Chapter 3

CRITICAL SIGNALS

3.1. Definition.

3.1.1. A Critical Signal is an action or information transfer that intentionally contributes to the weapon system engaging or activating a critical function in the proper manner or disabling a critical function in order to contribute to the safety of the weapon system. Per AFI 91-101 Attachment 1, critical functions are, "A function that readies a nuclear weapon for use." Not all components in a nuclear weapon system are critical to the safety of that nuclear weapon system.

3.1.2. DELETED

3.1.3. Critical Signals flow from human intent, which is the action of an authorized human to engage the critical function. Critical Signals result in the critical function initiation, which is the action on the weapon system. Critical Signals deliberately contribute to disabling surety and safety features, ultimately culminating in the detonation of a nuclear weapon. Critical Signals, specifically the Safing Command, restore surety and safety features the weapon system previously disabled. This manual considers Critical Signals that restore surety and safety features to have implicit human intent. Weapon system software can adversely affect nuclear safety by bypassing human intent through the generation or propagation of unauthorized Critical Signals or by failing to meet human intent through the failure to generate or propagate Safing Commands.

3.1.4. Inadvertent, unauthorized, or malicious generation of Critical Signals could directly lead to the unauthorized detonation of a nuclear weapon. The system often passes software-implemented or software-controlled Critical Signals as data or calls between software components or as messages over digital buses.

3.1.5. Types of Critical Signals:

3.1.5.1. Authorize. Entry of the Permissive Action Link or similar mechanism that directly authorizes the use of a warhead or DOE-provided weapon. If operators employ the mechanism prior to the start of a nuclear mission, then this Critical Signal does not count toward the requirement in **paragraph 8.11.5**.

3.1.5.2. Prearm and Prearm Consent. Activation of the initial "reversible consent" at the start of a nuclear mission.

3.1.5.3. Launch. Activation of rocket motors for a ground-launched nuclear weapon.

3.1.5.4. Release. Separation of a nuclear weapon from an aircraft.

3.1.5.5. Arm. Readying a nuclear weapon so that a fuzing signal will operate the firing system.

3.1.5.6. Navigation Validation. The actions necessary to:

3.1.5.6.1. Validate the location of the target prior to enabling the fuzing mechanism of the nuclear weapon.

3.1.5.6.2. Protect the targeting data, which contains the target of the nuclear weapon, as a Critical Signal. **Note:** Aircraft mission planning software does not require Nuclear Safety Design Certification, since the operator establishes human intent by entering targeting data into the nuclear weapon system.

3.1.5.7. Safing Command. A message the weapon system or operator initiate to place the nuclear weapon into a state such that the receipt of the final irreversible action does not cause a nuclear detonation through either the prevention of Launch or Release, or by disabling the nuclear weapon's capability to detonate. This manual considers the Safing Command to be a Critical Signal and human intent for the Safing Command to be implicit. See **paragraph 3.1.3** The nuclear weapon system should transmit the Safing Command to the nuclear weapon upon the withdrawal of human intent, when the nuclear weapon system detects failures that can affect proper Critical Signal operation, or when the status of the weapon system changes such that the detonation of the nuclear weapon would violate human intent.

3.1.6. This manual designates software components that can impact Critical Signals or prevent the transmission of Safing Commands as nuclear critical software.

3.2. Impact on Critical Signals.

3.2.1. The Critical Signal path flows from human intent to critical function activation. Along this path, it can flow through many different hardware components, processors, buses, software components, programmable logic devices, etc. However, not all those components are able to manipulate in a meaningful way or impact the Critical Signal in an unauthorized way. Impact only occurs when either:

3.2.1.1. The component has the designed ability to assign semantic meaning to a Critical Signal; in other words, the component understands what the Critical Signal means.

3.2.1.2. The component can alter the Critical Signal in any way that would result in erroneous activation of critical functions or prevent return of the system to a safe state. This does not apply when the receiver of the Critical Signal can verify the integrity and validity of the Critical Signal; the receiver should be able to verify the source of the Critical Signal.

3.2.2. For example, a Critical Signal to Prearm a weapon can pass through a component that converts the signal from one digital form to another. If the component in question does not understand the meaning of the data to modify it, and a mechanism such as encryption protects the Critical Signal, then the component cannot meaningfully alter the Critical Signal. The component in question does not have a safety impact on the Critical Signal.

3.3. Software Effects on Critical Signals.

3.3.1. Many requirements in this document place limits directly on the mechanisms behind Critical Signal propagation through software to ensure the integrity of the Critical Signal. These requirements focus on preventing inadvertent or unauthorized interactions between components and ensure the integrity of the Critical Signals by ensuring the integrity of:

3.3.1.1. The software that generates, alters, or interprets Critical Signals.

3.3.1.2. The data that represents Critical Signals.

3.3.1.3. The actions that transfer Critical Signals throughout the software and the system.

3.3.2. An effective way of preventing unauthorized interactions between different software components is designing the system so different software components do not share memory space.

3.3.3. The Development Organization can design the software components to reside on different single-core processors or to use an operating system that strictly enforces spatial partitioning. **Note:** Multi-core technologies generally do not solve the shared memory space issue without additional spatial and temporal partitioning. Cores on a multi-core processor generally have access to shared volatile memory and cache.

3.4. Critical Signal Analysis.

3.4.1. System program offices are responsible for understanding the flow of Critical Signals within their system and designing the system to minimize potential updates to the components that impact Critical Signals. System program offices should consider the impact on Critical Signals and their corresponding critical functions.

3.4.2. The system program office shall perform an analysis of the Critical Signals of the weapon system. **(T-1)** Review of the Critical Signal analysis should prove that it traces all Critical Signals from human intent through critical function activation by proving the analysis contains the following:

3.4.2.1. The flow of Critical Signals from human intent to critical function activation.

3.4.2.2. All components between human intent and critical function activation

3.4.2.3. The potential effect those components could have on the Critical Signal.

3.4.2.4. The protection mechanisms provided to Critical Signals.

3.4.3. The system program office shall provide the Critical Signal analysis of the weapon system in any Nuclear Certification Impact Statement and NSE that includes components in the Critical Signal path or components that interface with other components in the Critical Signal path. **(T-1)** System program offices should reference in the Critical Signal analysis any navigation components important to Navigation Validation and any Critical Signal paths that include Safing Commands. Analysis of the Nuclear Certification Impact Statement and NSE should prove that each document contains a Critical Signal analysis that includes enough detail to show what Critical Signals, if any, software updates may affect.

Chapter 4

NUCLEAR SAFETY VERIFICATION PHILOSOPHIES AND METHODOLOGIES

4.1. Applicability. The system program office uses the requirements found in this manual to develop or sustain the weapon system, verification procedures, and verification environment.

4.2. Nuclear Safety Objectives (NSO). NSOs are the high-level verification goals with which nuclear safety verification shows compliance in order to obtain Nuclear Safety Design Certification. The Verification Organization derives NSOs directly from requirements allocated in the Certification Requirements Plan. NSOs may be identical to requirements the Certification Requirements Plan allocates to the nuclear weapon system software. The Verification Organization performs a traceability analysis to demonstrate that the NSOs represent all allocated requirements and that all NSOs derive from one or more allocated requirements. The Verification Organization provides the traceability analysis in the Nuclear Safety Software Verification Plan.

4.3. Nuclear Safety Requirements (NSR). NSRs are the lowest level verification requirements with which nuclear safety verification shows compliance in order to obtain Nuclear Safety Design Certification. The Verification Organization derives NSRs from NSOs, derives verification activities such as tools and procedures from NSRs, and performs verification against NSRs. The Verification Organization performs a traceability analysis to demonstrate that NSRs represent all NSOs and that all NSRs derive from one or more NSOs. The Verification Organization provides the traceability analysis in the Nuclear Safety Software Verification Plan.

4.4. Verification. NSOs and NSRs are the requirements that the Verification Organization uses in generating verification procedures and verification tools. NSOs and NSRs should specifically focus on Critical Signals and related requirements as described in **paragraph 3.1**, along with the other requirements in this manual. **Figure 4.1** explains the general requirements flow. On the left side is a typical development process that a Development Organization might follow; on the right side is the Nuclear Safety Design Certification process that the Verification Organization should follow. **Table 4.1** describes the verification types that AFSEC/SEWN may invoke or require as part of the Nuclear Safety Design Certification process, followed by additional discussions on each verification type. The rules that govern the type of verification described in **Table 4.1** start in **paragraph 4.5** This includes the independence requirements for each verification type and the independence required in the verification process.

Figure 4.1. Nuclear Safety Software Development and Verification Flow Chart.

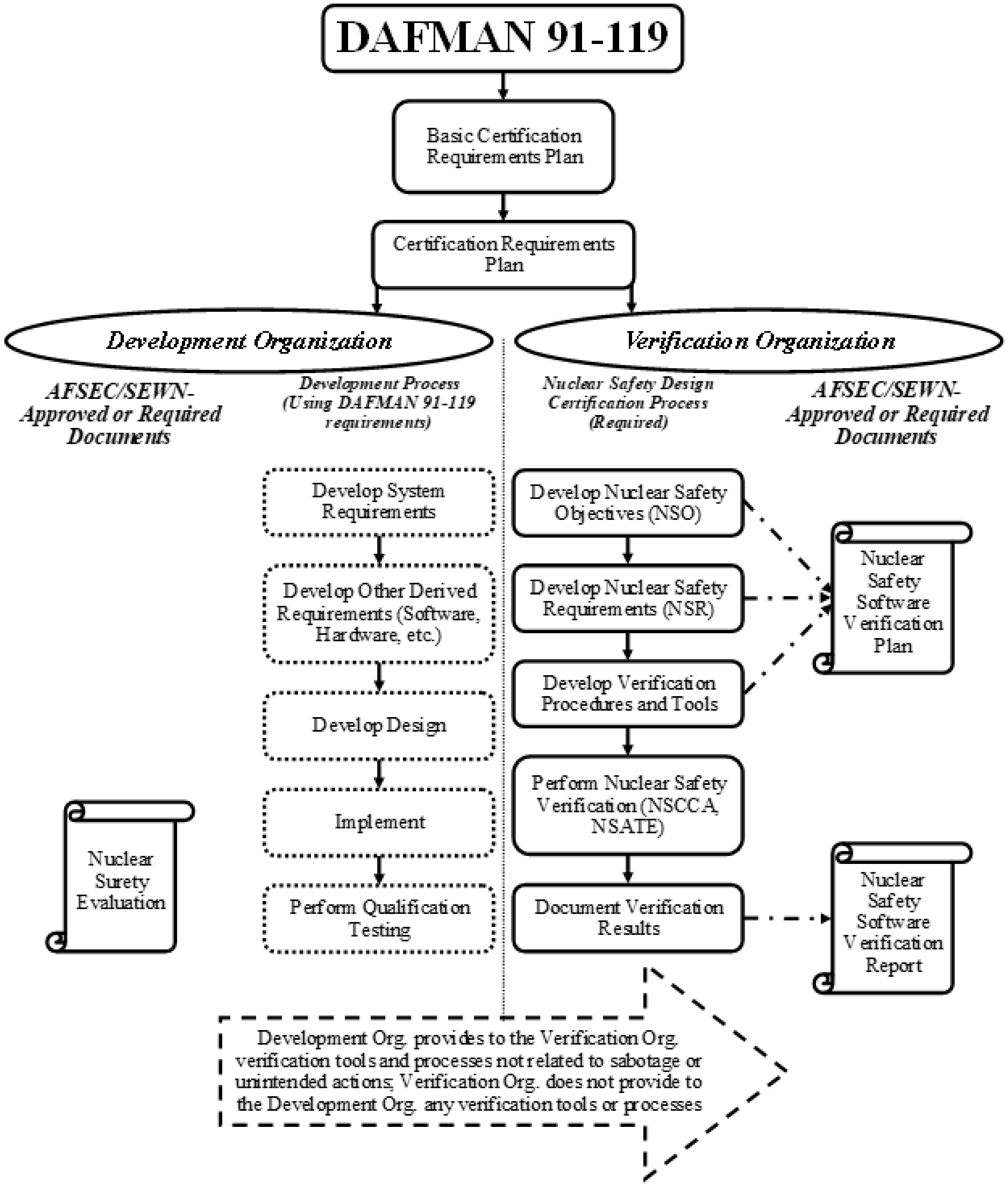
DAFMAN 91-119
Basic Certification Requirements Plan
Certification Requirements Plan
Development Organization
Verification Organization
AFSEC/SEWN-Approved or Required Documents
Development Process (Using DAFMAN 91-119 requirements)
Nuclear Safety Design Certification Process (Required)
AFSEC/SEWN-Approved or Required Documents
Develop System Requirements
Develop Other Derived Requirements (Software, Hardware, etc.)
Develop Design
Implement
Perform Qualification Testing
Nuclear Surety Evaluation
Develop Nuclear Safety Objectives (NSO)
Develop Nuclear Safety Requirements (NSR)
Develop Verification Procedures and Tools
Perform Nuclear Safety Verification (NSCCA, NSATE)
Document Verification Results
Nuclear Safety Software Verification Plan
Nuclear Safety Software Verification Report
Development Org. provides to the Verification Org. verification tools and processes not related to sabotage or unintended actions; Verification Org. does not provide to the Development Org. any verification tools or processes

Table 4.1. Verification Types.

Nuclear Safety Verification Key Area	**Nuclear Safety Cross-Check Analysis (NSCCA)**	**Nuclear Safety Analysis and Technical Evaluation (NSATE)**	**Nuclear Safety Regression Testing**
	Verification Type Purpose		
Verification Functionality	Functionality executes as this manual requires; software meets requirements in this manual as expressed by NSOs and NSRs; Functionality does not perform in any way to contribute to safety violation; No intentionally malicious software or unintended actions. See **paragraph 4.6**.	Functionality executes as this manual requires; software meets requirements in this manual as expressed by NSOs and NSRs. See **paragraph 4.5**.	Newly added features do not adversely affect the generation or propagation of Critical Signals; Updates do not adversely affect Critical Signal functions. Nuclear Safety Regression Testing is only appropriate for modifications to existing systems where Development Organizations are not making changes to nuclear code and AFSEC/SEWN has independently determined that an impact to nuclear functionality or certification is unlikely. See **paragraph 4.7**.
	Independence		
Data Sharing	Procedures in place to prevent inadvertent or intentional sharing of Verification Organization safety-related findings, NSOs, NSRs, procedures, processes, and tool suite definitions with Development Organization. See **paragraphs 4.5.7** and **4.6.11**.		Standard procedures.
Verification Organization	Independent Verification Organization (Managerially, Financially, Technically). See **paragraphs 4.5.3** and **4.6.5**.		Common Verification Organization and Development Organization.
Tool Chain	Independent Acquisition of Software Tool Chain (This should be the same tool chain both Development Organization and Verification Organization use). See **paragraphs 4.5.4** and **4.6.6**.		Shared Verification and Development Tool Chain.
Verification Tools	Independent Development and Acquisition of Verification Tools. See **paragraphs 4.5.6** and **4.6.7**.		Shared Verification and Development Verification Tools.

Nuclear Safety Verification Key Area	**Nuclear Safety Cross-Check Analysis (NSCCA)**	**Nuclear Safety Analysis and Technical Evaluation (NSATE)**	**Nuclear Safety Regression Testing**
Development of NSOs, NSRs, and Verification Procedures	Verification Organization independently develops NSOs, NSRs, and verification procedures that focus on sabotage or unintended actions. See **paragraph 4.6.7**.	Verification Organization may use any requirements or procedures provided by the Development Organization. See **paragraph 4.5.1**.	Shared Verification and Development and verification procedures (Qualification Tests that trace to safety design certification requirements); Expectation is Nuclear Safety Regression Testing should improve and expand through multiple releases. See **paragraph 4.7.3**.
Verification Facilities	With justification and government-approved oversight, Development Organization and Verification Organization may share Verification Facilities, but not concurrently. See **paragraph 4.6.8**.	Development Organization and Verification Organization may share Verification Facilities, but not concurrently. See **paragraph 4.5.6**.	Shared Verification and Development Verification Facilities.
	Deliverables		
Verification Plans and Verification Results	Verification Plan with an overview of verification types and verification approaches, and NSOs and NSRs; AFSEC/SEWN approves Verification Plan prior to execution, but the Verification Organization may provide portions as the Verification Organization completes the portions; Verification Report provided to AFSEC/SEWN, Verification Results summary included in Nuclear Surety Evaluation. See **paragraphs 4.4.1** and **4.4.2**.		Results of Nuclear Safety Regression Testing provided to AFSEC/SEWN.
	Development Management		
Security Concerns	Verification Organization implements special security and control measures to prevent sabotage of NSCCA, as the Verification Plan defines. See **paragraph 4.6.10**.	Verification Organization implements industry-standard security and cybersecurity implementations, as the Verification Plan defines.	Industry-standard security and cybersecurity implementations.

Nuclear Safety Verification Key Area	**Nuclear Safety Cross-Check Analysis (NSCCA)**	**Nuclear Safety Analysis and Technical Evaluation (NSATE)**	**Nuclear Safety Regression Testing**
Configuration Management	Tool Chain and Verification Tools secured; Verification Organization implements non-repudiable (non-spoofable) Configuration Management; Verification Plan includes Configuration Management approach; Verification Organization performs audits against Configuration Management to show that the Verification Organization tracks verified software configurations in the configuration management system and only includes reviewed tools in verification. See **paragraphs 4.6.9** and **4.6.10**.	Industry-standard configuration management for Development and Verification Organizations.	

4.4.1. When AFSEC/SEWN requires NSCCA or NSATE, the system program office shall provide the Nuclear Safety Software Verification Plan to AFSEC/SEWN for approval. **(T-1)**

4.4.1.1. Analysis of the Nuclear Safety Software Verification Plan should prove that it contains the following for NSCCA and NSATE, in a single document or multiple documents that the Verification Organization develops specifically for Nuclear Safety Software Verification:

4.4.1.1.1. Description of the Verification Organization's independence.

4.4.1.1.2. Procedures the system program office designed to maintain the Verification Organization independence.

4.4.1.1.3. Compensatory measures to ensure independence when the Development Organization and the Verification Organization share limited resources such as hardware integration laboratories.

4.4.1.1.4. The list of NSOs and NSRs, and related traceability analyses between the NSOs, NSRs, and verification procedures.

4.4.1.1.5. Summary descriptions and identifiers or names for all verification procedures.

4.4.1.1.6. The list of verification tools both purchased and developed, and a summary of the use of each tool.

4.4.1.2. Analysis of the Nuclear Safety Software Verification Plan should prove that it contains the following for NSCCA, in a single document or multiple documents that the Verification Organization develops specifically for Nuclear Safety Software Verification:

4.4.1.2.1. Procedures for the use of a secure, non-repudiable configuration management system.

4.4.1.2.2. Procedures for the use of security and control measures to prevent sabotage.

4.4.2. When AFSEC/SEWN requires NSCCA or NSATE, the Verification Organization shall provide the Nuclear Safety Software Verification Report directly to AFSEC/SEWN after the Verification Organization completes verification. **(T-1)** The Verification Organization should provide the report several times over the process of verification to streamline certification, but the Verification Organization only needs to provide the report once for each certification effort.

4.4.2.1. Analysis of the Nuclear Safety Software Verification Report should prove that it contains the following for NSCCA and NSATE, in a single document or multiple documents that the Verification Organization develops specifically for Nuclear Safety Software Verification:

4.4.2.1.1. Results and explanations of verification activities, providing a verification procedure identifier, verifier name, and result.

4.4.2.1.2. DELETED

4.4.2.1.3. Explanations of how the Verification Organization assembled the development and verification suite independently of the Development Organization.

4.4.2.1.4. Explanations of how the Verification Organization avoided providing any verification-related data to the Development Organization.

4.4.2.1.5. Explanations of how the Verification Organization used data representing real-world scenarios.

4.4.2.1.6. Explanations of how the Verification Organization executed verification procedures on production-equivalent platform hardware or demonstrably equivalent emulations and simulations.

4.4.2.2. Analysis of the Nuclear Safety Software Verification Report should prove that it contains the results and explanations of accomplishing the NSCCA, in a single document or multiple documents that the Verification Organization develops specifically for Nuclear Safety Software Verification.

4.4.3. The system program office shall submit to the Verification Organization all data required to verify that the software configuration is fully compliant with nuclear surety requirements. **(T-1)** Verification results should prove the provided data is sufficient for the Verification Organization conclusively to determine compliance with nuclear surety requirements. Per **paragraph 4.6.7**, the system program office can provide the Development Organization verification requirements and procedures that focus on identifying sabotage or unintended actions that could violate nuclear safety only after the Verification Organization has developed their own verification requirements and procedures that focus on identifying sabotage or unintended actions that could violate nuclear safety. The Verification Organization should evaluate the software using the following rules:

4.4.3.1. Evaluate the software at the source code level. All source code, including auto-generated code, should be intelligible and easily audited by the Verification Organization. The requirements in this document apply to all source code, even auto-generated code. Source code should include error handling within the source code as a means to safeguard against software faults, in compliance with **paragraph 8.10.5**.

4.4.3.2. Evaluate the software using software documentation, including models and system architectures as applicable, but not as the sole basis of evaluation.

4.4.3.3. Evaluate the version of software that the system program office expects to be certified.

4.4.3.4. If the Verification Organization is verifying new software for initial certification, the Verification Organization may request access to interfacing software; that is, software that interfaces with the software the Verification Organization is verifying. The Verification Organization should only request access to the interfacing software if the Verification Organization recognizes a potential safety impact. Notwithstanding this paragraph, the Verification Organization does not need to analyze any software except the software that the Verification Organization is verifying.

4.4.4. When the system program office has submitted modified, previously-certified software for certification upon which the Verification Organization must perform an NSCCA, the Verification Organization shall analyze all system components which interface with the modified software. **(T-1)** Verification results should prove the Verification Organization did not just analyze the modifications to the modified software, but looked at potential impacts to Critical Signals of the modifications across all system components that interface with the modified software.

4.5. NSATE. The NSATE is a software verification process that can include any of the common forms of verification: test, demonstration, inspection, and analysis. The NSATE extends throughout the weapon development and is performed by an organization that is technically, managerially, and financially independent of the Development Organization. Verification activities can include functional verification as well as tool-based analysis and dynamic analysis.

4.5.1. The NSATE process verifies that the software complies with established nuclear safety design criteria. The process analyzes software requirements, design, and code to detect software deficiencies before they can propagate into later development phases. The process also analyzes the final software design to determine its compliance with NSRs. NSATE is different from qualification testing because of its emphasis on proving that the software meets NSRs rather than requirements that are unrelated to Nuclear Safety. NSATE teams may use verification procedures or systems that the Development Organization developed but cannot provide the updated verification procedures or processes back to the Development Organization. The Verification Organization should complete NSATE by conducting a bit-for-bit comparison between the software the Development Organization delivered and the software the NSATE organization verified. The Verification Organization should regenerate or recompile the software, disassemble the object file as necessary, and compare the contents between the regenerated object file and the provided object file.

4.5.2. When AFSEC/SEWN requires NSATE, the NSATE team shall analyze the NSOs and NSRs for a weapon system to verify that the system executes the functionality as intended. **(T-**

1) Analysis of the Nuclear Safety Software Verification Report should prove it contains results and explanations of functionality verification based on the NSOs.

4.5.3. When AFSEC/SEWN requires NSATE, the system program office shall document the independence of the NSATE organization, managerially, financially, and technically in the Nuclear Safety Software Verification Plan. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove that it includes a description of the NSATE organization's independence. The system program office chooses the NSATE organization with AFSEC/SEWN approval.

4.5.3.1. The system program office shall ensure that the Development Organization and the NSATE organization are not in the same corporate entity or governmental organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove that the Development Organization and the NSATE organization do not reside in the same corporate entity, and that managerial levels in the governmental organization are not common until the commander of the MAJCOM or director of the agency.

4.5.3.2. The system program office shall ensure that the Development Organization does not engage former members of the NSATE organization for the Development Organization until three years or the length of the most recent software release cycle from the last time the NSATE organization engaged the former member have passed, whichever is longer. **(T-1)** A software release cycle is defined as the time from the previous official operational release of the software to the most recent official operational release of the software. Analysis of the system program office contracting documents, memorandums of understanding, other similar contractual documents, and employment records of the Development Organization should prove that the Development Organization has not engaged former members of the NSATE organization for the required amount of time.

4.5.3.3. If the Development Organization includes a contractor, the system program office shall write contracts to require the contractor not employ former members of the NSATE organization for the Development Organization in accordance with **paragraph 4.5.3.2**. **(T-1)** Analysis of the contracting language should prove that the system program office has included language that prevents the contractor from employing members of the NSATE organization in the same software development program for the required amount of time. **Note**: This does not prevent former members of the NSATE organization from working for the contractor; this requirement only excludes such individuals from the Development Organization employing them in the same program as the NSATE organization for the required amount of time specified in **paragraph 4.5.3.2**.

4.5.3.4. The system program office shall ensure that all personnel assigned to the NSATE organization sign a nondisclosure agreement preventing release of information concerning verification requirements, procedures, processes, and methodologies. **(T-1)** Analysis of employment records of the NSATE organization should prove that all personnel assigned to the NSATE organization have signed the nondisclosure agreement.

4.5.4. When AFSEC/SEWN requires NSATE, the system program office shall develop and implement procedures designed to maintain the independence of the NSATE organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove that it includes procedures designed to maintain the independence of the NSATE organization. The system program office should maintain compliance by providing AFSEC/SEWN an NSATE activity review at least annually, or more often as needed.

4.5.5. DELETED

4.5.6. When AFSEC/SEWN requires NSATE, the NSATE team shall independently identify, acquire, and develop as necessary the development and verification suite components. **(T-1)** The Development Organization should only provide the tools and data to the Verification Organization that are necessary to duplicate the Development Organization's development and verification environment. The Verification Organization should assemble all other components of the development and verification environment from the original provider such as a tool company or repository. Analysis of the Nuclear Safety Software Verification Report should prove the report contains the following:

4.5.6.1. Explanations of how the Verification Organization assembled a development and verification suite independent of the Development Organization.

4.5.6.2. Explanations of how the organizations used compensatory measures to ensure independence when the Development Organization and the Verification Organization shared limited resources such as hardware integration laboratories.

4.5.7. When AFSEC/SEWN requires NSATE, the NSATE team shall not share methods, verification plans, configurations, or other verification-related data with the Development Organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove the NSATE team avoided providing any verification-related data to the Development Organization.

4.5.8. When AFSEC/SEWN requires NSATE, the NSATE team shall use data that accurately represents operational use, including environments as specified in the weapon system Stockpile-to-Target Sequence for nominal verification activities. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove the use of data representing real-world scenarios.

4.6. NSCCA. The NSCCA is a software verification process that can include any of the common forms of verification: test, demonstration, inspection, and analysis. The NSCCA extends throughout the weapon development and is performed by an organization that is technically, managerially, and financially independent of the Development Organization. Verification activities can include functional verification as well as tool-based analysis and dynamic analysis.

4.6.1. In addition to the requirements of an NSATE (refer to **paragraph 4.5**), an NSCCA ensures that the software cannot perform in any way that could contribute to a nuclear safety violation. Verification activities focus on ensuring that the software correctly performs nuclear critical functions, and that the software does not perform any unintended actions that could violate nuclear safety. The NSCCA is also unique in its concern for malicious intent. While the other forms of software evaluation assume that any software deficiencies were unintentional, the NSCCA looks for intentionally caused issues and employs special security and control measures to prevent sabotage of the NSCCA effort itself.

4.6.2. Like the NSATE, an NSCCA requires the development of NSOs and NSRs. The Verification Organization should complete NSCCA by conducting a bit-for-bit comparison between the software the Development Organization delivered and the software the NSCCA organization verified. The Verification Organization should regenerate or recompile the software, disassemble the object file as necessary, and compare the contents between the regenerated object file and the provided object file.

4.6.3. When AFSEC/SEWN requires NSCCA, the NSCCA team shall analyze the NSOs and NSRs for a weapon system to verify that the system executes the functionality as intended. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove it contains results and explanations of verification based on the NSOs.

4.6.4. When AFSEC/SEWN requires NSCCA, the NSCCA team shall analyze the system to verify that the system only performs authorized functions when it receives authorization to perform those functions, and nothing else. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove it contains results and explanations of NSCCA-specific verification activities.

4.6.5. When AFSEC/SEWN requires NSCCA, the system program office shall document the independence of the NSCCA organization, managerially, financially, and technically, in the Nuclear Safety Software Verification Plan. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove it includes a description of the NSCCA organization's independence. The system program office chooses the NSCCA organization with AFSEC/SEWN approval.

4.6.5.1. The system program office shall ensure that the Development Organization and the NSCCA organization are not in the same corporate entity or governmental organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove that the Development Organization and the NSCCA organization do not reside in the same corporate entity, and that managerial levels in the governmental organization are not common until the commander of the MAJCOM or director of the agency.

4.6.5.2. The system program office shall ensure that the Development Organization does not engage former members of the NSCCA organization for the Development Organization until three years or the length of the most recent software release cycle from the last time the NSCCA organization engaged the former member has passed, whichever is longer. **(T-1)** A software release cycle is defined as the time from the previous official operational release of the software to the most recent official operational release of the software. Analysis of the system program office contracting documents, memorandums of understanding, other similar contractual documents, and employment records of the Development Organization should prove that the Development Organization has not engaged former members of the NSCCA organization for the required amount of time.

4.6.5.3. If the Development Organization includes a contractor, the system program office shall write contracts to require the contractor not to employ former members of the NSCCA organization for the Development Organization in accordance with **paragraph 4.6.5.2**. **(T-1)** Analysis of the contracting language should prove that the system program office has included language that prevents the contractor from employing members of the NSCCA organization in the same software development program for the required amount of time. **Note:** This does not prevent former members of the NSCCA organization from working for the contractor; this requirement only excludes such individuals from the Development Organization employing them in the same program as the NSCCA organization for the required amount of time specified in **paragraph 4.6.5.2**.

4.6.5.4. The system program office shall ensure that all personnel assigned to the NSCCA organization sign a nondisclosure agreement preventing release of information concerning verification requirements, procedures, processes, and methodologies. **(T-1)** Analysis of employment records of the NSCCA organization should prove that all personnel assigned to the NSCCA organization have signed the nondisclosure agreement.

4.6.6. When AFSEC/SEWN requires NSCCA, the system program office shall develop and implement procedures designed to maintain the independence of the NSCCA organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove it includes procedures designed to maintain the NSCCA organization independence. The system program office should maintain compliance by providing AFSEC/SEWN an NSCCA activity review at least annually, or more often as needed.

4.6.7. When AFSEC/SEWN requires NSCCA, the NSCCA team shall independently develop their own verification requirements and procedures that actively seek out sabotage or unintended actions that could violate nuclear safety. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove independent development of verification requirements and procedures that actively seek out sabotage or unintended actions that could violate nuclear safety.

4.6.8. When AFSEC/SEWN requires NSCCA, the NSCCA team shall independently identify, acquire, and develop as necessary the development and verification suite components. **(T-1)** The Development Organization should only provide the tools and data to the Verification Organization that are necessary to duplicate the Development Organization's development and verification environment. The Verification Organization should assemble all other components of the development and verification environment from the original provider such as a tool company or repository. Analysis of the Nuclear Safety Software Verification Report should prove the report contains the following:

4.6.8.1. Explanations of how the Verification Organization assembled a development and verification suite independent of the Development Organization.

4.6.8.2. Explanations of how the organizations used compensatory measures to ensure independence when the Development Organization and the Verification Organization shared limited resources such as hardware integration laboratories.

4.6.9. When AFSEC/SEWN requires NSCCA, the NSCCA team shall use secure, non-repudiable configuration management systems to maintain software source code, software products, verification tools, and verification procedures. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove it includes procedures for the use of a secure, non-repudiable configuration management system.

4.6.10. When AFSEC/SEWN requires NSCCA, the NSCCA team shall use security and control measures to prevent sabotage of the NSCCA process. **(T-1)** Analysis of the Nuclear Safety Software Verification Plan should prove it includes procedures for the use of security and control measures to prevent sabotage; the system program office should provide compliance information during the NSCCA activity review held at least annually.

4.6.11. When AFSEC/SEWN requires NSCCA, the NSCCA team shall not share methods, verification plans, configurations, or other verification-related data with the Development Organization. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove the NSCCA team avoided providing any verification-related data to the Development Organization.

4.6.12. When AFSEC/SEWN requires NSCCA, the NSCCA team shall use data that accurately represents operational use, including environments as specified in the weapon system Stockpile-to-Target Sequence for nominal verification activities. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove the use of data representing real-world scenarios.

4.7. Nuclear Safety Regression Testing.

4.7.1. Nuclear Safety Regression Testing is the selective retesting of a modified, certified software system. Nuclear Safety Regression Testing ensures that the changes to the software have not allowed the unauthorized or inadvertent operation of Critical Signals. Nuclear Safety Regression Testing also ensures that the changes to the software have not caused the unauthorized or inadvertent failure of Safing Commands. Nuclear Safety Regression Testing specifically tests the interactions between the updated software and software that potentially has a nuclear safety impact, specifically, software that would require an NSATE or an NSCCA. Traceability to sections of changed software should determine the extent of Nuclear Safety Regression Testing. Nuclear Safety Regression Testing occurs after the Development Organization makes a modification to software that AFSEC/SEWN has determined does not require an NSATE or an NSCCA.

4.7.2. Since many changes to software can have extended effects, Nuclear Safety Regression Testing should use all qualification tests that verify that Critical Signals only operate when authorized, verify that Safing Commands always operate correctly, and verify the interfaces between the updated software and the software that would require an NSATE or an NSCCA. These qualification tests should focus on the interface level. System-level tests may be too high-level to detect errors appropriately in the lower-level interfaces.

4.7.3. When AFSEC/SEWN requires Nuclear Safety Regression Testing, the Development Organization or an organization the system program office designates shall execute all qualification tests that the Development Organization has developed that verify that Critical Signals only operate when authorized, verify that Safing Commands always operate correctly, and verify the interfaces between the updated software and the software that would require an NSATE or an NSCCA. **(T-1)** Analysis of the NSE should prove that it contains the signature of a representative of the system program office, the results of Nuclear Safety Regression Testing, descriptions of the tests to show the tests properly verify the updated code, and evidence that the Development Organization performed verification activities on production-equivalent platform hardware.

Chapter 5

VERIFICATION GUIDELINES

5.1. Verification Type Guidelines. AFSEC/SEWN uses the following guidelines to determine when a system program office should apply a verification type (NSATE, NSCCA, and Nuclear Safety Regression Testing).

5.1.1. The Verification Organization should perform NSATE on any component that meets any of the following scenarios:

5.1.1.1. The component is responsible for the propagation of targeting data within the delivery platform.

5.1.1.2. The component provides navigation information. If either of the following scenarios apply, then the Verification Organization does not need to perform NSATE on the component, but the system program office should document the reasoning in the NSE:

5.1.1.2.1. The navigation information acts as a negating input that directly prevents Critical Signal propagation, particularly the Launch or Release of a nuclear weapon, but the navigation information is not used in Navigation Validation.

5.1.1.2.2. The weapon system combines at least two different, independent navigation sources that use different navigation phenomena. For example, a weapon system could combine the data from an inertial navigation system and a radar system to provide a single navigation solution. **Note**: Although this paragraph clarifies that independent navigation sources may not need an NSATE, software that combines the navigation data from the independent navigation sources into a single navigation solution would most likely require an NSCCA, as the combining software would affect Navigation Validation. See paragraphs **5.1.2.1** and **5.1.2.3**.

5.1.1.3. The component verifies the operation of components in the weapon system during operational use or prior to operational use through test equipment or other equipment separate from the operational weapon system or through built-in-test operations within the operational weapon system; in particular, this applies if the software meets any of the following scenarios. **Note**: The system program office should document in the NSE the reasoning to omit a NSATE on any test equipment software, using the following scenarios.

5.1.1.3.1. The software prevents weapons from straying into abnormal environments as the Stockpile-to-Target Sequence defines.

5.1.1.3.2. The software detects or prevents damage to the installed nuclear weapon system components that initiate Critical Signals; or propagate Critical Signals if the weapon system operates under the Energy Control Concept, as defined in AFMAN 91-118.

5.1.1.3.3. The software prevents the generation or propagation of unauthorized Critical Signals during the process of testing the weapon system components.

5.1.1.3.4. The software is responsible to place the unit under test in a safe state such that failure to place the unit under test in a safe state could lead to the generation of unauthorized Critical Signals. This only applies to weapon systems that operate under the Energy Control Concept, as defined in AFMAN 91-118, unless the weapon system is logically connected to the warhead.

5.1.1.3.5. The software detects damage to the nuclear weapon system components that are responsible for making the weapon or weapon system safe or for propagating the Safing Command and the software can provide a false positive passing result on one or more of those components; this could contribute to the inability of the weapon system to make the weapon safe.

5.1.1.3.6. The software acts in lieu of a weapon or a weapon system component and the software replicates or simulates the actions of the weapon or weapon system component.

5.1.1.3.7. The software verifies the functionality of a weapon system critical component prior to its installation into the operational system.

5.1.1.3.8. The software falls under one of the above scenarios, and the system program office has made updates to the software other than updates to constants that only affect the limits of one or more tests.

5.1.1.4. The component, with human interaction, contributes to the weapon system security or protects the weapon system from abnormal environments as the Stockpile-to-Target Sequence defines. The system program office does not need to perform NSATE against a security system certified or approved by the NSA or other government agency as approved by AFSEC/SEWN.

5.1.1.5. The component propagates Critical Signals or Safing Commands, and the Development Organization developed the component software specifically for the weapon system. **Note:** The Development Organization should acquire off-the-shelf weapon system components through mechanisms that protect the supply chain of the weapon system component.

5.1.1.6. The system program office has re-hosted the component on a different processor or has rebuilt the component using a changed tool chain if the component would normally require NSATE verification. The NSATE team should focus on the effects of the re-host process or tool chain changes.

5.1.1.7. The software loads Critical Software or Critical Data, and the system design does not use mechanisms described in **paragraph 6.2** to protect the Critical Software or Critical Data.

5.1.2. The Verification Organization should perform NSCCA on any component that meets any of the following definitions:

5.1.2.1. The software component is nuclear critical because it impacts or initiates a Critical Signal, as defined in **paragraph 3.1**.

5.1.2.2. The component executes in the same memory partition or unpartitioned memory space or the same Programmable Logic Device as any nuclear critical software component that impacts a Critical Signal, as defined in **paragraph 3.1**.

5.1.2.3. The component validates a Critical Signal, as defined in **paragraph 3.1**.

5.1.2.4. The component is solely responsible for weapon system security or the protection of the weapon system from abnormal environments as the Stockpile-to-Target Sequence defines, without any human interaction or intervention. The system program office does not need to perform NSCCA against a security system certified or approved by the NSA or other government agency as approved by AFSEC/SEWN.

5.1.2.5. The system program office has re-hosted the component on a different processor or has rebuilt the component using a changed tool chain if the component would normally require NSCCA verification. The NSCCA team should focus on the effects of the re-host process or tool chain changes.

5.1.3. The Development Organization should perform Nuclear Safety Regression Testing on any component or components the Development Organization has updated since the last certification and meet any of the following definitions:

5.1.3.1. The update affects a component that does not normally impact Critical Signals, but there remains a potential for impact due to unanticipated software interactions. This particularly applies when the updated component interfaces with a component that impacts Critical Signals and the updated component provides information or commands to the component that impacts Critical Signals. In all cases, AFSEC/SEWN should make the determination as part of the Nuclear Safety Design Certification portion of the Nuclear Certification Impact Statement.

5.1.3.2. The update requires a rebuild of nuclear critical software, even without changes to the nuclear critical software source code; or the update requires a rebuild of non-nuclear critical software when that software interacts with nuclear critical software. The Development Organization should provide evidence that the nuclear critical software is identical to the previously certified software, possibly by comparing intermediate build steps.

5.1.3.3. The update affects a software component that is responsible for propagating a Critical Signal but does not impact the Critical Signal. The component does not understand the Critical Signal, and the system appropriately protects the Critical Signal.

5.1.3.4. AFSEC/SEWN determines that a combination of changes to non-certified components, even across multiple software releases, could impact Critical Signals.

5.1.3.5. The update is to test equipment software or test software that does not fall under **paragraph 5.1.1.3** and subparagraphs, but interfaces with or is in the same build as software that falls under those paragraphs and subparagraphs.

5.2. Safety Certification. Any NSCCA-verified or NSATE-verified software component should be safety-certified and the Master Nuclear Certification List should include the component. As a rule, system program offices should list these software components as separate components or combined with other safety-certified components. Combining safety-certified components with components that are not safety-certified into a single application makes it difficult to separate out reasons for changes to the Master Nuclear Certification List. The Master Nuclear Certification List can be found by contacting the Air Force Nuclear Weapons Center, Certification Management Division. Discrepancy Reports are the mechanism the system program office uses to notify the developer and AFSEC/SEWN of a non-compliance. Discrepancy Reports should trace directly to NSRs.

5.2.1. The Verification Organization shall generate Discrepancy Reports for any errors or discrepancies in software design, code, or documentation found against the release candidate during NSATE or NSCCA. **(T-1)** Analysis of any Discrepancy Reports filed against discrepancies found in the release candidate should prove that AFSEC/SEWN or a delegated authority have adjudicated discrepancies as to safety impact, and the Development Organization has adequately addressed or mitigated discrepancies that affect safety. The Verification Organization should provide all Discrepancy Reports to the system program office, but may also provide Discrepancy Reports directly to AFSEC/SEWN. The Discrepancy Reports should not contain information that would allow the Development Organization to determine what verification procedures the Development Organization executed to find the discrepancy.

5.2.2. The Verification Organization shall provide Discrepancy Reports directly to AFSEC/SEWN for any errors or discrepancies found in NSATE or NSCCA that appear to show intentional sabotage. **(T-1)** Analysis of Discrepancy Reports should prove that if errors or discrepancies appear to show intentional sabotage, then the Verification Organization has provided these Discrepancy Reports directly to AFSEC/SEWN.

5.3. General Verification Guidelines. Analysis and inspection methodologies are generally appropriate to show compliance with the Nuclear Safety Design Certification requirements, but methodologies like testing and demonstration that execute the software are more likely to detect anomalous behavior, particularly in complex systems.

5.3.1. The Verification Organization shall perform the final NSCCA or NSATE on production-equivalent platform system hardware executing the production software, or hardware emulations and simulations that are demonstrably equivalent. **(T-1)** Analysis of the Nuclear Safety Software Verification Report should prove the Verification Organization performed verification activities on production-equivalent platform hardware or demonstrably equivalent emulations and simulations.

5.3.2. Verification Organizations do not need to use production-equivalent test equipment or production-equivalent systems that provide one or more interfaces to the production software. Test equipment should be appropriate for the verification needs.

5.3.3. The Development Organization shall perform the final Nuclear Safety Regression Testing on production-equivalent platform system hardware executing the production software. **(T-1)** Analysis of the NSE should prove the Development Organization performed verification activities on production-equivalent platform hardware.

5.3.4. Nuclear Safety Regression Testing is not mutually-exclusive with NSATE or NSCCA verification. In a complex update, it is possible that the Development Organization updates both nuclear critical software and other software that interfaces with nuclear critical software. In that case, AFSEC/SEWN may require the system program office to perform Nuclear Safety Regression Testing on one or more software components and NSATE or NSCCA on one or more software components.

5.3.5. As **paragraph 4.4.3.4** explains, if the Verification Organization recognizes a potential impact due to interfacing software, the Verification Organization can request access to the interfacing software. This would especially apply to new weapon system development.

Chapter 6

SOFTWARE AND CRITICAL DATA TRANSMISSION AND DISTRIBUTION

6.1. Introduction. The Development Organization should design the system to protect Critical Software and Critical Data from inadvertent or intentional manipulation during transport from one secure enclave to another secure enclave.

6.2. Transmission of Critical Software or Critical Data.

6.2.1. The transmission of Critical Software or Critical Data outside of immediate control of the weapon system can become a safety concern if the data is susceptible to intentional or accidental manipulation.

6.2.2. The software developer shall use protocols that protect the transmission of Critical Software via over-the-air broadcasts or transmission over media outside of immediate control of the weapon system from inadvertent or intentional corruption, through encryption, digital signatures, or similar methods. **(T-1)** Verification activities should prove that protocol protection mechanisms protect Critical Software during transmission of over-the-air broadcasts or transmission over media outside of immediate control of the weapon system. If the weapon system stores the Critical Software in an incorruptible manner, and the weapon system verifies the Critical Software during each restart, then this requirement no longer applies. AFSEC/SEWN prefers encryption as the mechanism for protocol protection, but the NSA should approve the encryption methodology. If the software does not properly meet this requirement, then any system that could modify the Critical Software may fall under NSCCA guidelines.

6.3. Physical Media. The operators may use physical data storage such as DVD-ROMs or portable flash drives to store and transport Critical Software and Critical Data between weapon systems or within a weapon system with the following requirements:

6.3.1. The software developer shall use standard, commercially available protocols or military-approved protocols to access physical data storage containing Critical Software or Critical Data. **(T-1)** Verification activities should prove physical data storage access protocols are commercially available or military-approved.

6.3.2. The software developer shall ensure that the software validates all low-level data through standard mathematically deterministic error-detecting and error-correcting operations when accessing physical data storage containing Critical Software or Critical Data. **(T-1)** Verification activities should prove physical data storage access protocols execute error detecting and correcting actions against the data.

6.3.3. The software developer shall protect Critical Software or Critical Data transported on rewritable physical data storage using methods capable of detecting tampering when outside of the control of the weapon system. **(T-1)** Verification activities should prove the rewritable physical data storage format uses encryption, digital signatures, or complex hashes to protect the data.

6.4. Write Once Read Many Media. Software developers can use Write Once Read Many media to store and transport Critical Software and Critical Data between weapon systems or within a weapon system with the following requirements:

6.4.1. The software developer shall use International Organization of Standardization (ISO) 9660:1988/Amendment 2:2020, *Volume and file structure of CD-ROM for information interchange* protocols or equivalent protocols that provide single-session, one-write operations for storing Critical Software and Critical Data to Write Once Read Many media. **(T-1)** Verification activities should prove the format of the data on Write Once Read Many media complies with ISO 9660:1988/Amendment 2:2020 or equivalent protocol and the recording technique is finalized single-session on fresh media.

6.4.2. The software developer shall use only single session recording techniques on fresh media for storing Critical Software and Critical Data to Write Once Read Many media. **(T-1)** Verification activities should prove the format of the data on Write Once Read Many media complies with ISO 9660:1988/Amendment 2:2020 or equivalent protocol and the recording technique is finalized single-session on fresh media.

6.4.3. The software developer shall finalize the recording session for storing Critical Software and Critical Data to Write Once Read Many media to prevent changes to the media after storage. **(T-1)** Verification activities should prove the format of the data on Write Once Read Many media complies with ISO 9660:1988/Amendment 2:2020 or equivalent protocol and the recording technique is finalized single-session on fresh media.

6.4.4. The software developer shall verify Write Once Read Many media by performing a comparison between the original data and the stored data using software different from what the software developer used to write to the media. **(T-1)** Verification activities should prove that the master disk and the copy are functionally identical via bit-for-bit comparison. The comparison should include all data on the media, using software that is different from the software the software developer used to create the media.

6.4.5. The software shall use only high-level file access mechanisms that preclude direct access to the data for reading or storing Critical Software and Critical Data from or to Write Once Read Many media. **(T-1)** Verification activities should prove that the software only uses high-level file access for reading or storing Critical Software and Critical Data. In general, software should only access files on Write Once Read Many media through the operating system file input and output interfaces to avoid malicious covert channel attacks.

Chapter 7

SAFETY AND RELIABILITY

7.1. Introduction. In many systems, safety concerns and reliability concerns appear to be at odds. Nuclear weapon systems should be both safe and reliable.

7.2. Reliability. No requirements in this manual will adversely affect the reliability of the weapon system, and most requirements in this manual directly contribute to the reliability of the overall weapon system.

7.3. Safety.

7.3.1. Most nuclear weapon system safety concerns are not dependent on reliability. If a Critical Signal fails to propagate from human intent to critical function activation, this failure affects the reliability of the system, but this failure does not generally affect the safety of the system. However, the inverse scenario does affect safety. If the weapon system has activated a Critical Signal and the operator withdraws human intent in order to reverse the critical function, a failure to propagate that human intent to the critical function is a safety concern.

7.3.2. If an operator provides human intent to the software through an interface, a failure of the software to propagate the Critical Signal is not a safety concern. The weapon system does not activate a critical function without proper authorization.

7.3.3. In the same scenario, assume the software on the processor correctly propagates the Critical Signal to the Prearm Consent critical function. If the software fails, a removal of human intent to Prearm Consent would result in a safety concern. The removal of human intent should always result in the deactivation of the critical function until the Release or Launch of the weapon.

7.3.4. There are design options that minimize the safety impact of unreliable software in this scenario. An example design option would be for the designers to put in a small device (processor or Programmable Logic Device) between the main processor and the critical function. The small device would have one job, to listen for a "keep alive" message from the processor, and to deactivate all critical functions if the keep alive message ceases for a length of time.

7.3.5. In this scenario, the small device could be very small and easily designed to meet the safety requirements. The designer could minimize the burden of proving reliability of certain software components such as the Real-Time Operating System (RTOS), drivers, or real-time processing components. **Note:** This does not alleviate monitoring requirements but it is a viable approach for Prearm Consent and functionality to Authorize. This manual is concerned with reliability for purposes of safety within the context of reliably returning to a safe state when commanded.

Chapter 8

GENERAL SOFTWARE REQUIREMENTS

8.1. Applicability. The following requirements apply to all software implementations referenced in **Chapter 2**, including programmable logic device implementations and Application Specific Integrated Circuits. The requirements in this chapter apply to all software that undergoes NSCCA or NSATE verification according to this manual. Some requirements define additional caveats for applicability.

8.2. Development Standards. Nuclear systems should comply with well-defined software development standards to ensure software developers use a systematic process that increases confidence for successful nuclear certification.

8.2.1. The software developer shall use a rigorous, disciplined practice that has the characteristics defined in the following subparagraphs. **(T-1)** Verification activities should prove that the software development complies with each portion of the requirement.

8.2.1.1. The software requirements development process defines and enforces procedures to generate nuclear surety and safety requirements. The software requirements development process ensures those requirements are complete, consistent, correct, verifiable, and traceable from relevant requirements in AFI 91-101, AFMAN 91-118, and this manual to the lowest unit of software development, typically the function or procedure.

8.2.1.2. The system and software architecture process and design process define and enforce procedures to develop an architecture and design that isolates surety critical functions from all other system functions.

8.2.1.3. The system and software safety analysis process confirms that the software architecture and design conform to the four Nuclear Surety Standards in DoDD 3150.02 and to all fault detection and recovery requirements in this manual.

8.2.1.4. The coding standards process incorporates and enforces written standards. The standards mandate deterministic and maintainable execution and incorporate or conform to generally accepted practices for safety critical code. Refer to **paragraph 9.2**.

8.2.1.5. The unit development process defines and enforces coding practices and procedures that ensure that all software units are configuration controlled.

8.2.1.6. The unit testing process defines and enforces procedures for frequent automated unit tests, including automated static analysis for conformance to coding standards prior to integration.

8.2.1.7. The integration testing process defines and enforces procedures for integration tests that include nominal and off-nominal test cases to test conformance with surety requirements at higher levels of integration.

8.2.1.8. The development regression testing process defines and enforces development regression testing procedures that ensure that the software developer subjects all changes to configuration-controlled software to the complete testing process as original code. Programs that integrate development regression testing effectively can significantly simplify the Nuclear Safety Regression Testing process when required by AFSEC/SEWN. Refer to **paragraph 4.7**.

8.2.1.9. The defect and issue reporting process defines and enforces complete defect and issue reporting and hazard tracking from initial discovery to closure and resolution.

8.2.1.10. The development process produces records, documents, and artifacts that show conformance with the development process and support an application for Nuclear Safety Design Certification.

8.2.1.11. The artifact review process defines and enforces procedures to review all records, documents, and artifacts produced by the development process to ensure that the records, documents, and artifacts meet the requirements in this manual and the development standards.

8.2.2. If compliance with current standards is not feasible, the software developer shall modify existing nuclear-certified software in compliance with, at a minimum, the standard under which the software developer originally developed it. **(T-1)** For example, the following are standards that may have been used during development of existing nuclear-certified software: DoD-STD-2167A, *Defense System Software Development*; Military-Standard (MIL-STD)-498, *Software Development and Documentation*; Electronic Industries Association/Institute of Electrical and Electronics Engineers (EIA/IEEE) J-STD-016, *Standard for Information Technology Software Life Cycle Processes Software Development Acquirer-Supplier Agreement*. Verification activities should prove that the software developer developed the software in compliance with the same standard under which the software developer originally developed the software.

8.2.3. The software developer shall incorporate nuclear surety design requirements found in AFMAN 91-118 and this manual into requirement specifications and trace nuclear surety design requirements through all levels of requirements specifications, design specifications, implementation, and qualification test documentation. **(T-1)** Refer to **Figure 4.1** for more information. Analysis of the software developer's requirements traceability matrix should prove that the software developer traced nuclear surety design requirements throughout all levels of the documentation.

8.2.4. The software developer shall have a process for both management and engineering activities that the software developer has documented, has standardized across programs, and has integrated into a standard software process for the organization. **(T-1)** Verification activities should prove that the software developer follows an established process for both management and engineering activities.

8.2.5. The software developer should use a defined, reproducible, and rigorous development process that the software developer also consistently uses for production of safety-critical code that complies with architecture, design, coding, and verification requirements of this manual and produces artifacts sufficient to verify compliance.

8.2.6. The software developer shall use secure, non-repudiable configuration management systems to maintain software source code, software products, verification tools, and verification procedures in accordance with the Program Protection Plan. **(T-1)** Verification activities should prove that the software developer protects the configuration management system in accordance with the Program Protection Plan.

8.2.7. The software developer shall design the software in a hierarchical manner as defined in the following subparagraphs. **(T-1)** Verification activities should prove via the software and software design documentation that the software follows a hierarchical design.

8.2.7.1. The software developer has broken down the software by functionality.

8.2.7.2. The software developer has thoroughly defined interactions between components and has limited those interactions to specific functions using data hiding techniques.

8.2.8. The software developer shall design the software such that the lowest-level software components that are directly responsible for Critical Signal initiation or propagation are single-purpose components for initiating or propagating the Critical Signal. **(T-1)** Verification activities should prove that the lowest-level software components that are directly responsible for Critical Signal initiation or propagation are single-purpose.

8.2.9. The software developer shall not use open source or Commercial Off-the-Shelf software unless the source code is available and the software developer has fully reviewed the source code to the level required for certification. **(T-1)** Verification activities should prove that open source software and Commercial Off-the-Shelf software does not contain potentially harmful software. The Verification Organization may consider review documentation of the software as verification. The software developer does not have to analyze RTOS software to the source code level if it meets the certification requirements specific to RTOSs.

8.2.10. While this manual does not require certification for compilers and test software, tools software developers use with nuclear software should be mature and a reputable source or vendor should provide them. Software developers should use the tools in a manner appropriate for the task. AFSEC/SEWN may disqualify some tools or vendors in the development of nuclear critical software based on available threat data.

8.2.11. The software developer shall acquire tools used to translate source code into machine or object code through means that will prevent the intentional insertion of malicious code into a nuclear weapon system. **(T-1)** Verification activities should prove that the software developer acquired the tools through blind buys or by mechanisms that did not allow the tool developer or malicious actors to recognize that the software developer will use the tools to develop a nuclear weapon system. The Verification Organization is not responsible for verifying this requirement if the Development Organization includes these tools under a Supply Chain Risk Management plan.

8.3. Certification Configurations. System program offices should not provide "partial" software releases for certification, or releases that have a known nuclear safety issue, with the intention of providing a later release to satisfy the remaining nuclear safety requirements. A "partial" release contains known defects or issues that may require update prior to certification. AFSEC/SEWN may reject partial releases with known issues, particularly nuclear safety issues. There is no schedule advantage to submitting a partial release for partial certification, since AFSEC/SEWN should certify the release as a whole.

8.3.1. The system program office shall submit the final software configuration for certification only if it fully complies with all nuclear surety requirements that have not been approved as deviations. **(T-1)** Analysis of the verification results and deviation approvals should prove that the weapon system meets all nuclear surety requirements the Certification Requirements Plan defines, or that AFSEC/SEWN has approved any deviation requests. This requirement does not preclude incremental deliveries during development. Programs should not expect AFSEC/SEWN to certify different components apart from the overall software system.

8.3.2. DELETED

8.3.2.1. DELETED

8.3.2.2. DELETED

8.3.2.3. DELETED

8.3.3. The system program office shall field to the operational environment only certified software configurations in accordance with AFI 63-125, *Nuclear Certification Program*. **(T-1)** Verification activities should prove that the certified configuration is identical to the proposed operational configuration. Verification activities may use bit-for-bit comparison demonstrations, for example.

8.4. Self-Modifying Code. The software developer shall design the software to not have the ability to modify its own instructions or the instructions of any other application. **(T-1)** Verification activities should prove that the certified configuration is unable to modify its own instructions or the instructions of other applications. A recommended method of partially meeting this requirement is using memory protections as paragraphs **9.3** and **10.3** provide.

8.5. Program Loading and Initialization.

8.5.1. The software developer shall design the software to execute only after the operational software system loads and verifies all program instructions, programming files, and initialization files. **(T-1)** Verification activities should prove that software only executes after all loading and verification are complete.

8.5.2. The software developer shall design the software to communicate results of the program load verification to the system operators or the crew. **(T-1)** Verification activities should prove that software communicates the results of the program load verification described in **paragraph 8.5.1** to the system operator or the crew, or to external systems with the intent of communicating the results to the system operator or the crew.

8.5.3. The system shall assume programs have not correctly loaded until receiving an affirmative load status. **(T-1)** Verification activities should prove that the system treats failure as the default load status.

8.5.4. The software shall perform volatile memory initialization prior to the execution of the main application. **(T-1)** Verification activities should prove that software performs volatile memory initialization by writing all zeros or a known pattern into memory prior to the execution of the main application.

8.5.5. The software shall load all non-volatile memory with executable code, data, or a non-use pattern that the weapon system detects and processes safely upon execution. **(T-1)** Verification activities should prove that software loads all non-volatile memory with known data; non-use patterns cause the processor to respond in a known manner.

8.6. Memory Protection.

8.6.1. The system shall provide at a minimum hardware double bit error detection and single bit correction on all volatile memory. **(T-1)** Verification activities should prove that hardware provides double bit error detection and single bit correction on all volatile memory.

8.6.2. For memory protection that is software-enabled, the software shall enable at a minimum double bit error detection and single bit correction on all volatile memory. **(T-1)** Verification activities should prove that software enables at a minimum double bit error detection and single bit correction when not automatically enabled by hardware.

8.7. DELETED.

8.8. Memory Wear Protection.

8.8.1. For weapon systems that use certain types of non-volatile memory such as flash memory to store data from one execution cycle to the next, memory wear-leveling is a concern. Wear-leveling causes changes in the addressing of data in the flash memory device, which degrades confidence in the verification activities.

8.8.2. The system shall not implement wear-leveling to limit the effect of erasures to non-volatile memory with memory wear limits when that non-volatile memory is used to store data that can directly affect the operations of software that can affect Critical Signals. **(T-1)** If the software responsible for translating virtual addressing to physical addressing is fully testable by executing tests against every possible virtual and physical address combination, then this requirement is met. The software developer should segregate the translation software; for example, the translation software could be placed in a dedicated Programmable Logic Device. Verification activities should prove that the system does not implement wear-leveling on memory with wear limitations, or that the software developer tests the wear-leveling algorithm for every possible virtual and physical address combination. The software developer can simulate the algorithm testing if the algorithm software is directly accessible by the software developer. This requirement does not apply if the wear-leveled non-volatile memory is only used for initialization sequences before the start of operations that can affect Critical Signals.

8.9. Safe Initialization and Shutdown.

8.9.1. The software shall initialize to a known safe state and verify full functionality of hardware involved in the generation or propagation of Critical Signals. **(T-1)** Verification activities should prove that software properly initializes hardware involved in the generation or propagation of Critical Signals. If the software uses an RTOS, then the RTOS should always initialize to a known safe state.

8.9.2. The system shall ensure a controlled transition to a safe state in the event of unanticipated power loss or system failure. **(T-1)** Verification activities should prove that the system performs a controlled transition to a safe state in the event of unanticipated power loss or system failure.

8.9.3. The software shall set all non-volatile devices involved in the generation or propagation of Critical Signals to a safe state upon controlled shutdown or program termination. **(T-1)** Verification activities should prove that the software properly shuts down non-volatile devices including physical devices such as relays or non-volatile memory devices involved in the generation or propagation of Critical Signals during a controlled shutdown or program termination.

8.9.4. The system shall incorporate hardware watchdog timers on each nuclear critical processor to assist in detecting failures in the software. **(T-1)** Verification activities should prove that each nuclear critical processor has a dedicated hardware watchdog timer.

8.9.5. The software shall respond to a hardware watchdog notification or warning by performing a controlled shutdown and reverting to a safe state. **(T-1)** Verification activities should prove that software responds appropriately to a hardware watchdog notification or warning.

8.10. Fault Detection and Fault Response.

8.10.1. The focus of the weapon system fault detection and response is the potential of faults to impact Critical Signals. Faults may occur due to hardware failures, software defects, or malicious attacks. The software should be able to detect all faults regardless of the cause of the fault.

8.10.2. The software developer shall identify Critical Signal-impacting fault conditions and define maximum acceptable Critical Signal transient and recoverable fault rates. **(T-1)** Analysis of the design documentation should prove that they contain all appropriate fault conditions. The Verification Organization needs to analyze the list for completeness.

8.10.3. The software shall verify all hardware device integrity and operational states that can impact Critical Signals or prevent the propagation of the Safing Command. **(T-1)** Verification activities should prove that the software verifies the integrity and proper state of hardware devices. Tests should include, at a minimum, verification of major types of failures, with at least one failure type per hardware device tested.

8.10.4. The software shall verify functionality of volatile memory prior to use. **(T-1)** Verification activities should prove that the software verifies volatile memory prior to use.

8.10.5. The software shall detect and appropriately respond to Critical Signal-impacting hardware and software faults during all stages of execution; including startup, operation, and shutdown; within the lesser of (1) the maximum operationally acceptable time and prior to the time limit; and (2) the time to any associated irreversible adverse system event. **(T-1)** Verification activities should prove that the software detects hardware and software faults during all stages of execution within an acceptable time and responds to those faults within an acceptable time.

8.10.6. The software shall report the events defined in the following subparagraphs to the operator within the lesser of (1) the maximum operationally acceptable time and prior to the time limit, and (2) the time to any associated irreversible adverse system event. **(T-1)** Verification activities should prove that the software reports the listed events and automated actions to the operator within an acceptable time.

8.10.6.1. Hardware and software faults that could adversely affect the initiation or propagation of Critical Signals.

8.10.6.2. Hardware and software faults that could adversely affect the ability of the weapon system to accurately recognize its own state.

8.10.6.3. The status of any actions the software takes to address the fault.

8.10.7. The software shall revert to a known safe state and stop all Critical Signal propagation when the software detects an unrecoverable Critical Signal-impacting fault. **(T-1)** Verification activities should prove that the software reverts to a known safe state and stops all Critical Signal propagation within an acceptable time when the software detects a Critical Signal-impacting fault.

8.10.8. The software shall revert to a known safe state and stop all Critical Signal propagation when the software detects unauthorized modifications to the executable code during execution. **(T-1)** Verification activities should prove that the software reverts to a known safe state and stops all Critical Signal propagation when the software detects an issue with the integrity of the executable code; the software should also immediately set the global state of the software to a safe state to prevent or interrupt the transmission of Critical Signals. The detection method can be an automated software process or a hardware mechanism such as processor-based runtime integrity checking.

8.10.9. The software that verifies the integrity and operational state of hardware devices through methods that verify interfaces between devices shall verify those interfaces to all layers of the interface protocol. **(T-1)** See **paragraph 8.10.3** Verification activities should prove that the software verifies the interfaces to all protocol levels, not solely the physical electrical values or network information present in the interface.

8.11. Human Intent.

8.11.1. All Critical Signals originate with human intent, the action of an authorized human to initiate a critical function. The system should avoid requiring positive human actions to prevent the initiation or propagation of Critical Signals. The following would be a design paradigm to avoid: a system that automatically prearms a weapon unless the operator activates a switch.

8.11.2. The software shall withdraw or cancel previously issued Critical Signals upon withdrawal of human intent and revert to the state the system was in prior to issuance of the Critical Signal, until the final irreversible action of Launch or Release, within the lesser of (1) the maximum operationally acceptable time and prior to the time limit, and (2) the time to any associated irreversible adverse system event. **(T-1)** Verification activities should prove that the software cancels Critical Signals after withdrawal of human intent through a single human action within an acceptable time.

8.11.3. The system shall require only a single human action for withdrawal of human intent. **(T-1)** Verification activities should prove that the software recognizes a single human action as withdrawal of human intent and an unrecoverable failure as implied human intent and cancels Critical Signals.

8.11.4. The system shall identify improper or incorrect operator entries and notify the operator of the improper entries within the lesser of (1) the maximum operationally acceptable time and prior to the time limit, and (2) the time to any associated irreversible adverse system event. **(T-1)** Verification activities should prove that the software identifies improper operator entries and notifies the operator within an acceptable time.

8.11.5. The software shall issue the final irreversible Critical Signal representing the activation of the associated final irreversible critical function of Launch or Release, only after receiving two distinct human affirmations that the human operator cannot inadvertently actuate. **(T-1)** Verification activities should prove that the software requires two distinct human affirmations that the human operator cannot inadvertently actuate before issuing the final irreversible Critical Signal representing the activation of the associated final irreversible critical function. The software may automate the final irreversible Critical Signal if the software has previously satisfied the human intent requirements in the employment sequence. The following sequences of separate actions meet this requirement: Prearming and Authorization; or Prearming and Release or Launch.

8.11.6. The software shall originate or propagate Critical Signals only if the software recognizes all applicable preconditions. **(T-1)** Verification activities should prove that the software only originates or propagates Critical Signals if and only if the software recognizes all applicable preconditions. Refer to **paragraph 8.12.3**.

8.11.7. The software source code or configuration files shall not contain the pattern that represents the Prearm unique signal within the software in a directly usable form. **(T-1)** Verification activities should prove that the software does not contain the pattern that represents the Prearm unique signal.

8.11.8. The software shall not assemble the Prearm unique signal except as a result of operator action. **(T-1)** Verification activities should prove that the software assembles the Prearm unique signal only after operator action.

8.11.9. The software shall not repair or attempt to repair Prearm unique signals or Authorization codes, except as part of a mathematically deterministic error-correction operation. **(T-1)** Verification activities should prove that the software does not attempt to repair Prearm unique signals or Authorization codes. Mathematically deterministic operations, such as Reed-Solomon encoding, memory error detection and correction, and other forward error correction algorithms, are not applicable to this requirement.

8.11.10. The software shall delete Prearm unique signals or Authorization codes after transmission by writing a value that has no relationship to the original signal or code to all memory locations that contain the signals or codes. **(T-1)** Verification activities should prove that the software writes over Prearm unique signals and Authorization codes with a number that has all bits set to “1,” for example.

8.11.11. The software shall invalidate the navigation information and inhibit the arming of the weapon if a real-time processing error, software reset, or other software failure has compromised the integrity of the target location or weapon location. **(T-1)** Verification activities should prove that software errors or resets cannot adversely affect the integrity of Navigation Validation due to loss or corruption of data.

8.11.12. The software shall always attempt to send a Safing Command to the weapon and weapon system after the software recognizes that the weapon should be made safe due to a failure or the removal of human intent. **(T-1)** Verification activities should prove that the software attempts to send a Safing Command to the weapon and weapon system even if the weapon will be powered down automatically or manually. For this requirement, failures include but are not limited to failures of any component that provides human intent such as a component that reads switches or button presses or failures of any component that propagates Safing Commands throughout the weapon system.

8.11.13. The software shall prioritize the propagation of Safing Commands above all other message types. **(T-1)** Verification activities should prove that the software prioritizes Safing Commands in queues, task priorities, and other constructs to propagate the Safing Commands as quickly as possible.

8.12. Internal Data Formats and Verification.

8.12.1. Due to the importance of Critical Signals in maintaining the safety of the nuclear weapon systems, the format of the Critical Signals, command words, and state representations inside the software is of the utmost concern.

8.12.2. The software shall use data patterns for Critical Signal representations internal to the software, command words internal to the software, and state representations internal to the software that meet the restrictions in the following subparagraphs. **(T-1)** Verification activities should prove that Critical Signal format, command words, and state representations in the software all meet the restrictions preventing inadvertent recognition of an incorrect value. For commands that the software passes through multiple software components, designers should consider values that the software mathematically manipulates within each function. This verifies that all functions have processed the command words correctly:

8.12.2.1. The data patterns are a complex sequence of bit “ones” and “zeros.”

8.12.2.2. The data patterns are not all “ones” or all “zeros.”

8.12.2.3. The data patterns maximize the Hamming distance from other valid values of the data type in question, Critical Signal, command word, or state, while avoiding values that are bitwise complementary.

8.12.2.4. The data patterns are unique, preventing the software from accidentally using a value of one data type in a different data type.

8.12.2.5. The data patterns are such that errors containing two-bit flips do not result in a valid value of that data type. It is acceptable to use multiple sets of patterns in order to meet this restriction, effectively creating much larger base types. For example, two 32-bit variables may be set and read together as one variable, even if the variables are not contiguous in memory.

8.12.3. The software subroutines that are directly responsible for Critical Signal initiation or propagation shall initiate or propagate the Critical Signal only after verifying states and preconditions to ensure that the software executes the low-level components only with proper authorization. **(T-1)** Verification activities should prove that the software subroutines - functions or procedures, or other types of subroutines - that are directly responsible for Critical Signal initiation or propagation verify states and preconditions prior to initiating or propagating Critical Signals.

8.12.4. The software that represents the lowest-level software components that are directly responsible for Critical Signal initiation or propagation shall notify the operator if state and precondition verification fail within the lesser of (1) the maximum operationally acceptable time and prior to the time limit; and (2) the time to any associated irreversible adverse system event. **(T-1)** Verification activities should prove that the software components that are directly responsible for Critical Signal initiation or propagation notify the operator within an acceptable time when state and precondition verification fails.

8.13. External Data Formats and Verification.

8.13.1. The weapon system should protect Critical Signals as they travel throughout a weapon system. The weapon system design should protect Critical Signals in manners similar to those described in **Chapter 3**, but also have additional protective mechanisms that ensure that malicious actors have not intentionally altered the Critical Signals.

8.13.2. The software shall use data patterns for Critical Signal representations throughout the weapon system and in transit between weapon system components that meet the restrictions in the following subparagraphs. **(T-1)** Verification activities should prove that the Critical Signal data patterns throughout the weapon system and in transit between weapon system components meet the restrictions preventing inadvertent recognition of an incorrect value.

8.13.2.1. The data patterns are complex sequence of bit "ones" and "zeros."

8.13.2.2. The data patterns are not all "ones" or all "zeros."

8.13.2.3. The data patterns maximize the Hamming distance from other valid values while avoiding values that are bitwise complementary.

8.13.2.4. The data patterns are such that errors containing two-bit flips do not result in a valid value.

8.13.3. The software shall use message formats for Critical Signals throughout the weapon system and in transit between weapon system components such that the Critical Signal messages meet the restrictions in the following subparagraphs. **(T-1)** Verification activities should prove that Critical Signal message formats throughout the weapon system and in transit between weapon system components meet the restrictions preventing inadvertent or malicious corruption.

8.13.3.1. Critical Signal messages use message protection mechanisms to prevent inadvertent or intentional corruption of the Critical Signal.

8.13.3.2. A malicious actor cannot spoof Critical Signal messages, particularly on shared bus architectures or over-the-air broadcasts.

8.13.4. The software shall use message formats, message-protection mechanisms, and identification methods that ensure the receiver authenticates the sender as a valid sender for transmission of Critical Signals via over-the-air broadcasts or transmission over media outside of immediate control of the weapon system. **(T-1)** Verification activities should prove that the weapon system protects Critical Signals for transmission of over-the-air broadcasts or transmission over media outside of immediate control of the weapon system.

8.13.5. The software shall validate message content down to the bit level for all aspects of any data provided to the software on interfaces with non-USAF-certified items. **(T-1)** This includes NSA-certified hardware, DOE-qualified hardware, and any other items included in the weapon system that could affect Critical Signals, such as Inertial Measurement Units, flight control systems, or environmental sensors. Verification activities should prove that the software uses validation methods on the message content of interfaces with non-USAF hardware that meet the following guidelines:

8.13.5.1. Software validates the data down to the bit level.

8.13.5.2. Software checks each data type in the message for valid values.

8.13.5.3. If certain portions of the message interpret other portions of the message, then the software should sequentially validate the message by meaning; if one part of the message provides information on how to interpret another part of the message, then the software should process the message by sections in order to validate fully each section.

8.13.6. The software shall validate the message content of any Critical Signals the software has received from any location down to the bit level. **(T-1)** Verification activities should prove that the software uses validation methods on the message content of any Critical Signals that meet the following guidelines:

8.13.6.1. Software validates the data down to the bit level.

8.13.6.2. Software checks each data type in the message for valid values.

8.13.6.3. If certain portions of the message interpret other portions of the message, then the software should sequentially validate the message by meaning; if one part of the message provides information on how to interpret another part of the message, then the software should process the message by sections in order to validate fully each section.

8.13.7. The software shall delete any data representing Critical Signals other than the Safing Command that the software has received from outside of the software immediately after receipt and validation of the Critical Signal by writing a value that has no relationship to the original Critical Signal data to all memory locations that contain the Critical Signal data. **(T-1)** Verification activities should prove that the software deletes Critical Signal representations immediately after receipt and validation.

8.13.8. The software shall delete any data representing Critical Signals other than the Safing Command that the software has transmitted outside of the software immediately after transmission of the Critical Signal by writing a value that has no relationship to the original Critical Signal data to all memory locations that contain the Critical Signal data. **(T-1)** Verification activities should prove that the software deletes Critical Signal representations immediately after transmission.

8.14. Security.

8.14.1. The software should use secure methods to validate that the software applications are the approved, trusted applications.

8.14.2. The software shall use secure methods like root-of-trust and cryptographic secure boot to ensure that the weapon system executes only trusted applications. **(T-1)** Verification activities should prove that the weapon system does not execute any untrusted software prior to trusted validation.

8.15. Coding Prohibitions and Analysis. Safe software avoids certain coding constructs and paradigms. Development Organizations and Verification Organizations can use commercially available static analysis tools for compliance with most or all of these requirements, as well as coding standard requirements.

8.15.1. The software shall not contain unused code. **(T-1)** Verification activities should prove that the software does not contain unused code with the following clarifications:

8.15.1.1. The software developer should still test source code that is obsolete due to changes to the weapon system. The software developer should remove obsolete code that is untestable.

8.15.1.2. The software developer should document in the Nuclear Surety Evaluation source code that is untestable due to the nature of the code, error handling in the code or low-level source code the software developer designed for very specific scenarios.

8.15.1.3. The preferred unused code analysis is statement coverage testing. This form of dynamic analysis is a conclusive method for proving that software does not contain unused code.

8.15.1.4. Fuzzing is also a recommended tool to ensure coverage-testing focuses only on necessary sections of code. For Complex Programmable Logic Devices and Application Specific Integrated Circuits, the software or build process disables functions in intellectual property sections at interface boundaries.

8.15.1.5. The software developer may use synthesis optimization to eliminate functions that the software cannot reach if this option is available. This requirement applies to spare gates and unused gate array cells only so far as to require the software developer to wire these inactive in layout.

8.15.2. The software shall not contain global variables. **(T-1)** Verification activities should prove that the software does not contain global variables.

8.15.3. The software shall not contain uninitialized variables. **(T-1)** Verification activities should prove that the software does not contain uninitialized variables.

8.15.4. The software shall bound the depth of recursive functions such that recursion does not exhaust available resources. **(T-1)** Verification activities should prove that the software does not contain unbounded recursion – the repeated application of a function with no built-in logical limit – or recursion that could exhaust resources such as stack memory.

8.15.5. The software shall not execute memory allocation functions after application initialization. **(T-1)** Verification activities should prove that the software only executes memory allocation functions before or during application initialization.

8.16. Real-Time Processing. The purpose of this section is to ensure that the software does not lockup or shut down unexpectedly due to Real-Time Processing defects. The speed at which the weapon system processes Critical Signals other than the Safing Command is not a safety concern.

8.16.1. The software shall implement strict scheduling algorithms and task prioritization to avoid task lockup scenarios and meet response time requirements on the target processor under worst-case conditions if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software includes scheduling algorithms, task priorities and other such information that may be necessary to prove that the software should meet such deadlines under all system load and interrupt processing conditions. The software developer may do this by analyzing rate monotonic analyses performed against the software.

8.16.2. The software shall not use any paradigm or abstraction that contributes to non-deterministic execution sequences if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software does not use any paradigm or abstraction that contributes to non-deterministic execution sequences or non-static memory allocation such as distributed resource scheduling, dynamic memory allocation, runtime allocation of processors or cores, non-reentrant subroutines, or any algorithm that uses random execution.

8.16.3. The software shall implement mechanisms to prevent or resolve deadlock conditions if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software implements mechanisms such as priority inversion or task-assigned resources to prevent or resolve deadlock conditions.

8.17. Aircraft-Unique Requirements. The following requirements are unique to aircraft that deliver nuclear weapons:

8.17.1. The software shall verify the transmission of Critical Signals prior to the final irreversible action. **(T-1)** Verification activities should prove that the software verifies the transmission of Critical Signals prior to Release.

8.17.2. If the in-flight reversible locks are under software control, the software shall have a unique control or control setting for locking and unlocking the in-flight reversible lock separate from the Release controls. **(T-1)** Verification activities should prove that the software control for locking and unlocking the in-flight reversible lock is unique and separate from the Release controls.

8.17.3. After the software recognizes that the in-flight reversible lock should be locked due to failure or human action, the software shall attempt to send a lock command to the in-flight reversible lock. **(T-1)** Verification activities should prove that the software attempts to send a lock command to the in-flight reversible lock even if the in-flight reversible lock is expected to lock due to other weapon system actions.

8.18. Multifunction Control and Displays. For Aircraft Monitoring and Control systems with multifunction controls and displays, the designer should ensure that the displays are unambiguous in their representation of weapon system controls or monitor information, or the use of the weapon system controls. Per **paragraph 9.5.7**, Aircraft Monitoring and Control systems should use Avionics Application Standard Software Interface (ARINC) 653 certifiable operating systems, particularly for controls and displays.

8.18.1. For legends and controls for Aircraft Monitoring and Control systems:

8.18.1.1. The software shall display nuclear weapon screen legends next to control buttons if and only if the control button is capable of initiating a function. **(T-1)** Verification activities should prove that the software displays screen legends if and only if the control button is capable of initiating a function.

8.18.1.2. The system shall provide separate controls or multiple unique button presses for the activation of individual nuclear weapon commands. **(T-1)** Verification activities should prove that the software requires users to use separate controls, a toggle switch, or multiple unique button presses to activate individual nuclear weapon commands. For example, MONITOR should not become SAFE, LOCK should not become UNLOCK, and Prearm

Consent should not become enabled or disabled by the same subsequent action on the control or button.

8.18.2. For legends and controls for multi-crew aircraft:

8.18.2.1. The software shall restrict weapon system control to one aircrew member at a time. **(T-1)** Verification activities should prove that the software allows only one aircrew member to control the weapon system at a time. Only one aircrew member at a time can enter targeting data.

8.18.2.2. The software shall provide a mechanism to allow the transfer of weapon system control between aircrew members. **(T-1)** Verification activities should prove that the software allows aircrew members to assume control of the weapon system from each other.

8.18.3. For Aircraft Monitoring and Control systems dedicated displays:

8.18.3.1. The system shall provide at least one dedicated display or a dedicated positionally static display area of a larger display for monitoring the weapon status. **(T-1)** Verification activities should prove that the software system provides at least one dedicated display or a dedicated area of a large display that stays in the same position for monitoring the weapon status.

8.18.3.2. The software shall provide an advisory system that only human acknowledgment can remove to alert the aircrew to anomalous nuclear weapon system conditions. **(T-1)** Verification activities should prove that the software system provides an advisory system that alerts the crew when it detects an anomalous nuclear weapon system condition.

8.18.3.3. If the system uses a non-static display or display area for nuclear weapon control, the software shall provide an unambiguous notification that the display is now providing nuclear weapon control. **(T-1)** Verification activities should prove that the software system provides a notification that unambiguously alerts the crew when a display begins providing nuclear weapons control.

8.18.3.4. If the system uses a non-static display or display area for nuclear weapon control, the software shall allow control activations if and only if the display is providing nuclear weapon control. **(T-1)** Verification activities should prove that the software control activations occur only when a display is providing nuclear weapon control.

8.18.3.5. The software shall provide unambiguous formatting differences between different critical function selections on the weapon system control display to indicate the different critical functions. **(T-1)** Verification activities should prove that the software provides unambiguous differences between the different critical function selections on the weapon system control display.

8.19. Essential Facility System Requirements.

8.19.1. The following requirement is unique to Maintenance, Handling, and Storage Facilities.

8.19.2. The system shall provide both audible and visual alarm mechanisms. **(T-1)** Verification activities should prove that emergency scenarios result in both an audible and visual alarm.

Chapter 9

APPLICATION SOFTWARE REQUIREMENTS

9.1. Applicability. The following requirements apply to application software specifically. Application software executes on processors, microprocessors, embedded microcontrollers, and other hardware technologies that operate on sequential commands. The requirements in this chapter apply to all software that undergoes NSCCA or NSATE verification according to this manual. Some requirements define additional caveats for applicability.

9.2. Programming Languages. Software developers should design and develop software to be reproducible during compilation and during execution. Just-in-time compilation is not acceptable due to the variability of the executable from one execution to the next. To that end, Ada, C, and C++ are the only recommended programming languages for development of new software. Assembly language is acceptable if AFSEC/SEWN approves it. Examples of assembly language usage that AFSEC/SEWN would approve include low-level interfaces, boot code, and time-critical software. AFSEC/SEWN discourages the use of dynamic linking in general, but particularly if the execution location of the object code changes during different executions. See **paragraph 9.2.5**.

9.2.1. Software developers should use the standard library functions a development environment or compiler provides, rather than locally developed library functions. If there is a compelling reason to change or add to the standard library functions, the software developer may locally develop library functions. Compelling reasons for developing local library functions include defects or type incompatibility in a standard library function.

9.2.2. The software developer shall use an internationally recognized, standardized, and mature programming language. **(T-1)** Verification activities should prove that the software uses only appropriate programming language(s), specifically Ada, C, or C++.

9.2.3. The software developer shall not use assembly language in software applications without prior AFSEC/SEWN approval. **(T-1)** Verification activities should prove that the software does not use assembly language. If the software does use assembly language, the software developer should provide evidence to ensure that AFSEC/SEWN has approved the use of assembly language. As noted in section **paragraph 9.5.5**, assembly language used in RTOSs is exempt from this justification requirement.

9.2.4. The software developer shall use the original programming language when modifying software unless the software developer has obtained prior AFSEC/SEWN approval. **(T-1)** Verification activities should prove that the versions of languages and development environment tools are consistent across software versions.

9.2.5. The software developer shall develop software that always executes using the same static memory map during each execution, power cycle, software reset, or processor reset. **(T-1)** Verification activities should prove that the software does not use just-in-time compilation or run-time dependent configurations and always executes from the same memory map during each execution.

9.2.6. The software developer shall use industry-standard software coding guidelines and conventions to reduce safety concerns, such as Motor Industry Software Reliability Association standards for C and C++ revision current at time of implementation or the Ravenscar profile for Ada. **(T-1)** Verification activities should prove that the software meets an industry-standard AFSEC/SEWN approved software coding guideline and convention. The software developer should justify variances from the standard. The software developer should use an automated tool to verify that the developed software meets the industry-standard software development guidelines and conventions.

9.3. Memory Characteristics of Application Software. At a minimum, processors should have a Memory Management Unit to assign access permissions to specific areas of hardware, but AFSEC/SEWN prefers keyed access control.

9.3.1. The software shall enable memory protection through a memory management unit at a minimum to the access permission levels found in the following subparagraphs. **(T-1)** Verification activities should prove that the software appropriately enables memory protection through a memory management unit. The software developer may use other mechanisms besides a memory management unit, but the software developer should prove the effectiveness of the mechanisms.

9.3.1.1. Object code. Read only, Executable.

9.3.1.2. Variables. Read/Write, Non-Executable.

9.3.1.3. Constants. Read only, Non-Executable.

9.4. Real-Time Embedded Systems. An embedded system is a processor with software that usually serves a dedicated function within a larger system. Embedded systems rarely have traditional user interfaces typical of computing systems and are not as multipurpose as other computing systems. Embedded systems have specific software safety concerns. Real-time software has strict timing constraints that require the system to react to inputs within a certain amount of time. Real-time software uses operating systems and software paradigms to maintain the timing constraints.

9.4.1. Embedded systems in the nuclear mission usually contain real-time software. Per **paragraph 8.16**, Real-Time Processing is only nuclear critical if a failure of the software can lead to the failure to cancel a Critical Signal after an operator has withdrawn human intent or if a failure of the software can lead to a failure to issue a Safing Command. If the design precludes these scenarios, then Real-Time Processing requirements are not applicable.

9.4.2. The software shall not use any instruction intended to cause the processor to pause or stop operations if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software does not use any instruction intended to cause the processor to pause or stop operations. Allowable exceptions to this requirement include wait states associated with memory access time, exceptions that the software cannot handle such as multiple bit errors in volatile memory, or anti-tamper protections.

9.4.3. The software shall handle all exceptions that would cause the processor to stop operations if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software handles all exceptions that would cause the processor to pause or stop operations.

9.4.4. If the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command, then the software should not disable interrupts for longer than necessary to process time-critical sections.

9.5. Real-Time Operating Systems (RTOS) and Run-Time Systems.

9.5.1. An RTOS is a software application that provides operating system services and specifically provides real-time tasking to minimize the latency of input-to-output functions. If the software uses an RTOS, this section enforces specific requirements to protect the system from malicious actors. These requirements apply if a failure of the software can lead to the failure to deactivate a Critical Signal after the withdrawal of human intent or the software is responsible for providing a Safing Command.

9.5.2. A Run-Time System is a software component that provides execution services. In many cases, Run-Time Systems are specific to a particular programming language. In general, the C Run-Time System is fairly small, the C++ Run-Time System is bigger due to the need to instantiate classes and other object-oriented constructs, and the Ada Run-Time System is fairly large because it instantiates object-oriented constructs and maintains a tasking environment.

9.5.3. This section only specifies requirements for RTOSs, but the same requirements apply to equivalent functionality found in Run-Time Systems.

9.5.4. ARINC 653 is a software standard that defines space and time partitioning in operating systems. An ARINC 653-compliant RTOS allows software in one partition to be safe from interference by software in a different partition.

9.5.5. A Board Support Package is the layer of software that contains the interfaces between higher-level software and the hardware. Software developers often refer to Board Support Packages as “drivers.” **Note:** The software developer does not need to justify assembly language included as part of an RTOS or manufacturer-provided Board Support Package in the same manner as assembly language sections defined by the software developer.

9.5.6. The RTOS shall be mature and commercially licensed if the software uses an RTOS. **(T-1)** Verification activities should prove that the RTOS manufacturer has made the RTOS commercially available for at least five years, updated the core functionality of the scheduler less than an average of twice a year for the last five years, added new functionality to the RTOS at least once a year, fixed defects in the RTOS at least once a year, and provided active support for commercial licensing of the RTOS.

9.5.7. The software shall conform to ARINC 653 or equivalent spatial partitioning if the software uses an RTOS. **(T-1)** Verification activities should prove that the software uses an ARINC 653-conforming RTOS (at least Supplement 1) with at least one user partition separate from the kernel. Refer to **paragraph 3.3**.

9.5.8. The software shall load and execute the RTOS immediately after the software completes all boot operations and prior to the application execution if the software uses an RTOS. **(T-1)** Verification activities should prove that the software loads the RTOS immediately after the software completes all boot operations and prior to the application execution.

9.5.9. The software shall transition into a known safe state, cancel Critical Signals, and require a restart, automatic or manual, after experiencing either a stack overflow or a failure in real-time operations if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software transitions into a known safe state, cancels Critical Signals, and requires a restart upon stack overflows or real-time operation failures.

9.5.10. The software shall report failure to maintain real-time operations or stack overflow to the operator if the software is responsible for cancelling a Critical Signal after the withdrawal of human intent or if the software is responsible for issuing a Safing Command. **(T-1)** Verification activities should prove that the software reports real-time operations failures to the operator.

9.5.11. The software developer shall disable all configuration options in the RTOS that are not necessary for software operation if the software uses an RTOS. **(T-1)** Verification activities should prove that the RTOS kernel or main application has all unnecessary configuration options removed. For example, the software developer should remove network stack capabilities when the software does not need a network stack.

9.5.12. If the software uses an RTOS, the software developer shall use the requirements in this document to develop Board Support Packages or drivers for portions of the Board Support Package or drivers that the manufacturer of the RTOS did not provide. **(T-1)** Verification activities should prove that new or updated sections of the manufacturer-provided Board Support Package meet all requirements in this document.

9.6. General-Purpose Computer Systems. A general-purpose computer system is a multi-purpose computer that a single individual uses in a single location or multiple users use across multiple locations, as compared to an embedded or specific-purpose computer system. Developers of general-purpose computer systems may host them on different computer systems from single board computers to large workstations. General-purpose computer systems generally cannot maintain hard real-time constraints due to the use of operating systems that cannot execute hard real-time operations. Due to the complexities involved in general-purpose computer systems, system program offices should use general-purpose computer systems for limited applications such as test equipment or for verifying security credentials.

9.6.1. The general-purpose computer operating system shall be compliant with the most current United States Government Approved Protection Profile for General Purpose Operating Systems which the National Information Assurance Partnership manages. **(T-1)** Verification activities should prove that the general-purpose computer operating system is compliant with the most current profile in the National Information Assurance Partnership list or USAF information security approves the operating system.

9.6.2. The weapon system shall not use a general-purpose computer system in components that impact Critical Signals. **(T-1)** Verification activities should prove that the general-purpose computer system does not impact Critical Signals. In situations where it is necessary to provide Critical Signal data through a general-purpose computer system, data validation can use a separately configured general-purpose computer system.

9.6.3. If the general-purpose computer system uses an antivirus, the general-purpose computer system shall use antivirus applications that USAF information security approves. **(T-1)** Verification activities should prove that USAF information security has approved the antivirus application.

9.6.4. The general-purpose computer system shall use a whitelist application to prevent the execution of unauthorized applications. **(T-1)** Verification activities should prove that the general-purpose computer system contains a configured whitelist application to reject all unapproved applications.

9.6.5. The general-purpose computer system shall physically disable or remove unused peripheral interfaces. **(T-1)** Verification activities should prove that the general-purpose computer system disables or removes unused peripheral interfaces. AFSEC/SEWN encourages system program offices to lock out unused peripheral interfaces by removal or by physical methods. System program offices may also use software protective technologies the NSA provides or approves.

9.6.6. The general-purpose computer system shall restrict users to the lowest-level account privileges necessary to perform the required actions, known as the principle of least privilege. **(T-1)** Verification activities should prove that the general-purpose computer system restricts users to the lowest-level account privileges necessary.

9.6.7. The general-purpose computer system shall not allow users to have root or administrator privileges. **(T-1)** Verification activities should prove that the general-purpose computer system restricts users from having root or administrator privileges except through mediation efforts such as difficult physical access and controlled administrator or root passwords.

Chapter 10

PROGRAMMABLE LOGIC DEVICE REQUIREMENTS

10.1. Applicability. The following requirements apply to programmable logic device-specific implementations and Application Specific Integrated Circuits. The requirements in this chapter apply to all software that undergoes NSCCA or NSATE verification according to this manual. Some requirements define additional caveats for applicability.

10.2. Hardware Description Languages. Very High-Speed Integrated Circuit Hardware Description Language (VHDL) is the recommended hardware description language for development of nuclear critical Programmable Logic Devices.

10.2.1. The software developer shall use an internationally recognized and standardized hardware description language that compiles to programming files for placement on a Programmable Logic Device, or to the fabrication level for an Application Specific Integrated Circuit. **(T-1)** Verification activities should prove that the software uses only appropriate hardware description language(s). VHDL meets this requirement.

10.2.2. The software developer shall use the original hardware description language when modifying software unless the software developer has obtained prior AFSEC/SEWN approval. **(T-1)** Verification activities should prove that the versions of hardware description languages are consistent across versions.

10.2.3. The software developer shall use Programmable Logic Device coding development guidelines and conventions to reduce safety concerns. **(T-1)** Verification activities should prove that the software meets an industry-standard AFSEC/SEWN approved software coding guideline and convention. The software developer should justify variances from the standard. The software developer should use an automated tool to verify that the developed software meets the development guidelines and conventions.

10.3. Memory Characteristics of Programmable Logic Devices.

10.3.1. AFSEC/SEWN prefers one-time fused Field Programmable Gate Arrays and Complex Programmable Logic Devices. Software developers often build Programmable Logic Devices on top of either volatile or non-volatile memory. The memory containing the programming file before the loading mechanism loads it into the Field Programmable Gate Array should meet the requirements laid out in the General Software section. Configuration memory does not have restrictions; incorrect configuration generally makes the Programmable Logic Device inoperable. The memory built into the Field Programmable Gate Array that receives the programming file is exempt from memory restrictions apart from the following requirement.

10.3.2. The application software shall meet the requirements in this manual if a processor is instantiated in a Programmable Logic Device or Application Specific Integrated Circuit and the instantiated processor executes application software. **(T-1)** Verification activities should prove that the software meets the requirements in this manual, particularly the general software requirements in **Chapter 8**, and the application software requirements in **Chapter 9** if a processor is instantiated in a Programmable Logic Device or an Application Specific Integrated Circuit.

10.4. Programmable Logic Device Verification. Programmable Logic Devices are complex, and both users and malicious actors can easily program them. Because of these facts, the Development Organization and Verification Organization should verify Programmable Logic Devices throughout the development lifecycle. The Development Organization performs the verification as part of the development. The Verification Organization performs the verification as part of NSCCA or NSATE.

10.4.1. The weapon system shall perform a check on Programmable Logic Device software to ensure that accidental or malicious actions have not modified the software without authorization during each load and execution cycle for loaded program files. **(T-1)** Verification activities should prove that the weapon system software performs a check during each load cycle. The software developer can meet this requirement by permanently writing or fusing a Programmable Logic Device.

10.4.2. The Programmable Logic Device software shall provide a verification mechanism during the manufacturing process for fused Programmable Logic Devices. **(T-1)** Verification activities should prove that the software provides a verification mechanism during the manufacturing process. Fused devices use a mechanism to permanently retain the programming file.

10.4.3. The Verification Organization shall verify that the programming file matches the input hardware description language of the Programmable Logic Device build process. **(T-1)** Analysis of the documentation should prove that the software developers verified the hardware description language and the resulting programming file. For example, the software developer could have verified the hardware description language matches the programming file by using netlist equivalency prior to the programming file.

10.5. Application Specific Integrated Circuit Verification.

10.5.1. The Development Organization and the Verification Organization should verify the Application Specific Integrated Circuits using methods that are more sophisticated to ensure that accidental or malicious actions have not added any logic to the Application Specific Integrated Circuit after the Development Organization provided the fabrication data to the manufacturer.

10.5.2. The Verification Organization shall verify that the production Application Specific Integrated Circuit is logically equivalent to the input hardware description language source code. **(T-1)** Verification activities should prove that the Application Specific Integrated Circuit logically matches the source code.

JOHN T. RAUCH
Major General, USAF
Chief of Safety

Attachment 1

GLOSSARY OF REFERENCES AND SUPPORTING INFORMATION

References

AFI 33-322, *Records Management and Information Governance Program*, 23 March 2020

AFI 63-125, *Nuclear Certification Program*, 16 January 2020

DAFI 91-101, *Air Force Nuclear Weapons Surety Program*, 26 March 2020

AFMAN 33-363, *Management of Records*, 1 March 2008

AFMAN 91-118, *Safety Design and Evaluation Criteria for Nuclear Weapon Systems*, 13 March 2020

AFPD 91-1, *Nuclear Weapons and Systems Surety*, 24 November 2019

AFPD 13-5, *Air Force Nuclear Mission*, 17 July 2018

ARINC 653, *Avionics Application Software Standard Interface*, Supplement 2, January 2007

DAFMAN 90-161, *Publishing Processing and Procedures*, 15 April 2022

DoDD 5210.41, *Security Policy for Protecting Nuclear Weapons*, 31 August 2018

DoDD 3150.02, *DoD Nuclear Weapons Surety Program*, 31 August 2018

DoDI S-5200.16, *Objectives and Minimum Standards for Communications Security (COMSEC) Measures Used in Nuclear Command and Control (NC2) Communications (U)*, 27 September 2019

DoD-STD-2167A, *Defense System Software Development*, 29 February 1988

EIA/IEEE J-STD-016, *Standard for Information Technology Software Life Cycle Processes Software Development Acquirer-Supplier Agreement*, 13 March 1996

ISO 9660:1988/Amendment 2:2020, *Volume and file structure of CD-ROM for information interchange,* April 2020

MIL-STD-498, *Software Development and Documentation*, 8 November 1994

Adopted Forms

Air Force Form 679, *Air Force Publication Compliance Item Waiver Request/Approval*

Air Force Form 847, *Recommendation for Change of Publication*

Abbreviations and Acronyms

AFI—Air Force Instruction

AFMAN—Air Force Manual

AFPD—Air Force Policy Directive

AFSEC—Air Force Safety Center

AFSEC/SEWN—Air Force Safety Center, Nuclear Weapon Safety

ARINC—Aeronautical Radio, Inc.

DAFGM—Department of the Air Force Guidance Memorandum

DAFI—Department of the Air Force Instruction

DevOps—Development Operations

DevSecOps—Development Security Operations

DoD—Department of Defense

DoDD—Department of Defense Directive

DoDI—Department of Defense Instruction

DoDM—Department of Defense Manual

DOE—Department of Energy

EIA—Electronic Industries Alliance

IEEE—Institute of Electrical and Electronics Engineers

ISO—International Organization of Standardization

MAJCOM—Major Command

MIL-STD—Military Standard

NSA—National Security Agency

NSATE—Nuclear Safety Analysis and Technical Evaluation

NSCCA—Nuclear Safety Cross-Check Analysis

NSE—Nuclear Surety Evaluation

NSO—Nuclear Safety Objective

NSR—Nuclear Safety Requirement

OPR—Office of Primary Responsibility

RTOS—Real-Time Operating System

TNT—Trinitrotoluene

USAF—United States Air Force

USSF—United States Space Force

Terms

Consent—A function implemented by a deliberate human action that is a necessary, reversible step in the process of using a nuclear weapon.

DevOps/DevSecOps—Development + Operations, Development + Security + Operations; the processes and tools which allow organizations to release and deliver software faster and with more immediate feedback from the user. DevSecOps includes steps to ensure that organizations include good security practices throughout the software development, not just at the end of the development.

Nuclear Critical Software—A designation for software or other components of a nuclear weapon system that could cause an unauthorized nuclear Critical Signal and the unauthorized activation of a critical function due to inadvertent, unauthorized, or malicious actions; or that could prevent a Safing Command. Specifically, software components that can impact Critical Signals, either by initiating the Critical Signal or by directly affecting the contents of the Critical Signal or inhibiting the Safing Command.

Nuclear Safety Objective—A high-level verification goal that a nuclear weapon system should satisfy in order to obtain nuclear safety certification.

Nuclear Safety Requirement—A lower-level functional verification goal that a nuclear weapon system should satisfy in order to obtain nuclear safety certification.

Prearm and Prearm Consent—A function implemented by a deliberate human action that is a necessary, reversible first step in the process of using a nuclear weapon.

Real-Time Operating System—Software application that provides common services and specifically provides tasking capabilities to minimize the latency of input-to-output functions.

Safe/Safing—The act of placing a nuclear weapon in a safe state, a state such that the receipt of the final irreversible action does not cause a nuclear detonation, as defined in AFI 91-101.

Safing Command—Critical Signal that causes the weapon system and the weapon to go into a safe state.

System Program Office—Organization led by the Program Manager, as defined in AFI 91-101.

Zeroize—The practice of erasing or obliterating sensitive information (electronically stored data, cryptographic keys, and other critical security parameters) from memory technology to prevent unauthorized disclosure or compromise.

Attachment 2

SOFTWARE DEVELOPMENT LIFECYCLE REDUCTION AND CONTINUOUS DELIVERY

A2.1. Philosophy.

A2.1.1. This attachment describes a framework under which a system program office may use continuous delivery methodologies in nuclear weapon systems. The system program office is still responsible for meeting all requirements in this manual, as allocated by the Certification Requirements Plan.

A2.2. Software Development and Software Release Methodologies.

A2.2.1. **Paragraph 8.2** , and all subparagraphs, provide requirements for the software development process. This manual does not provide direction as to what kind of software development process a system program office should use, as long as the software development process meets the requirements contained herein. This includes traditional forms of software development such as Agile, waterfall development, or spiral development.

A2.2.2. **Paragraph 8.3** , and all subparagraphs, provide requirements for software release methodologies. It clarifies that the Nuclear Safety Design Certification process does not accept partial releases for certification. Per paragraphs **8.3.1** and **8.3.3**, each software release that the system program office intends to field to the operational environment should be a complete release. This manual does not provide direction as to what kind of release process a system program office should use, as long as the software release process meets the requirements contained in **paragraph 8.2**, and all subparagraphs. This could include DevOps, DevSecOps, and similar continuous delivery methodologies. These methodologies generally work by reducing overhead in the transition of software products from development to production environments. Continuous delivery methodologies assume the following:

A2.2.2.1. Testing and verification the software developer performs in development is sufficient.

A2.2.2.2. Testing and verification the software developer performs in the release process, which includes automated testing the software developer performs prior to moving the release to production, is sufficient.

A2.2.2.3. Additional verification required by the system program office, such as verification and validation or flight-testing, is sufficient.

A2.3. Nuclear Safety Design Certification.

A2.3.1. The Nuclear Safety Design Certification process encompasses more than the software development process and the software release process. It is not possible to certify individual components of nuclear critical software. This is due to the following reasons:

A2.3.1.1. Certifying individual software components does not take into account the interactions between distinct nuclear critical software components as part of a total system.

A2.3.1.2. Certifying individual software components does not take into account the interactions between nuclear critical software components and non-nuclear critical software components where there are interfaces between those component types.

A2.3.1.3. Although a single nuclear critical software component may not have safety concerns, when combined with other software components on a single processor or in a single memory partition, the overall nuclear critical software may be unsafe and risk a potential violation of DoD Nuclear Surety Standards.

A2.3.2. For nuclear critical software, Nuclear Safety Design Certification requires additional verification by an independent Verification Organization. Refer to **Chapter 4**. The assumption that the testing and verification the system program office performed are sufficient is contrary to DoDD 3150.02 and the established USAF nuclear certification process. DoDD 3150.02 clarifies that nuclear weapons require special consideration, and that certifying agencies should apply additional safeguards to ensure that nuclear weapons have positive measures in place to comply with the DoD Nuclear Surety Standards.

A2.3.3. The use of an independent Verification Organization is not unique to the USAF nuclear weapons surety program. Aircraft and aerospace systems use independent verification techniques as prescribed by the Federal Aviation Administration and the European Union Aviation Safety Agency. The U.S. Nuclear Regulatory Commission requires that nuclear power plant software developers use independent verification organizations to ensure that safety-critical software remains safe.

A2.4. Continuous Delivery.

A2.4.1. As this attachment outlines, a system program office cannot use continuous delivery methodologies, such as DevOps and DevSecOps, to deliver nuclear critical software for operational use. Continuous delivery to operational commands would circumvent the independent Verification Organization and the oversight of USAF nuclear certification process owners and would violate not only USAF requirements but also the intent of DoDD 3150.02 policy.

A2.4.2. Continuous delivery methodologies on non-nuclear critical software may be acceptable. If the system program office designs the nuclear weapon system to meet the responsibilities in the following section, then the system program office can safely implement continuous delivery methodologies such as DevOps and DevSecOps on the non-nuclear critical software.

A2.5. Program Responsibilities.

A2.5.1. Nuclear Critical Software and Critical Signals. **Chapter 3** of this manual describes Critical Signals and the importance of Critical Signals to the safety and surety of the nuclear weapon system. The system program office cannot apply continuous delivery methodologies to software that requires an NSCCA or an NSATE, but the system program office may apply continuous delivery methodologies to non-nuclear critical software. The system program office should design the weapon system within the following framework:

A2.5.1.1. Protect software that can impact Critical Signals by physical partitioning between processors or by the use of spatial partitioning provided by an operating system. Refer to **paragraph 3.3.3**.

A2.5.1.2. Ensure software that can impact Critical Signals does not receive data messages or command messages from continuously-deployed non-nuclear critical software.

A2.5.1.3. Ensure that the nuclear critical software provides status information to non-nuclear critical software automatically through a push mechanism, not by request from non-nuclear critical software through a pull mechanism.

A2.5.1.4. Ensure that status information from the nuclear critical software to the non-nuclear critical software is a one-way transfer over a one-way bus or other physical media. Moderated buses are not one-way, and changes to the outside software can potentially cause the interface to change significantly. These changes can introduce inadvertent or malicious consequences into the nuclear critical software.

A2.5.1.5. Ensure the weapon system encrypts Critical Signals that flow outside of immediate control of the nuclear critical software, and the software outside of the nuclear critical software does not have the ability to decrypt the Critical Signals. Refer to **paragraph 8.13.3**.

A2.5.1.6. Follow the Nuclear Safety Design Certification process for certification, starting with the Nuclear Certification Impact Statement, for the first certified software release and delivery.

A2.5.1.7. Follow the Nuclear Safety Design Certification process for certification if the system program office makes any changes to the nuclear critical software. Refer to **paragraph 5.3**, and all subparagraphs.

A2.5.2. Non-Nuclear Critical Software. If the system program office implements a continuous delivery methodology, the system program office shall ensure that the independent Verification Organization reviews the software design and system to ensure that the continuously delivered non-nuclear critical software cannot do anything to affect the nuclear critical software. **(T-1)** Analysis of the software and system design should prove that the design protects the nuclear critical software through the following guidelines:

A2.5.2.1. The nuclear critical software does not process input from software that uses a continuous delivery methodology.

A2.5.2.2. All data transfers between continuously delivered non-nuclear critical software and the nuclear critical software are one-way transfers from the nuclear critical software. This guideline especially applies to status data.

A2.5.2.3. Software outside the nuclear critical software does not have the ability to decrypt the Critical Signals. The system program office should consider encryption and decryption technology used on Critical Signals as part of the Critical Signal path.

A2.5.3. Release Process.

A2.5.3.1. The system program office should define a release process that protects the nuclear critical software and receives AFSEC/SEWN approval.

A2.5.3.2. If the system program office implements a continuous delivery methodology, the independent Verification Organization shall review the release process to ensure that the process is safe and meets the requirements of this manual. **(T-1)** Analysis of the continuous delivery release process should prove that the release process does not change the nuclear critical software, the release process cannot introduce unknown or malicious code into shared memory with nuclear critical software, and the release process meets requirements of USAF policy and this manual.

A2.5.4. Nuclear Safety Regression Testing. As defined in **paragraph 4.7**, the system program office should perform Nuclear Safety Regression Testing after changes to the non-nuclear critical portions of software. Therefore, for continuous delivery, the system program office should perform Nuclear Safety Regression Testing as part of the release and delivery process. This is, in fact, part of the standard release and delivery process defined by DevOps and similar continuous delivery methodologies. The following section defines the Nuclear Safety Regression Testing requirements:

A2.5.4.1. If the system program office implements a continuous delivery methodology, the system program office shall focus Nuclear Safety Regression Testing on the interfaces and shared hardware between the nuclear critical software and non-nuclear critical software. **(T-1)** Analysis of the Nuclear Safety Regression Testing process should prove that the testing identifies all potential interfaces and shared hardware between the nuclear critical software and non-nuclear critical software, and explicitly tests those interfaces and hardware to identify potential effects on Critical Signals. "Shared hardware" refers to hardware other than processors through which Critical Signals and non-critical data flow, such as network controllers.

A2.5.4.2. If the system program office implements a continuous delivery methodology, the system program office shall automate the Nuclear Safety Regression Testing to execute on every release as part of the release and delivery process. **(T-1)** Analysis of the release and delivery process should prove that the process automatically executes Nuclear Safety Regression Testing on each release. The system program office should not reduce Nuclear Safety Regression Testing in order to meet release timelines, but use additional resources to reduce the time needed to execute Nuclear Safety Regression Testing.

A2.5.4.3. If the system program office implements a continuous delivery methodology, the independent Verification Organization shall ensure the Nuclear Safety Regression Testing process meets these restrictions and the requirements of this manual. **(T-1)** Analysis of the Nuclear Safety Regression Testing process should prove that the testing meets all requirements in this manual. The Verification Organization does not need to review every change to the Nuclear Safety Regression Testing process, only the overall process that the software developer initially defined for each nuclear critical software release.

A2.6. AFSEC/SEWN Certification Actions.

A2.6.1. If AFSEC/SEWN and the independent Verification Organization concur that the system program office has designed the weapon system to the framework that **paragraph A2.5** and subparagraphs defines, then AFSEC/SEWN should allow the system program office to perform regular updates to the non-nuclear critical software without requiring Nuclear Safety Design Certification for each update.

A2.6.2. AFSEC/SEWN reserves the right to require additional verification such as NSCCA or NSATE for the nuclear critical software even without change to the nuclear critical software; this additional testing would be based on the accumulation of changes to the non-nuclear critical software over time.