly satisfied.	Traceability analysis
4.9	Reviews should be conducted within the Development Phase to identify flaws in the software code. Formal reviews are a more structured way to review aspects of the code. Roles and responsibilities are defined clearly and materials are often distributed prior to a formal review to facilitate organization and order during the review. When defects are discovered in a formal review, they are usually recorded in great detail, including their location, severity, type, and other relevant details. All of this information may be used later to guide the quality assurance (QA) team. Alternatively, an informal review does not require as much planning or organization. Informal reviews can be over-the-shoulder feedback, where the developer explains the code to another team member who can identify flaws in code logic, semantics, and bad programming practices, or pair programming, which employs the use of programming and reviewing in parallel.[21]	Security-focused code review

No.	Guidance	Security Activities
	Formal reviews should be conducted on every build or iteration, highlighting changes that have occurred since the last formal review. Informal reviews are more ad hoc and can occur at the convenience of the developers. Both formal and informal reviews should be used to provide the maximum manual coverage to the software code. Code reviews should be a part of the activities practiced throughout the project and should be enforced by technical and team leads. This practice ensures that small bugs are caught early in the lifecycle and are dealt with at a low cost. When code reviews are related to or addressing security requirements, it is essential to involve security stakeholders in developer-led reviews. Projects should have a process to maintain security stakeholders' involvement throughout the lifecycle. For security-related reviews, attention should be paid to the security requirements, and there should be a record of how these are directly addressed by features in the code. When participants are focused on the security aspects of the code, they can start by verifying that secure coding best practices are properly applied and that parts of the code architecture or implementation do not create any security vulnerabilities for the overall system. If security vulnerabilities are identified during code reviews, they should be prioritized, categorized, and resolved based on the functional area that is affected by the vulnerability.	
4.10	Static analysis and dynamic analysis are often used to find bugs and flaws in software code. Static analysis is performed on source or binary code, usually without the need to run the program. Static analysis tools can work with many different programming languages to identify flaws and can sometimes allow definitions of custom rules in their configuration. Implementing static analysis methods is highly encouraged for use in early bug and flaw detection and has been known to reduce development costs. Aside from manual static analysis, such as code inspection, automated tools should be used as an additional flaw detection method. Tools in this category may be able to find • redundant code • dead code • logic flaws Dynamic analysis requires the program to operate in order to monitor its functionality within the runtime environment. Dynamic analysis is useful for discovering complex vulnerabilities and performance metrics that cannot be easily detected with static analysis. Although they	Static code analysis

No.	Guidance	Security Activities
	are usually more specific to a particular programming language or operating system, dynamic analysis tools can also rapidly find flaws or vulnerabilities in both the code and program functionality.[22] Tools in this category may be able to detect • integer/buffer overflows • memory leaks and patterns of bad memory use or allocation Dynamic analysis tools may also simulate attacks by malicious individuals to find unexpected vulnerabilities that may be missed by other testing techniques. Code coverage is the percentage of how in depth the analyses test the developed code. For full code coverage, test cases should execute every line of executable code and follow every path or branch within a functional control flow. Full code coverage is an ideal scenario for testing the development code. Many tools are used to provide metrics on code coverage by running automated tests and injecting tracing calls to identify control flows.[23] From a security perspective, at every development iteration, it makes sense to focus on uncovered code that recently had one or more security problems to take advantage of any coverage inequity. While code coverage may be a good metric of how much testing is being completed, it is not necessarily a good metric of how *well* the testing is working. Other metrics should be used along with code coverage to ensure quality (See Item 4.1).	
4.11	During the process of building software, compiler errors are usually blockers, which show up before the software can be tested or released. These errors can be caused by simple issues like mistyped statements, missing files, or more complex issues, like logical problems in the code structure. Since the errors are blockers, they are always dealt with almost immediately. Compiler warnings, on the other hand, are sometimes ignored and considered a very low threat. Most compilers can be set to use the highest warning level available, in order to provide more meaningful information. This information can be used to verify if the warnings are connected to some kind of security flaw. The development team can also make use of static and dynamic analysis tools to detect and eliminate additional security flaws. It is always good practice to eliminate	Traceability analysis

No.	Guidance	Security Activities
	all warnings in the most efficient and secure way to reduce any possibility of introducing security gaps in the software.	
4.12	Input from all untrusted data sources must be validated. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files. In user interfaces, or anywhere human interaction is allowed, there should be checks for wrong data types, default null selections, or special characters. All of these can come from accidental input or malicious activity to test how the UI responds or breaks and provides access to the system in case of errors. Validation should always include • data type verification, so only the accepted values are able to get through • proper character validation, with filtering applied in user interfaces • boundary condition checks in APIs and user interfaces (numeric ranges, zeros, large integers, null values) In order to enforce compliance, teams must ensure testing includes a wide variety of conditions that are applied against the system's interfaces. This would expose issues with validation as soon as the code is committed into the repository.	Secure coding practices
4.13	Data sanitization involves removing or encrypting data that—if exposed—could cause danger for the system or to the overall mission. All data should be sanitized prior to transmission, storage, or use by any component of the system. Data should also be validated and verified for integrity. The methods for using validated data as well as how to handle data deemed malicious or malformed should be thoroughly documented. Libraries that perform such verifications should be imported or implemented in development. Risks for malformed data should be controlled through proper error handling, integrity checks, and data validation. Use of encryption techniques should be evaluated against NSA requirements for certification and approved prior to use. Before a system uses data from external sources, the data should be verified to be valid and checked for integrity. NIST Risk Management	Secure coding practices

No.	Guidance	Security Activities
	Framework Control SI-7 highlights the importance of including security and integrity checks for information systems.[4] Checks should be applied to both external and internal data to the system. There should be a documented process for verifying and validating data from other system components as well as external sources.	
4.14	Secure coding standards should be used in software projects, either to train developers on the secure coding standards chosen by the organization or to use automated checks or code reviews that ensure the utilization of those standards. Performing an automated check later is always cost-effective but may not effectively catch all types of problems. In order to enforce the use of secure coding standards, team, technical, and integrated product team (IPT) leads should be aware of their utilization and also ensure that their teams have received training and know in which scenarios to use them. Using a team exclusively to review the produced code turns out to be extremely expensive and may add delays to the development. In any case, both automated checks and code reviews are important to ensure the security of the developed system. An organization should develop its own tailored secure coding standards, based on widely accepted ones (e.g., CERT's Secure Coding Standards10), which should contain prohibited practices as well as required ones.	Secure coding practices
4.15	Unit tests are tests that involve the smallest unit of code, module, or procedure necessary to verify that the code has specific functionality and to validate the functionality against the design. MIL-STD-498 Sections 5.7 and 5.8 describe requirements for developing and implementing unit tests and unit integrations tests.[24] Use and abuse cases should be integrated into unit tests.	Security unit testing
4.16	The principle of least privilege (PoLP) is the methodology of providing the minimum amount of authority for an entity to perform a specific function. Implementing the PoLP helps to protect the system against unauthorized actions being performed. When paired with audit logging, employing PoLP can help identify potential attacks that try to circumvent authorization security policies. Trust boundaries are logical separations of data classification as well as levels of trust. Systems usually trust internally created data, and therefore their designers may choose whether to apply more strict	Secure coding practices

No.	Guidance	Security Activities
	security measures on such data. Alternatively, systems usually distrust externally created data, which requires extra validation to be performed on such data sets. The PoLP and trust boundaries should be properly considered in the design and later implemented in code development. Access control lists, integrity checks using encryption or hashing, and group policies are some of the artifacts that can be used to enforce PoLP and trust boundaries in the architecture.	
4.17	Software reuse can reduce time and costs associated with developing new software. Software should be developed using techniques that encourage reuse, such as object-oriented programming. Whenever possible, software should be considered for reuse. The considered piece of software should be deemed relevant for the evaluated scenarios and fit within the new system's architecture. A risk analysis for reusing software should be completed, and risks should be identified and mitigated prior to integrating reused software components into the architecture. Common risks associated with software reuse are programming language compatibility, version control management, and additional security vulnerabilities that may be introduced.	Secure coding practices
4.18	Any piece of code that depends on authentication to execute its function should operate based on the default-deny principle. The default-deny principle bases access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted. Under these principles, any access that cannot be verified as authorized will make the code fail. All the possible exception cases should be handled gracefully, directing users to a point where they can either return to the previous screen without any substantial data loss (e.g., directed back to form, but form data after submission attempt is lost) or to a point where they are able to provide the required credentials and complete the operation.	Secure coding practices
4.19	Even though Agile has been around for a while, its application to DoD projects is very recent, having started with the Agile mandate from 2012.[25] Plain Agile has been developed with industry practices in mind and does not account for all the barriers that the DoD	Traceability analysis

No.	Guidance	Security Activities
	environment has, such as different access or clearance levels, air-gapped systems and difficulties in obtaining ATO. However, Agile can provide the same industry benefits to DoD projects, and some adaptations have been developed to make it feasible for the DoD use. Back in 2010, MITRE produced a technical report, *Handbook for Implementing Agile in Department of Defense Information Technology Acquisition*,[26] which provides useful information on how Agile can be used in DoD projects. More recently, the SEI has produced the "Acquisition & Management Concerns for Agile Use in Government Series," a series of six brochures about Agile in the government, which has a more modern take on the same subject. The brochures are listed below: • Agile Development and DoD Acquisitions • Agile Culture in the DoD • Management and Contracting Practices for Agile Programs • Agile Acquisition and Milestone Reviews • Estimating in Agile Acquisition • Adopting Agile in DoD IT Acquisitions	

5. Test

No.	Guidance	Security Activities
5.1	Early in the SDLC, when requirements are captured and approved by stakeholders, test cases are defined based on the requirements. Test cases should verify and validate that the security requirements are implemented. While the code is being written, unit tests should be written in parallel, matching the security requirements related to that functional piece of the software. The execution of the unit tests should be automated and run every time the software is built, along with other security tests related to the application environment (e.g., web application, client-server application, stand-alone). This process aims to speed up the fixes and have them done at the lowest possible cost. There should also be documentation about the security test cases, explaining the rationale behind the test and any additional technical details that make it easier for anybody to understand why and how it works, should any issues appear.	Security test planning

No.	Guidance	Security Activities
	One important aspect of running tests is the automation of this process. Over the years, it has been proven that automating tests eliminates the variability introduced by humans—the differences range from environment configuration settings to language versions, plugins loaded, and more—and makes sure that the parity across environments works in the team's favor.[27] In order to have this automation implemented, teams can use artifacts as simple as scripts connected to version control system servers or build requests, or a more sophisticated continuous integration product running to coordinate commits, builds, and releases for the team. Many options are available, from free open-source software to commercially supported products.	
5.2	NIST RMF Control CM-2, Baseline Configuration, states that programs should create baseline configurations to "serve as a basis for future builds, releases, and/or changes to information systems."[28] In an environment where systems of systems include reused software and hardware components, programs should be vigilant in testing different system configurations. Each component interface provides its own set of security concerns and incompatibilities. Additional security tests should be created against the requirements that led to the configuration. As different components are used together to create new configurations or baselines, security tests should be repeated to identify new flaws that may be introduced. NIST RMF Control SA-10, Developer Configuration Management, highlights the importance of maintaining traceability of component changes in each baseline configuration and safeguarding against security vulnerabilities that may be present.	Verification of process and procedures
5.3	Fault tolerance allows a system to continue to operate despite some of its components failing. Fault tolerance is critical to high-availability systems to ensure that minimum functionality is provided at all times in security- and life-critical events. Systems should be designed to gracefully degrade in the event of a failure instead of completely shutting down and losing all functionality. Security requirements and test cases should include fault tolerance principles. Test cases should test for fault tolerance; stress tests paired with penetration tests provide details on the effectiveness of the system's tolerance. Additionally, redundancy solutions should be researched to provide a more fault-tolerant system environment. Redundancy usually includes some sort of backup to the system or its components. Due to various cost and performance drawbacks associated with a fully redundant system, it is often more feasible to provide redundancy for only the	Verification of security implementation

No.	Guidance	Security Activities
	most critical components of the system.[29] Each test case should validate the effectiveness of the redundancy solution chosen in the system. Fault tolerant and redundant solutions should also provide notifications and/or audit logs to ensure that operators understand there is a failure and can act when appropriate. Tests simulating system failure and checking for notifications should be included in the test plan. During the Requirements and Design Phases, the program manager should be sure the solutions that provide fault tolerance and redundancy are documented and justified. In securing mission-critical systems, the principles of fault tolerance and redundancy should be used in ways that allow graceful degradation, provide warnings to users and administrators about unauthorized access attempts, and even disable a service in case of failure to protect the network and the data that flows in it.	
5.4	In his document *Automated Penetration Testing with White-Box Fuzzing*, John Neystadt states that fuzzing, or fuzz testing, is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.[30] The program is then monitored for exceptions such as crashes or failing built-in code assertions and examined for potential memory leaks. Typically, fuzzers are used to test programs that take structured inputs. This structure is specified, such as in a file format or protocol, and distinguishes valid from invalid input. An effective fuzzer generates semi-valid inputs that are "valid enough" in that they are not directly rejected by the parser but do create unexpected behaviors deeper in the program and are "invalid enough" to expose corner cases that have not been properly dealt with.[30] Typically, a fuzzer is considered more effective if it achieves a higher degree of code coverage. The rationale is that if a fuzzer does not exercise certain structural elements in the program, then it is also not able to reveal bugs that are hiding in these elements. Some program elements are considered more critical than others. For instance, a division operator might cause a division by zero error, or a system call may crash the program. A black-box fuzzer treats the program as a black box and is unaware of internal program structure. A white-box fuzzer leverages program analysis to systematically increase code coverage or to reach certain critical program locations.	Security testing Fuzz testing

No.	Guidance	Security Activities
	Fuzzing is used mostly as an automated technique to expose vulnerabilities in security-critical programs that might be exploited with malicious intent. More generally, fuzzing is used to demonstrate the presence of bugs rather than their absence. For a software development team, having access to the code allows COTS fuzzing tools to learn about the code structure and then generate meaningful tests to determine what can introduce security gaps. Following the tests, the development team should be diligent in looking at the test reports and closing these gaps. According to the SANS Institute, a penetration test, also known as a pen test, "is the authorized, scheduled and systematic process of using known vulnerabilities in an attempt to perform an intrusion into host, network or application resources. The penetration test can be conducted on internal (a building access or host security system) or external (the company connection to the Internet) resources. It normally consists of using an automated or manual toolset to test company resources."[31] When "a vulnerability is utilized by an unauthorized individual to access company resources, company resources can be compromised. The objective [of] this kind of test is to address vulnerabilities before they can be utilized." During a penetration test, the core services offered by the company should be tested. These may include server resources, such as mail, DNS, firewall systems, password syntax, File Transfer Protocol (FTP) systems and Web servers, as well as other part of the network infrastructure. Recent information shows that wireless systems and Public Branch Exchange (PBX) systems should also be tested. Throughout the execution of these tests, logging routines and all kinds of intrusion detection systems should be enabled, in order to test their effectiveness against eventual attacks. In some occasions, firewalls and the most external protection layers can be temporarily turned off, so testers can find other security gaps which live in inner layers and that can also be exploited if the attacker is able to break through the first line of defense. Once everything is tested and corrections are applied, all the defense mechanisms must be re-enabled and the system can be made available to uses once again.	
5.5	When obtaining a software release, it is fundamental to make sure it is coming from a trustworthy source. Integrity checks ensure that	Verification of security

No.	Guidance	Security Activities
	software has not been tampered with or corrupted while in transit. Even if the program is not concerned with external tampering, integrity checks should still be implemented to prevent insider threats. Ensuring good software precedence depends on a few activities: • Software producers need to generate a report that guarantees that the dependencies are authentic. • The final product must provide ways to verify its integrity and authenticity, such as coming from a trusted source and having a checksum or certificate in the compression algorithm that shows it is the right version, created by the expected team. If followed, these practices help the operations team ensure that the expected dependencies are all trusted and no tampering occurred during the fabrication and distribution of the software. Additionally, after the software is obtained and deployed, it is still possible to corrupt the software. Programs should guarantee that while software is operational or in physical media used to transport data/software, only the allowed parties have access to it, and it is not modified on the way to its destination or while operating. Performance requirements should be considered and validated to ensure that the integrity checks do not create an unacceptable risk to mission success.	Implementation
5.6	All the code produced in a software project—especially one that runs on Agile methodologies—should be thoroughly tested at each development iteration. If the development team is also building project dependencies, their tests should be executed in reports and inspected at every build or release. When project dependencies come precompiled, testing becomes a little more complex. If the dependency source code is available, the development team may want to create tests for that code. Otherwise, there are utilities that can inspect the compiled code and check for rogue routines, phone home functionality, or any undesired activity that is not explicit in the dependency documentation.	Dependency monitoring
5.7	Updates to third-party software components should be centrally managed, tracked, and tested to ensure that new security issues are not introduced. At every development iteration where dependency upgrades happen, all the security testing—based on security requirements—should be	Dependency monitoring

No.	Guidance	Security Activities
	repeated and results should be thoroughly reviewed. All the guidance described in Item 5.1 also applies here, as it is important to make sure that no security gaps are being introduced with the new code. The addition of new code—especially code over which there is no source visibility—can introduce new issues that may affect unrelated parts of the application. Always make sure that your test scripts are executed and have good coverage across dependencies. The program should invest in keeping track of software dependency updates,[32] especially security-related fixes, to ensure that software is patched against known vulnerabilities and free from common security flaws.	
5.8	Ensuring environment parity is of utmost importance in a DevSecOps pipeline. OpenSAMM2 EH 1.A, "Maintain operational environment specification," demonstrated the importance of defining the operational environment and reviewing the specification for accuracy.[33] By making sure every process—from development to security testing—runs in the same environment, one can reduce variability across processes and trust that the results obtained in a development or staging setting will show the same behavior that should be expected in production. Infrastructure as Code (IaC) is a practice used in DevOps to turn the environment configuration into code so it can be automated and provide parity across all the servers used for producing software.[34] The creation and provisioning of a server's environment should always be done by IaC scripts or frameworks, in order to effectively reduce the number of issues throughout the SDLC.	Verification of process and procedures
5.9	Security tests should be executed at every new iteration or cycle in the software development. The creation of software may introduce new issues, or even make changes that include issues that have already been fixed previously. The frequency of the test execution is up to the technical, team, or IPT leads, but it is recommended that they are run at every single build of the software. This practice, as already mentioned in Item 5.4, aims to get the issues fixed as soon as they are discovered and at the minimum possible cost, before the software hits QA, staging, or production environments.	Verification of security implementation

No.	Guidance	Security Activities
5.10	Under Agile and/or DevOps methodologies, most, if not all, of the testing should be executed via scripts. This automation guarantees that • every single team that has to build the product will be able to run the tests, which represent knowledge from other parts of the whole team that may not be present at build time • if IaC is used to generate the development environment, then tests are always executed under the same conditions, whether the team is still in the Development Phase or already in production Tests scripts should be integrated into the DevOps pipeline, possibly in the configurations of the continuous integration module, which will make sure they are executed every time the product is built, be it at every commit or during a daily build. If that is not possible, scripts that build the product should make calls to those scripts as a means to ensure they are executed at every build. It is important, though, to make a distinction about the different kinds of testing that happen throughout the SDLC. There are tests generated by developers, to ensure that the different functional pieces of the code they wrote perform as expected. These tests are usually generated as unit tests, as code in the same language in which they developed the product, and are almost always automated. Further on in the SDLC, the QA team will analyze the results of the automated tests generated by developers and will run an additional battery of tests. The QA tests may include automated tests to verify if the UI is working properly, tests to evaluate the communication across different parts of the system, and very frequently, simulations of users interacting with the system run by a human (i.e., not automated). At this point, any variability introduced by a QA engineer should not diminish the quality of the testing, as the automated tests are already ensuring a certain level of quality. However, any findings resulting from manual testing should contain a thorough explanation of what was done to produce a reported issue.	Security testing
5.11	Establishing a DevSecOps pipeline for embedded software development introduces challenges not seen in non-embedded (general-purpose) systems.[35] While the software is usually developed in personal computers, it can only be partially tested in this environment and, at some point, needs to be transferred to the hardware so it is fully tested in a more realistic environment.	Security test planning

No.	Guidance	Security Activities
	This is when the challenges appear. In a DevOps pipeline, there is an assumption that systems can be tested, with the test results being available and easy to collect, because both the software being produced and the test suite all live in the same environment. These assumptions fall through as soon as the software moves into a pipeline that involves hardware, and two of the main questions that arise are (1) how do you test your hardware and (2) how do you collect the test results to make any decisions about moving forward? Pushing the code into the hardware does not seem to be a problem, but there are three different ways of testing it: 1. performing a manual hardware test, and collecting and entering results manually in a tracking system 2. using a hardware "harness" that enables a computer system to communicate, test, and collect results from the embedded system 3. performing testing via simulation Option 1 offers the least amount of complications but introduces human tasks, which add variability and also the possibility of human error. Option 2 requires a harness as the interface between the main computer and the embedded system hardware, but as the tested hardware evolves, it may also be necessary to produce newer versions of the harness to support the new capabilities of the main product. Finally, Option 3 provides possibly the most flexibility, allowing the team to simulate the hardware in a simulation platform and test it as much as possible virtually before it goes into a more advanced stage of prototyping. The third option allows the team to fix all the simpler and easier issues at a lower cost, leaving a more polished product with a much smaller number of issues for a real hardware testing phase, and thus minimizing the expenditure at this phase. The decision on what approach to use must be defined up front, and a cost comparison must be performed to clearly show benefits and possibilities to the team. Each case for embedded systems is different, but some questions may guide the team through this process: • Can you use hardware simulation to test your embedded software? If not, why? • Do you need to test your embedded software on real hardware? What particularities create that need? • How do you integrate your automated test results on embedded software back into the development pipeline?	

No.	Guidance	Security Activities
	• If monitoring for reverse-engineering and tampering, what procedures are followed when an anomaly is detected?	
5.12	Before deploying software officially to a production environment, it is usually a good idea to place it in a staging environment, or even in the real production environment, and allow it to run while that environment is heavily monitored. Everything that may detect security gaps should be enabled, from debug messages in the code, generated by the instrumentation mentioned in Item 4.2, to security tools looking at which network resources are accessible. A control group should then perform regular tasks in the newly deployed system, in order to generate messages in log files. These messages will later be examined by a team of developers and testing and security experts to make sure that no security gaps are left open. As soon as the test is deemed successful and the system is considered secure, the team may want to reset the monitoring back to an informational state. This state should still be able to capture any abnormal states or operations when those happen.	Application security monitoring
5.13	Any security issues found during the Test Phase should be documented—as well as possible—in a defect tracking system. This system should provide enough transparency so every project participant (user, tester, stakeholder, etc.) is able to add more details to any recorded issue. Every part of the system being developed supports one requirement captured at the beginning of the development. Defects found in each part of the system should be linked to the requirements related to that part, and the correction should be considered a fundamental part of the completion of its development. This is why any defect tracking system must work in parallel with the requirements system and the system that controls iterations along the development, so that any defect can be properly correlated to requirements and sprints.	Security test planning
5.14	Whenever the development team creates a system release candidate, the application is placed in a staging environment—with parity to the production environment—for QA testing. After receiving approval from the QA team, the application should be placed in the production environment. At this point, a select group of	Secure environment practices

No.	Guidance	Security Activities
	end users must be allowed to use the deployed software temporarily in production, to see if any production-related behaviors or defects are caught before the release to regular users. Users should be encouraged to use the system as they would in an ordinary situation. They can eventually be guided to make anomalous use of the system, to allow more variability, to see if it breaks with any additional cases. If any issues are found, they should be entered in a defect tracking system, as indicated in Item 5.11.	
5.15	Independent software testing or auditing is an important aspect of software development, especially in the corporate or government environment. The independence of the teams performing the audit should represent their impartiality toward the results. In this way, they can ensure that the testing will be executed without special protections to any of the parties involved in the development and testing, and the probability of covering or not reporting issues will be greatly reduced. The challenge that comes with an external, independent team is that it may not have the same access level to the software development infrastructure as the rest of the team does. It becomes necessary to enable these privileges, so the auditing team can access the code base and report its findings through the issue tracking system. At the end of the auditing/testing process, a meeting should be called so the teams can share findings and propose fixes to them. This live interaction tends to be more productive than just a simple report, but in the impossibility of a meeting or a teleconference, a good report could also work.	Verification of security implementation

6. Delivery

No.	Guidance	Security Activities
6.1	Software requirements indicate the needs of stakeholders; requirements will be implemented as features of the final product and therefore should have strong agreement from all stakeholders before the software is produced. In the initial requirements meetings, one important activity is to record how much stakeholders agree on requirements. If there are any disagreements, feasible solutions that cover most of those requirements	Final security review

No.	Guidance	Security Activities
	should also be captured to provide more context to developers as they create functionality that covers such items.	
6.2	At delivery, the customer should be made aware of who to contact in case of security concerns. There should always be two layers of contacts: an internal contact and a contact at the developer group. An internal contact belongs to the customer organization and can take quick action at the organization infrastructure level because they are knowledgeable at that level. The second contact belongs to the group that developed the product, and they are able to provide more precise action when the issue originated in the system design or development. In an ideal scenario, these two layers should be connected in a way that allows collaboration in the effort to find who owns the issue. If the issue is distributed across organizations, these contacts should be able to work together to resolve problems. In any case, and as described in Item 5.11, security issues found after deployment to production should be entered in an issue tracking system and updated accordingly at every change in state or contact with any groups who try to solve the issue.	Transition incident response planning
6.3	Once a software release is delivered, it may still contain latent issues that will appear after the development has ended. At this point, an SLA may be the instrument through which this issue will be resolved. It should stipulate how long any investigation should last depending on the severity of the problem, and how long—on average—it should take for an estimate to be delivered on addressing the issue. This agreement should have information on how to initiate this process and also who the point of contact is. If everything else fails, the FAR 52.233-1 regulates contract disputes and how they could be resolved.[36]	Transition incident response planning
6.4	Rowe and Sikes' paper "Lessons Learned: Taking It to the Next Level" best portrays the importance of recording lessons learned throughout the project as well as using those captured from similar projects: *Capturing lessons learned should be an on-going effort throughout the life of the project. This mindset should be strongly encouraged by the project manager from day one. Whether we are using lessons learned to prepare for current projects or for*	Certify, release, and archive

No.	Guidance	Security Activities
	identifying project management process improvements, we learn from project failures as well as project successes. By not learning from project failures we are doomed to repeat similar situations. *Lessons learned are the documented information that reflects both the positive and negative experiences of a project. They represent the organization's commitment to project management excellence and the project manager's opportunity to learn from the actual experiences of others. However, we are all at different levels of lessons learned utilization. Some of us do not routinely capture lessons learned because there are no defined lessons learned process in place. Or we capture lessons learned at the end of a project and never do anything with them.* *The person who will be facilitating the lessons learned session should prepare in advance. In preparation for the lessons learned session the facilitator should have the participants complete a project survey. The project survey will help the participants to be better prepared to respond during the lessons learned session and will also give them the opportunity to provide input if they are unable to attend.* *The project survey should be organized by category. The use of categories will ensure key information is not missed and will later help to focus the discussion. Standard categories for each project should be defined and additional categories specific to a project can be added. Suggested categories include project management, resources, technical, communication, business processes, requirements, design and build, testing, implementation and external areas.* *A lessons-learned session focuses on identifying project success and project failures, and includes recommendations to improve future performance on projects. To obtain optimum results, the lessons learned sessions should be facilitated by someone other than the project manager. If the project manager chooses to facilitate the session, the project survey results should be summarized by someone other than the project manager and shared with the participants during the session. The facilitator should always ask the three key questions.* • *What went right?* • *What went wrong?* • *What needs to be improved?*	

No.	Guidance	Security Activities
	Step two of the lessons learned process is to document and share findings. After lessons learned are captured, they should be reported to project stakeholders. Step three of the lessons learned process is to analyze and organize the lessons learned for application of results. Step four of the lessons learned process is to store in a repository. Finally, step five of the lessons learned process is to retrieve for use on current projects. *The final important step to ensure a successful lessons learned program is a commitment from senior level management. That commitment is made visible through regular repository metrics reviews, actions taken to implement best practices, and support to improve negative or re-occurring project trends.*[37]	
6.5	One very important aspect of the delivery is training. Anybody who might interact with the system should receive proper training and understand his or her role in using it. Training should be prepared by a team who can translate the features of the system into a language that laypeople can understand, without hiding any details that exist only in the minds of the people who designed and developed the system. This is a hard but feasible task that is underestimated most of the time by software development teams. Ideally, the training materials should be developed with the requirements in mind. Materials should also contain details obtained from developers and testing in the final phases, in order to add information about how visual components and parts of the user interface work. These training materials can be used in the creation of operation manuals, which is described in Item 6.6. Training delivery should occur *before* users have a chance to actually use the system, and it should always start from the most basic interactions and then move into more sophisticated features that may not be used every day.	Transition incident response planning
6.6	In parallel to developing the training, as described in Item 6.5, operation and instruction manuals should be produced and faithfully describe the system functionality for future reference. All the material delivered as manuals must be strongly synchronized with the delivered training. Any discrepancies might become sources of problems in the system operation and promote bad habits that result in incorrect practices or bad data. Before delivery, manuals should be	Transition incident response planning

No.	Guidance	Security Activities
	reviewed by all teams involved in the system development, and any issues should be resolved. At every iteration, this process should be repeated for better accuracy of the materials delivered. The team must consider that manuals should cover most of the questions that operators may have, to reduce calls for support. The remaining questions should be covered by an SLA or any other form of support agreement established to help users overcome difficulties in using the system.	
6.7	As indicated in Item 4.1, it is essential to establish desired metrics about the operation of the system before and during its development. During those phases, the knowledge about how to obtain them is fresh in the developers minds, and adding instrumentation is a simple and cheap task. Metrics will be a fundamental part of the continuous monitoring, which starts with the deployment of the system to production. The data generated can be either placed in log files or be part of a larger monitoring solution, where the output is sent to a database that provides data to visualizations in a dashboard. It all depends on how much the organization wants to develop in that area, but at a minimum, metrics can be captured at a low cost for further analysis. Through continuous monitoring, organizations may be able to analyze the number of system users at any given time and evaluate the load they generate. They may also use the metrics to predict hardware updates to the infrastructure, and even discover signs of attacks and/or misuse of the system (e.g., for data exfiltration).	Application security monitoring
6.8	While the system is deployed and fielded, an issue tracking system should be maintained to capture incidents that arise. Assigned responsibility for maintaining such a system should be agreed upon by both the developing organization and the users/operators. If this is the responsibility of the development team, they may be able to use their original issue tracking system, as long as there is enough transparency for the operations team to access it to either report or query about existing issues. Otherwise, the operations team will be responsible for standing that system up and having a team that coordinates fixes with whoever is responsible (e.g., IT, for generic network	Application security monitoring

No.	Guidance	Security Activities
	fixes, a development team for any changes that get down to the code level). An SLA helps define the responsibilities assigned to each team and may be the right tool to clarify any incidents that appear post-delivery, along with establishing response time and providing an idea of average cost per incident.	
6.9	At the end of its life, the process to decommission a system should implement the measures prescribed by the RMF, in order to ensure that 1. no classified, sensitive, or privacy information is exposed 2. artifacts and supporting documentation are disposed of according to their sensitivity or classification, in accordance with current DoD guidance on disposal of material and documents 3. data or objects in the decommissioned system are reviewed and appropriate actions taken to minimize any impact to the enterprise The USAF CIO Office has created the "Information Systems & Platform IT (PIT) Systems Decommissioning Guide," which provides relevant guidance on how to properly decommission systems according to the DoD rules.[38]	Secure operational enablement

APPENDIX: Relevant Frameworks and Policies

Relevant Industry Frameworks

- SEI CERT Top 10 Secure Coding Practices
- SEI CERT Secure DevOps Practices
- Building Security In Maturity Model (BSIMM)
- OWASP Software Assurance Maturity Model (OpenSAMM)
- NIST Publication 800-160 (Jan 3rd 2018), Systems Security Engineering
- NIST 800-37, Applying RMF to Federal IS
- NIST 800-53, Security & Privacy Controls for IS & Organizations
- ISO/IEC/IEEE 15288-2015, Systems and software engineering – System life cycle processes

Relevant DoD Guidance and Policies

- USAF Systems Security Engineering (SSE) Acquisition Language Guidebook (24 March 2017)
- DoDI 5000.02, Operations of the Defense Acquisition System (23 January 2020)
- DoDI 5002. 02, Enclosure 14, Cybersecurity in the Defense Acquisition System (10 August 2017)
- Mil-STD 498, Software Development and Documentation (5 December 1994)
- DoDI 8510.01, Risk Management Framework (RMF) for DoD Information Technology (28 July 2017)
- DoD Defense Innovation Board, Ten Commandments of Software (Draft 0.14, 15 April 2018)
- DISA, Application Security and Development STIG (V4R6, 27 April 2018)
- Kessel Run Memo, "Implementation of Ongoing Authorization for Agile Software Development" (18 April 2018)

References

[1] McGraw, Gary; Migues, Sammy; & West, Jacob. "BSIMM 8." Creative Commons. October 2017.

[2] Chandra, Pravir. "Open Software Assurance Maturity Model." Creative Commons. April 2017.

[3] Department of Defense (DoD). Operation of the Defense Acquisition System. DoD Instruction 5000.02. DoD. January 2020.

[4] National Institute of Standards and Technology. Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. SP 800-37 Rev. 2. December 2018.

[5] Alberts, Christopher; Dorofee, Audrey; Higuera, Ron; Murphy, Richard; Walker, Julie; & Williams, Ray. Continuous Risk Management Guidebook. Software Engineering Institute. January 1996.

[6] National Defense Authorization Act for Fiscal Year 2013. Improvements in assurance of computer software procured by the Department of Defense. P.L. 112-239 § 933. January 2013.

[7] The MITRE Corporation. Common Vulnerability Enumeration (CVE). 1999-2021. https://cve.mitre.org

[8] The MITRE Corporation. Common Weakness Enumeration (CWE). 2006-2021. https://cwe.mitre.org

[9] The MITRE Corporation. Common Attack Pattern Enumeration and Classification. 2007-2021. https://capec.mitre.org

[10] Software Engineering Institute, CERT Division. "SEI CERT Secure Coding Standards." 2016. https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

[11] Executive Order 13526. Classified National Security Information. December 2009.

[12] Office of the Under Secretary of Defense. Safeguarding Covered Defense Information and Cyber Incident Reporting. Defense Federal Acquisition Regulation Supplement 252.204-7012. December 2019.

[13] National Institute of Standards and Technology. Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations. NIST SP 800-171, Rev. 2. February 2020.

[14] Rights in Data and Copyrights. 48 C.F.R Subpart 27.4.

[15] US Air Force Systems Security Engineering (SSE) Acquisition Language Guidebook, Version 1.1. March 2017.

[16] DISA. Application Security and Development Security Technical Implementation Guide (STIG), Version 4, Release 7. STIG ID: APSC-DV-003230. July 2018.

[17] Mead, Nancy; Shull, Forrest; Vemuru, Krishnamurthy; & Villadsen, Ole. "A Hybrid Threat Modeling Method." CMU/SEI-2018-TN-002. Software Engineering Institute, Carnegie Mellon University. 2018. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=516617

[18] Software Engineering Institute. "Software Architecture." September 2018. https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328

[19] Department of Defense. Department of Defense Handbook: Product Support Analysis. MIL-HDBK-502A. DoD. March 2013.

[20] Department of Defense. Cross Domain (CD) Policy. DoDI 8540.01. DoD. August 2017.

[21] Cohen, Jason et al. "Five Types of Review." *Best Kept Secrets of Peer Code Review*. Somerville: Smart Bear, 2006. 21-36.

[22] DuPaul, Neil. "Static Testing vs. Dynamic Testing." CA Technologies, Veracode. July 2017. https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing

[23] Penov, Franci & Mortensen, Peter. "What Is Code Coverage and How Do YOU Measure It?" StackOverflow. October 2008. https://stackoverflow.com/a/195027

[24] Military Specification MIL-STD-498. Software Development and Documentation. December 1994.

[25] Office of the Secretary of Defense. A New Approach for Delivering Information Technology Capabilities in the Department of Defense, Report to Congress, November 2010, Pursuant to Section 804 of the National Defense Authorization Act for Fiscal Year 2010. United States Department of Defense, 2010. http://dcmo.defense.gov/documents/OSD%2013744-10%20 -%20804%20Report%20to%20Congress%20.pdf

[26] Northern, Carlton et al. *Handbook for Implementing Agile in Department of Defense Information Technology Acquisition*. No. MTR-100489. MITRE. 2010.

[27] Yasar, Hasan. "Deploy Secure Application with DevOps: DevSecOps." Sonatype. April 2018. https://blog.sonatype.com/deploysecureapplication

[28] National Institute of Standards and Technology. Security and Privacy Controls for Federal Information Systems and Organizations. SP 800-53 Rev. 4. April 2013.

[29] Dubrova, Elena. Fault-Tolerant Design. Springer. 2013.

[30] Neystadt, John. “Automated Penetration Testing with White-Box Fuzzing”. Microsoft. February 2008.

[31] SANS Institute. “Penetration Testing: Is It Right for You?” 2002.

[32] Chandra, Pravir. “Open Software Assurance Maturity Model.” *Environment Hardening (EH) 1.B*. Creative Commons. April 2017.

[33] OWASP. *Software Assurance Maturity Model: A Guide to Building Security into Software Development*. Version 1.0. March 2009. https://opensamm.org/downloads/SAMM-1.0.pdf

[34] Morris, Kief. “Infrastructure as Code: Managing Servers in the Cloud.” O’Reilly. 2016.

[35] Vai, Mankuan M. et al. *Secure Embedded Systems*. MIT Lincoln Laboratory Lexington United States. 2016.

[36] Federal Acquisition Regulation 52.233-1: Disputes. May 2014.

[37] Rowe, S. F. & Sikes, S. “Lessons Learned: Taking It to the Next Level.” Presented at PMI® Global Congress 2006. North America, Seattle, WA. Newtown Square, PA: Project Management Institute. 2006.

[38] US Air Force SAF/CIO A6. “Information Systems and PIT Systems Decommissioning Guide.” November 2018. https://www.dau.mil/cop/bes/DAU%20Sponsored%20Documents/Information%20Systems%20and%20PIT%20Systems%20Decommissioning%20Guide.docx

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

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally fund-ed research and development center.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM21-0420