AMVS: Automated Mapping of Vulnerabilities to Security Standards in Military Software

Sungjae Choi
Division of Computer Convergence
Chungnam National University
Daejeon, Republic of Korea
Email : ntchoi@o.cnu.ac.kr

Dongkyun Yang
Division of Computer Convergence
Chungnam National University
Daejeon, Republic of Korea
Email : h2861331@o.cnu.ac.kr

Sojin Na
Division of Computer Convergence
Chungnam National University
Daejeon, Republic of Korea
Email : sjna@o.cnu.ac.kr

Jinsoo Jang
Division of Computer Convergence
Chungnam National University
Daejeon, Republic of Korea
Email : jisjang@o.cnu.ac.kr

***Abstract*— As software plays a critical role in the performance and security of modern weapon systems, integrating security engineering throughout the software development life cycle is becoming increasingly essential. This paper introduces AMVS, an automated framework that systematically maps responses to software vulnerabilities to international security standards. AMVS leverages ISO/IEC 25023 for software quality, the Risk Management Framework (RMF) based on NIST SP 800-53 Rev. 5, and real-time CVE/CWE data to establish traceable and quantifiable security requirements from the design stage. The framework collects CVE data through web crawling, classifies them by CWE, and assigns severity-based weights to each vulnerability. It then automatically maps optimal security measures to corresponding ISO and RMF control items. These mappings are applied directly to system design documents, enabling developers to prioritize and implement security controls with minimal manual effort. A case study on the PX4 flight control platform demonstrates the effectiveness and scalability of the proposed framework. AMVS enhances the development of resilient and standards-compliant military software systems capable of proactively responding to emerging cyber threats.**

Keywords*— *Cyber-attacks, SDLC, weapon system, OSS, ISO/IEC 25023, RMF, Security measures, CVE, CWE, Drone

I. Introduction

As modern weapon systems rapidly evolve, embedded software has become essential for ensuring reliability, responsiveness, and mission success. Autonomous and highly connected platforms—such as military drones used for reconnaissance, surveillance, target acquisition, and precision strikes—are particularly vulnerable to cyberattacks that exploit software weaknesses. Real-world incidents, including the spoofing attack on the U.S. Q-drone, have demonstrated that software vulnerabilities can pose serious threats to national security [1,2].

Addressing such challenges requires more than reactive patching; instead, a systematic and standards-based security methodology must be applied throughout the software lifecycle—including design, development, deployment, and maintenance. This approach entails defining security requirements during the design phase and ensuring their traceability through implementation and verification.

To support this goal, this study leverages three core elements: (1) ISO/IEC 25023, an international standard for measuring software quality; (2) the U.S. Department of Defense's RMF; and (3) structured vulnerability data from public sources such as CVE and CWE. ISO/IEC 25023 provides quantifiable quality metrics, RMF offers lifecycle-integrated security controls for defense systems, and CVE/CWE enable proactive vulnerability tracking and classification.

However, ISO/IEC 25023 does not specify detailed security control mechanisms for mission-critical systems, and RMF—though comprehensive—is complex and only partially disclosed. While CVE and CWE standardize vulnerability disclosure, they do not directly map to mitigation controls or design requirements. Although integrating these standards and datasets enables the systematic definition and traceability of security requirements from the design stage, the manual mapping process is time-consuming, error-prone, and inconsistent, especially as the volume of known vulnerabilities continues to grow.

To address this issue, we propose an automated framework that collects the latest CVE data, classifies them into CWE categories, and assigns weighted scores based on severity and likelihood. Through this approach, developers can automatically associate prioritized ISO and RMF-based security controls with system design components. This enables the application of validated countermeasures for critical vulnerabilities while reducing the risk of omissions.

This study validates the proposed framework using PX4, a widely adopted open-source flight control system for military drones. Through expert comparison and evaluations spanning the firmware, operating system, and application layers, we demonstrate the framework's scalability, objectivity, and practical effectiveness. By continuously updating its knowledge base and aligning with international standards, the framework contributes to the development of resilient and trustworthy software systems in defense environments.

The remainder of this paper is structured as follows: Section II reviews international standards related to software quality and security for weapon systems, including ISO/IEC 25023 and RMF. Section III presents the design and methodology of the proposed framework that integrates ISO and RMF controls. Section IV illustrates its applicability through a case study on PX4. Finally, Section V concludes the paper and outlines directions for future work.

II. Related work and Background

In general, verifying the quality of software in weapon systems remains limited for several reasons. First, when a weapon system is being developed for the first time, it is often

difficult to define user requirements clearly. Even when such requirements are defined, there is a lack of internationally recognized standards for measuring the quality of weapon system software. Furthermore, discrepancies between development phases can lead to inconsistencies where the deliverables fail to fully reflect the defined requirements.

To address this, it is necessary to adopt a systematic approach to improving software quality from the early stages of weapon system development. The international standard ISO/IEC 25023, which provides quantitative metrics for software quality, offers a promising solution [3]. However, when applied in isolation, ISO/IEC 25023 has limitations in several key areas, including security threat management, dynamic threat response, supply chain security, security integration across development phases, and secure maintainability [4 -7].

Beyond software quality management, one of the most critical aspects of operating weapon systems is ensuring the integrity and trustworthiness of data processed within the system. In response to this need, the United States established the RMF in 2013 as a security evaluation standard [8]. The RMF includes control families—comprehensive sets of safeguards designed to satisfy security requirements. These controls are accompanied by supplemental guidance and control enhancements. Based on NIST SP 800-53 Rev. 5, the RMF consists of 20 control families and 1,013 individual control numbers [9]. By clearly defining security requirements, the RMF allows developers to incorporate security considerations from the earliest stages of system design and ensures consistent application of security controls across various components of complex weapon systems, thereby enhancing overall system security [10].

Previous studies on RMF have examined its application in weapon system software development and cybersecurity enhancement [11–13], and further research has explored improvements in RMF implementation procedures and evaluation methods [14,15]. However, when applying the RMF to weapon systems development outside the United States, several limitations arise. Although RMF is a publicly available framework based on NIST documentation, many of its components—such as details on classified operational environments, military encryption technologies, and specific threat models for weapon systems—are not publicly disclosed. This restricts its direct application as an international standard. Therefore, rather than fully adopting the RMF framework, selectively applying its relevant components to reinforce cybersecurity may be a more practical approach.

Systematically addressing software vulnerabilities is essential to improving the reliability of weapon system software. While databases like CVE and CWE are valuable for identifying and categorizing known vulnerabilities, they do not provide explicit guidance on mitigation strategies or design-level solutions. As a result, developers are burdened with the responsibility of analyzing, fixing, and developing prevention strategies on their own. This manual and subjective process makes it difficult to objectively assess whether the resulting software meets required security quality standards, thereby increasing the likelihood of degraded quality and the emergence of additional vulnerabilities.

To overcome these challenges, this study proposes the following: (1) automated vulnerability management, (2) automatic mapping of security countermeasures using ISO/IEC 25023 and RMF standards, and (3) support for efficient documentation throughout the software development process. Collectively, these contributions address the limitations of existing practices in this domain.

III. AMVS Design

The overall process of the AMVS framework is depicted in Fig 1. When a software vulnerability is identified, the proposed system collects the latest CVE and CWE data, analyzes and classifies them by considering severity and weighted factors, and then utilizes a custom-developed LLM-based engine to automatically map optimal mitigation strategies. This mapping is performed with reference to NIST and ISO security standards to ensure the most effective countermeasures. As a result, the framework provides developers with optimal vulnerability responses and automatically generates design documents, such as the SRS and SDD, that incorporate these requirements.

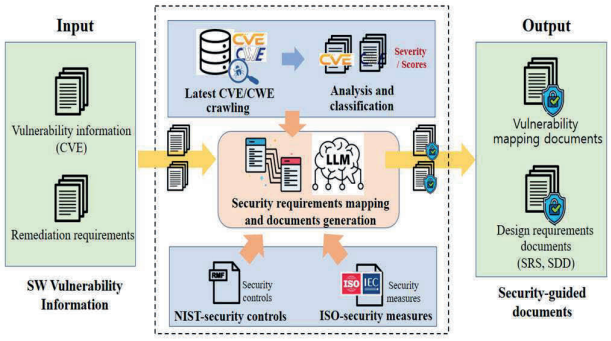

Fig. 1. AMVS framework

A. Vulnerability Collection and Classification

The **AMVS** framework aims to reduce software vulnerabilities by providing developers with mitigation strategies based on the latest CVE data. It automatically collects and organizes vulnerability information from CVE records in the NVD.

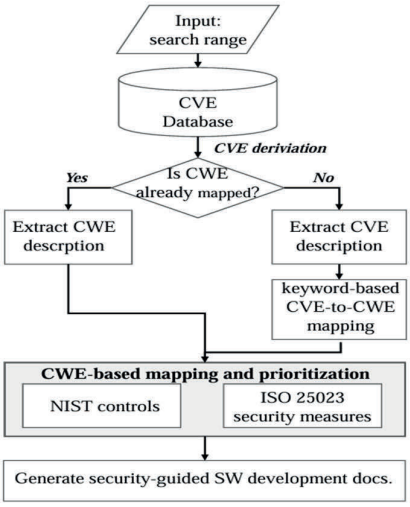

Fig. 2. CVEs extraction and their mapping to security measures and controls.

The core architecture of the framework is illustrated in Fig 2. First, the latest CVE data is obtained using the CVE-fixes database from the Secure IT project [16], which provides

normalized NVD data in SQLite format. Each entry includes the CVE ID, publication date, CVSS score, vulnerability description, and associated CWE identifier. Based on user-defined criteria, CVEs are filtered and checked for the presence of CWE identifiers. If available, the CWE data is linked to the corresponding CVE record.

The mapping process is performed in two stages. First, CWE entries are mapped to ISO/IEC 25023 security characteristics using a keyword-based approach. Predefined keywords are associated with each ISO security attribute, and their occurrence in CWE descriptions determines the mapping. Second, CWE entries are linked to security controls from the NIST SP 800-53 using TF-IDF vectorization. This technique emphasizes domain-specific security terminology while minimizing noise from general vocabulary. After removing English stop words and assigning weights to key terms, cosine similarity is calculated to identify the most relevant NIST control items.

Some CVE entries in the NVD either lack CWE information or are labeled as unknown. In such cases, the system infers ISO and NIST mappings directly from the vulnerability description. To support this, the framework is designed for extensibility. Specifically, a pre-trained LLM is used to process the CVE description field as a prompt and extract relevant security characteristics and control items. This approach enables flexible interpretation of unstructured vulnerability data that is otherwise difficult to classify, and it is expected to improve the coverage and accuracy of the framework in future enhancements.

During the reporting phase, CVE data filtered by a user-specified date range can be exported in CSV or PDF format. Reports include the CVE ID, publication date, CVSS score, CWE, vulnerability description, ISO/IEC 25023 characteristics, and NIST SP 800-53 control items. Users may also apply filtering options, such as excluding unmapped entries.

B. Mapping between CWE and both security measures and controls

To mitigate vulnerabilities effectively, it's essential to analyze CWEs. MITRE's scoring formula for CWEs uses both the frequency of occurrence in NVD and the average CVSS score, normalized as follows:

$\text{Score}(\text{CWE}_X) = \text{Fr}(\text{CWE}_X) \times \text{Sv}(\text{CWE}_X) \times 100$

where:

$$\text{Fr}(\text{CWEX}) = \frac{\text{count}(\text{CWEX} \in \text{NVD}) - \min(\text{Freq})}{\max(\text{Freq}) - \min(\text{Freq})}$$

$$\text{Sv}(\text{CWEX}) = \frac{\text{average}_{\text{CVSS(CWEX)}} - \min(\text{CVSS})}{\max(\text{CVSS}) - \min(\text{CVSS})}$$

CVSS values (0.0–10.0) are classified into severity tiers as shown in Table 1:

TABLE I. CVSS SEVERITY CLASSIFICATION

Severity	CVSS Score Range	Description
High	7.0–10.0	Severe impact vulnerabilities
Medium	4.0–6.9	Moderate impact vulnerabilities
Low	0.0–3.9	Limited impact vulnerabilities

Next, CWEs are mapped to mitigation strategies based on relevance, categorized as direct, partial, or indirect. This determines risk level, implementation priority, and weight, as summarized in Table 2.

TABLE II. MAPPING RELEVANCE AND WEIGHTING

Relevance Type	Risk Level	Implementation Strategy	Weight
Direct	Critical	Core Requirement	3
Partial	High	Enhancement	2
Indirect	Medium	Optional Support	1

AMVS assesses and maps the relationship between CWE and their countermeasures using LLM. This approach allows for automated analysis by enabling the LLM to analyze large datasets and learn patterns. Due to the nature of its training data, the model can identify relationships between entities. Even as new vulnerability data emerges, continuous learning enables it to keep the relationships between CWEs and corresponding countermeasures up to date with evolving security environments. Furthermore, analyzing the relationships between CWEs and countermeasures using LLMs and assigning weights provides more consistent and objective results compared to manual mapping, which may vary depending on who performs it.

In the current software security landscape, a reactive approach predominates wherein developers must identify and implement mitigations only after vulnerabilities have been discovered and classified as CVEs and CWEs. This process burdens developers with individually researching and implementing countermeasures for each vulnerability. Such an approach not only leads to inefficient utilization of time and resources but may also compromise the consistency and systematization of security responses.

This study analyzed 269,734 CVE vulnerability data points as of 2025 and performed automatic mapping to NIST control lists and ISO/IEC 25023 security measures based on the CWE Top 25 published in 2024 [17]. The results for the top five items from the Top 25 are systematically organized in Tables 3 and 4.

TABLE III. MAPPING RESULTS ISO SECURITY MEASURES (TOP 5)

Rank	ID	Score	Security Measures	Weight	Final Score
1	CWE-79	56.92	Confidentiality (SCo-1-G)	3	1.71
2	CWE-787	45.2	Integrity (SIn-1-G)	3	1.36
3	CWE-89	35.88	Integrity (SIn-1-G)	3	1.08
4	CWE-352	19.57	Authenticity (SAu-1-G)	2	0.39
5	CWE-22	12.74	Confidentiality (SCo-3-S)	2	0.25

TABLE IV. MAPPING RESULTS NIST CONTROL ITEMS (TOP 5)

Rank	ID	Score	Representative NIST Control	Weight	Final Score
1	CWE-79	56.92	SI-11 (Input Validation)	3	1.71
2	CWE-787	45.2	SI-16 (Memory Protection)	3	1.36
3	CWE-89	35.88	SI-10 (Input Verification)	3	1.08
4	CWE-352	19.57	SC-17 (PKI)	2	0.39
5	CWE-22	12.74	AC-3 (Access Enforcement	3	0.38

The final score, considering weights, is calculated as follows:

$$\text{Final Score} = (\text{Score} \times \text{Weight}) \div 100$$

Finally, the execution priorities were determined based on the final score and the execution deadlines were presented as follows Table 5.

TABLE V. EXAMPLE OF EXECUTION PRIORITIES AND CORRESPONDING COUNTERMEASURES BASED ON FINAL SCORES.

Final Score	Interval	Countermeasures
> 1.0	Critical	Immediate Implementation
0.2–1.0	High	Within 6 Months
0.1–0.19	Medium	Within 12 Month
< 0.1	Low	Periodic Review

This automatic mapping system provides risk-based priorities through quantitative weighting that reflects the frequency and severity of vulnerability occurrence, and supports developers to respond quickly and effectively by standardizing security requirements. As a result, it contributes to systematizing the organization's security management and maximizing efficiency.

C. Enhancing SRS and SDD with Security Guidance

The SRS is a deliverable created at the beginning of the software development process, where software requirements are analyzed and defined—including user requirements related to specific security features and attributes of the system to mitigate security threats [18]. As outlined in IEEE Std 830, a typical SRS includes sections such as the overall product description, system function requirements, and other requirements. The system function requirements cover functional requirements, performance requirements, external interface requirements, and design constraints.

When drafting the system function requirements in the SRS, AMVS can assist by providing countermeasures that address weighted security vulnerabilities, as shown in Table 6. By presenting prioritized security requirements based on the sub-characteristics of security from ISO/IEC 25023 and relevant NIST security control items, the specification can be made more comprehensive and actionable.

This approach offers several advantages. Clearly defined requirements help reduce implementation errors and enhance overall security. By detailing requirements based on system-generated content, ambiguity between developers and stakeholders can be minimized, thereby preventing misinterpretations during development. Furthermore, incorporating security considerations from the design phase helps reduce the cost and complexity of addressing vulnerabilities later in the lifecycle. Specifically, including explicit security controls—such as input validation, memory protection, and access control—at the SRS drafting stage ensures early integration of security, reducing the likelihood of vulnerabilities and offering long-term cost-saving benefits.

TABLE VI. EXAMPLE OF SRS WITH AMVS-MAPPED SECURITY GUIDELINES

1. Functional Security Requirements			
Design requirement		***AMVS –mapped security guideline***	
ID	***Requirement***	***Mapping***	***Priority (Score)***
1.1. Input Validation			
SEC-FR-01	All user inputs shall be filtered for HTML/JS using OWASP ESAPI	SI-11, SIn-1G	Critical (1.71)
SEC-FR-02	Whitelist-based input filtering shall be implemented to prevent OS command injection	SI-10, SIn-2G	High (0.34)
1.2. Memory Protection			
SEC-FR-03	Address Space Layout Randomization (ASLR) shall be enabled to prevent buffer overflows	SI-16, SIn-1G	Critical (1.36)
1.3. Access Control			
SEC-FR-04	Account lockout functionality shall be implemented after authentication failures	IA-5	Medium (0.10)

The SDD, as defined in IEEE Std. 1016 [19], is a critical document in the SDLC that translates the security requirements specified in the SRS into concrete system design elements. By applying AMVS during the design phase—which outlines the software's structure, components, and interactions—security issues related to data flow and communication paths can be more clearly identified and addressed.

Incorporating security requirements systematically at the design stage enables structured evaluation of potential vulnerabilities during implementation and verification. The SDD reflects the weighted priorities provided by the mapping system for the content defined in the SRS, and presents detailed measures in Table 7 to guide the development of software that mitigates vulnerabilities according to their criticality.

TABLE VII. EXAMPLE OF SDD WITH AMVS-MAPPED SECURITY GUIDELINES.

2. Security Design Viewpoints			
Design requirement		***AMVS –mapped security guideline***	
Design Measure	Implementation	Mapping	Priority (Score)
2.1. Input Validation			
All user inputs shall be filtered for HTML tags and JavaScript using OWASP ESAPI	Server-side validation Module (input Sanitizer) implementation	SI-11, SIn-1G	Critical (1.71)
2.2.MemoryProtection			
Enable Address Space Layout Randomization (ASLR) and Stack Canary	Compiler option:-fstack-protector	SI-16, SIn-1G	Critical (Score: 1.36)
2.3.AccessControl			
Implement Role-Based Access Control (RBAC) model	Spring Security @PreAuthorize annotation	AC-6, SAc-1G	High (Score: 0.30)
Support Multi-Factor Authentication (MFA) and OAuth2.0	Keycloak integration	IA-5, SAu-1G	Medium (Score: 0.18)

IV. EVALUATION

In this study, to compare the expert-based approach with the proposed AMVS, three security experts were selected, each with over ten years of practical experience in the fields of cybersecurity and the defense industry. These experts possess diverse hands-on experience in areas such as defense software security verification and cyber threat analysis. The open-source PX4, widely used by drone manufacturers and research institutions around the world, was selected as the evaluation target. A manual vulnerability analysis was conducted based on CVE and CWE data, followed by security

countermeasure mapping to ISO and NIST standards. Each expert was given the same set of four CVE entries, and evaluations were independently conducted in terms of analysis time and accuracy of the proposed mitigation strategies. (This study was conducted with the voluntary participation of the experts. No personally identifiable information was collected, and since it does not involve human subjects research, IRB approval was not required.)

The process of vulnerability analysis and countermeasure derivation was carried out using two approaches—an expert-based method and the automated framework method—as illustrated in Fig. 3.

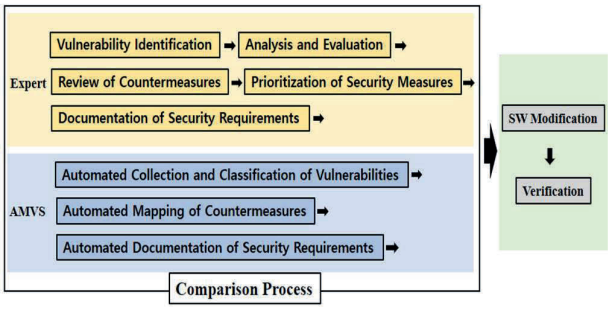

Fig. 3. Vulnerability Response Procedure.

The individual mapping time and accuracy of each of the three experts for the appropriate ISO and NIST items are shown in Figure 4.

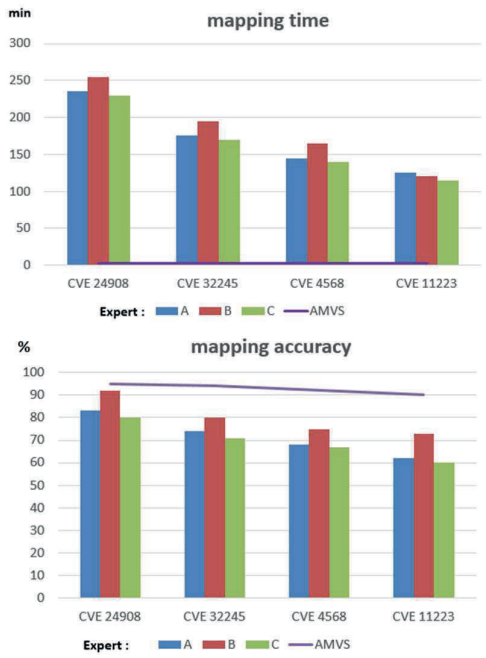

Fig. 4. Comparison of Mapping Time and Accuracy Between Three Experts and AVMS.

Table 8 presents the average values from the experts and the corresponding results obtained using the AMVS. On average, the experts required approximately 172 minutes per vulnerability for the mapping analysis, with a mitigation accuracy of around 73.7%. In contrast, AMVS completed the mapping in less than 3 minutes on average and achieved an accuracy of approximately 92%, demonstrating its efficiency and effectiveness.

TABLE VIII. MAPPING RESULTS OF VULNERABILITY MITIGATION MEASURES FOR PX4.

CVE ID	CWE ID	Expert		AMVS	
		mapping Time(min)	*Mapping Accuracy (%)*	*mapping Time(min)*	*Mapping Accuracy (%)*
CVE-2020-24908	CWE-306	240	85	3	95
CVE-2022-32245	CWE-120	180	75	3	94
CVE-2023-45678	CWE-285	150	70	3	92
CVE-2024-11223	CWE-798	120	65	3	90
Average		172.5	73.75	3	92.75

The automated mapping result of PX4 vulnerabilities to ISO/IEC 25023 and NIST control items using the AMVS system is presented in Table 9. The calculated risk score of 0.294, derived from severity and priority assessments, classifies this as a high-risk issue. To mitigate this vulnerability, AMVS recommends implementing the following specific security controls:

• IA-2(Identification and Authentication)– Enforce robust authentication mechanisms.

• SAu-1-G– Verify operational authenticity

TABLE IX. MAPPING OF PX4 CVES TO ISO/IEC 25023 AND NIST SP 800-53.

CVE ID	CWE ID	Score	ISO/IEC 25023 Measure	NIST SP 800-53 Control	Priority	Final Score
CVE-2020-24908	CWE-306	9.8	SAu-1-G	IA-2	3	0.294
CVE-2022-32245	CWE-120	8.1	SIn-1-G	SI-16	3	0.243
CVE-2023-45678	CWE-285	7.2	SAc-1-G	AC-6	2	0.144
CVE-2024-11223	CWE-798	6.5	SCo-2-G	IA-5	2	0.13

Regarding the major security vulnerabilities identified in PX4, the proposed automated mapping framework was able to automatically generate mappings within the SDLC development phases—specifically to the SRS and SDD stages—as shown in Tables 10 and 11. This approach helps reduce implementation errors and mitigates ambiguity in requirements during the development process, thereby enhancing overall security.

TABLE X. PX4 SRS WITH AMVS-MAPPED SECURITY GUIDELINES.

1. Functional Security Requirements			
ID	***Requirement Description***	***AMVS-Mapped Security Guideline***	
		Mapping	***Priority***
SEC-FR-01	All MAVLink packets must verify length (msg->len) and checksum (CRC32) upon receipt.	SI-10, SIn-1-G	Critical
SEC-FR-02	Check for integer overflow when parsing GPS data (if altitude > INT32_MAX).	SI-16, SIn-2-G	High
SEC-FR-03	Verify Ed25519 digital signature during firmware upload (px4_firmware_verify).	IA-2, SAu-1-G	Critical

SEC-FR-04	Check role-based permissions when accessing uORB topics (px4_orb_access_check()).	AC-6, SAc-1-G	High
SEC-FR-05	SD card encryption keys must not be hardcoded; load from environment variables at runtime.	IA-5, SCo-2-G	Critical

Similarly, Table 11 demonstrates how SDD requirements explicitly incorporate vulnerability mitigation strategies by AMVS. This design aims to strengthen authentication protocols, key management, and data protection measures.

TABLE XI. PX4 SDD WITH AMVS-MAPPED SECURITY GUIDELINES.

2.Detailed Security Measures			
Design Measure	***Implementation***	***AMVS*** ***-Mapped Security Guideline***	
		Mapping	***Priority***
2.1. Authentication Enhancement			
Firmware Signature Verification: Use Ed25519 digital signatures; manage signing keys via TPM 2.0 (Pixhawk 6X).	Halt boot process on invalid signature detection.	IA-2, SAu-1-G	Critical
HSM Integration: Secure key storage via TPM 2.0 (Pixhawk 6X).	Simulate and test key extraction attempts.	IA-5	High
2.2. Key Management			
SD Card Key Management: Dynamically load from environment variables.	Simulate key absence scenarios.	IA-5, SCo-2-G	High
Memory Protection: Apply XOR masking for sensitive data.	Perform memory dump analysis to verify key obfuscation.	SC-28	Medium
2.3. Data Protection			
SD Card Encryption: AES-256 encryption with keys loaded from environment variables.	Analyze memory dumps for key leakage.	IA-5, SCo-2-G	Medium
MAVLink Encryption: Integrate DTLS 1.3.	Perform Wireshark packet analysis.	SC-8, SCo-1-G	Medium

The key advantages of AMVS for PX4 security hardening include the ability to define security requirements systematically and standards-based during the design process, automated vulnerability-to-control mapping through prioritized remediation workflows, and optimized resource allocation through risk-weighted action plans. The framework proactively addresses vulnerabilities while maintaining compliance with ISO/IEC 25023 and RMF standards.

V. CONCLUSIONS

This paper proposed the AMVS framework to enhance the cybersecurity of weapon systems by automatically mapping software vulnerabilities to appropriate mitigation strategies. AMVS supports the software development process by providing context-aware security measures and automated recommendations for functionally similar components, thereby enabling proactive risk elimination.

The framework introduces a novel integration of ISO/IEC 25023, an international standard for software quality measurement, and the RMF from the U.S. Department of Defense. By continuously updating vulnerability data and assigning response priorities based on risk levels, AMVS significantly improves software security posture.

The proposed approach offers reliable and standardized responses to common security challenges. It minimizes subjective bias by automatically identifying and prioritizing high-risk and frequently occurring vulnerabilities, ultimately reducing development time and cost. These capabilities support the development of safe and trustworthy software by delivering effective and repeatable mitigation strategies.

By applying this framework to practical development artifacts such as the SRS and SDD in the early stages of weapon system development, accurate security requirements can be defined, known vulnerabilities can be addressed, and proactive protection can be assured. This contributes directly to the realization of robust software security in operational environments.

Future work will focus on developing AI-based tools that automatically assess vulnerabilities in development documents and suggest corresponding mitigations. This includes automating software quality evaluation processes and integrating AI-based solutions for issue resolution. Furthermore, by aligning with international standards such as NATO STANAG, this research aims to contribute to the global standardization of software quality metrics in defense systems.

REFERENCES

[1] J. Smith et al., “Military applications of UAVs: A comprehensive review,” Defense Technology, vol. 14, no. 1, 2020.

[2] B. Williams, “Analysis of the RQ-170 spoofing attack: Lessons learned,” Cyber Defense Review, vol. 8, no. 3, 2019.

[3] ISO/IEC 25023, “Systems and Software Engineering—Measurement of System and Software Product Quality,” ISO, 2016.

[4] M.HowardandS.Lipner, The Security Development Lifecycle. Microsoft Press, 2006.

[5] Y. Chen and J. Kim, “Dynamic Risk Analysis in Cybersecurity,” IEEE Security & Privacy, 2020.

[6] Microsoft SDL Practices Documentation, 2022.

[7] P. Sullivan et al., “Challenges in Supply Chain Security for Modern Software Systems,” ACM Transactions, 2019.

[8] NIST, Risk Management Framework for Information Systems and Organizations, NIST SP 800-37 Rev. 2, Dec. 2018.

[9] NIST, Security & Privacy Controls for Information Systems and Organizations, NIST SP 800-53 Rev. 5, Sept. 2020.

[10] G. Stoneburner, A. Gojuen, and A. Feringa, Risk Management Guide for Information Technology Systems (NIST SP 800-30), 2002.

[11] A. Johnson et al., “Implementation of RMF in Federal IT Systems: Case Studies and Lessons Learned,” Government IT Review, vol. 734 15, no. 2, 2020.

[12] L. Chen et al., “Adapting RMF for Critical Infrastructure Protection,” IEEE Transactions on Cybersecurity, vol. 8, no. 3, 2021.

[13] S. Park, “Case Study on RMF Application in Defense Systems,” Journal of Defense Cybersecurity, n.d.

[14] J. Lee et al., “Improving RMF Processes for Agile Development Environments,” Software Security Review, vol. 11, no. 4, 2019.

[15] M.Smith, “Streamlining RMF: Challenges and Opportunities,” Cyber Process Optimization Journal, n.d.

[16] G. Bhandari, A. Naseer, and L. Moonen, “CVEfixes: Automated Collection of Vulnerabilities and Their Fixes from Open-Source 743 Software,” in Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering, Athens, 744 Greece, Aug. 2021, pp. 30–39.

[17] CWE Top 25 Most Dangerous Software Weaknesses. [Online]. Available: https://cwe.mitre.org/top25/ (accessed May 2025).

[18] IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specification (IEEE Std 830-1998), 1998.

[19] IEEE Computer Society, IEEE Standard for Information Technology System Design SW Design Description (IEEE Std 1016-2009), 2009.