AJEMA GLOBAL RESEARCH NETWORK

AMERICAN Journal of Engineering, Mechanics and Architecture
Volume 4, Issue 08, 2026 ISSN (E): 2993-2637

AI-Augmented DevSecOps for Protecting U.S. Enterprise Software Supply Chains and Critical Digital Services

Fawad Mir
Master of Science in Information Technology, Washington University of Science and Technology (WUST), Alexandria, Virginia, USA

Jawad Yaqoob Mir
Master of Science in Information Technology, Washington University of Science and Technology (WUST), Alexandria, Virginia, USA
jawadmir488@gmail.com

Muhammad Saad khawar
Master of Science in Information Technology, Washington University of Science and Technology (WUST), Alexandria, Virginia, USA

Abstract: Modern enterprise applications rely on extensive third-party code, automated build systems, cloud-native infrastructure and rapidly changing vulnerability intelligence. Security controls are then spread out across the development, the software supply-chain assurance and production operations making it hard to relate some process deficiency with the subsequent impact seen in operations. This paper proposes an AI-driven DevSecOps approach focusing on governance with connections between Secure Software Development, Software Provenance, Runtime Observability, AI for analysis, Deterministic Policy Enforcement, and Responsible Human Decision Making. The framework is developed in a design-science methodology, and structured as the following layers: mission and regulatory context; AI-augmented DevSecOps pipeline; cloud-native runtime; unified observability and threat intelligence; AI intelligence and decision support; and policy governance with human oversight. It has a core principle that AI can correlate evidence, assess risk, and inform action but cannot override the requirements of policy, or authorize high impact operational decisions. A healthcare software-supply-chain scenario is the basis for an example of - a controlled rollback enabled by software bills of materials, provenance records, runtime telemetry, vulnerability intelligence, AI reasoning, policy checks, and human approval. It provides an end-to-end conceptual architecture, a decision model driven by policy, and a standards-based foundation for implementation. Since the evaluation is scenario based, the framework should be viewed as a design artifact and it is necessary to prototype and empirically validate it across multiple sectors.

Keywords: AI-augmented DevSecOps; SBOM; software provenance; cybersecurity governance; critical digital services.

Introduction

Enterprise software is no longer being built as a stand alone product. Applications are composed from open-source libraries and commercial parts, build services, container images, infrastructure templates and cloud platforms. These dependencies result in faster deliveries but also increase the number of organisations, tools and trust relationships which can impact a production service. The two examples of SolarWinds and Log4Shell demonstrated a compromise of a trusted software delivery process and a quick spread of risk via a popular component [1], [2]. In the case of organisations providing healthcare, financial, energy, telecommunications or government services, a defect or compromise upstream consequently become an operational-resilience issue, not an ordinary application-security issue.

U.S. guidance has strengthened various components of this cycle. Executive Order 14028 brought a much-needed focus from the federal government on addressing software development and software supply-chain transparency issues [3]. The NIST Secure Software Development Framework (SSDF) outlines activities for preparing organisations, protecting software, developing good software releases and dealing with software vulnerabilities [4]. The NIST Cybersecurity Framework (CSF) 2.0 also increases its focus on enterprise risk and governance [5], and NIST Special Publication (SP) 800-204D concentrates on incorporating software supply-chain security into continuous integration and continuous delivery (CI/CD) pipelines, a component of DevSecOps [6]. Supply-chain Levels for Software Artifacts (SLSA) specification, software bills of materials (SBOMs), provenance attestations, and digital signatures are also additional factors that enhance the visibility of components and build integrity [7]-[10]. What is left is not that they lack individual controls, but that there is a lack of connectivity between controls. The development teams might have an awareness of the libraries that were included in an artefact but the operations teams may be dealing with strange behaviour without that information. Instead, runtime signals could indicate potentially unsafe deployment behavior, and yet not affect subsequent pipeline gates. Adoption study of DevSecOps shows common problems such as tool integration, organisational silo, management of data consistency and automation and accountable decisions [11]. The resulting fragmentation slows exposure analysis, weakens root-cause attribution, and makes it possible for redundant or conflicting security patches. AI systems can help by correlating information on sources, builds, deployment, vulnerabilities, and telemetry – a task that would be hard for analysts to perform by hand. It can be used for anomaly detection, prioritization of vulnerabilities, risk of deployment and hypothesis generation. But probabilistic recommendations do not equal policy authority. Miscalibration (not working correctly), out-of-bounds inputs, unfamiliar environments, out of calibration, or lack of explainability can cause detrimental actions to be carried out, especially if the recommendation is for a mission critical service. This is addressed in the NIST AI Risk Management Framework (AI RMF) and explainable-AI research that prioritizes governance, traceability, validity, and context relevant oversight [12], [13].

This paper asks the following question: How can pre-deployment software assurance be intertwined with the post-deployment operational intelligence without delegating high impact decision making to AI? The design is guided by three subsidiary questions; (1) what evidence needs to travel across the software lifecycle; (2) how is AI recommendations to be integrated with deterministic policy and human authorisation; and (3) how do you operationalise existing U.S. cybersecurity guidance into a single architecture? The paper makes three contributions. Now, it also introduces a six-layered framework that links the mission context, secure software development, supply-chain evidence, cloud native run-time telemetry, AI decision-support and governance. Second, it is a policy-governed decision model based on mandatory rules that are to be adhered to before AI suggestions are accepted and for high impact decisions to be implemented, authorisation is required from a human. Third, it illustrates the framework in the context of a healthcare software-supply-chain incident and obtains a tangible evaluation agenda. The framework is conceptual – it does not claim experimentally measured detection or recovery

improvements for detection or recovery – but it provides for an organization of responsibilities and information flows.

Background and Research Gap

DevSecOps and software-supply-chain assurance

DevSecOps is an extension of DevOps with security flowing through planning through to coding, building, testing, release, deployment, and operation. Examples of pipeline controls are threat modelling, static and dynamic analysis, software composition analysis, secret detection, infrastructure-as-code checks, container scanning, signing and policy gates. Moving these activities back can minimise rework, but not at release; security isn't over when it's released. What is acceptable during build time may be uncovered after a vulnerability has been disclosed, and a verified artefact might have unforeseen behaviour under production conditions. Thus, effective lifecycle protection needs to come with a whole-of-lifecycle evidence trail from the source and build systems through to the deployment and lifecycle monitoring [4], [6], [11].

Software-supply-chain assurance is a source for this continuing validity. SBOMs identify components and versions, provenance records describe where and how an artefact was created, signatures aid integrity/authenticity checks, and SLSA and in-toto are structured strategies to supply chain attestations and verifiable steps in the supply chain [7–10]. The NIST supply-chain risk guidance also notes that risks associated with cybersecurity do not lie just within the borders of the organisation but go beyond that, into their supply chain of services, and suppliers [14]. These provide for better transparency, but are of limited operational value if they cannot be linked quickly to running services, deployments histories and observed behaviour.

AI-assisted cybersecurity and governed automation

AI and machine learning technologies can be used to identify repeatable actions and relationships across a high volume of security data, prioritize, and summarize relationships between security events. Some of the useful functions would be prioritizing vulnerabilities based on exposure and criticality of the services, correlating anomalies to recent deployments, identifying likely root causes, estimating impacts of remediation solutions while operating the services, and providing explanations for analysts. Vulnerability prioritization must factor in complementary data: CVSS defines technical severity; EPSS predicts probability of exploitation in near future; and the CISA Known Exploited Vulnerabilities (KEV) Catalog shows evidences of exploitation [15–18]. Taking one of these signals as a standalone risk choice would neglect business context, software reachability, and operational impact.

The governance issue is also key. For highly delineated reversible actions that are pre authorised for action, autonomy can be appropriate but it is not a service model for critical services. When AI leads to consequential decision making, it needs to be explainable, capable of quantifying uncertainty, auditable, able to track model performance, and clearly separate the roles of humans and AI [12, 13]. Although Zero Trust principles resonate with ongoing verification, ongoing verification does not replace the need for explicit authority boundaries [19]. The architecture designed here hence de-couples evidence collection, AI reasoning, deterministic evaluation of policies, and human approval.

Research gap

Current solutions offer well developed but limited features. DevSecOps bolsters security during development, supply-chain frameworks strengthen the traceability of software origins and components, observability platforms explain software activity at runtime, AI systems offer analytical insights for support and cyber-resilience practices enable continuity and recovery. The missing part is the end-to-end control model, retaining the linkage between these capabilities and

making the authority boundary explicit. The proposed framework fills this gap with applying common evidence objects, bidirectional feedback mechanism, and decision hierarchy following a policy.

Table 1. Comparison of Existing Approaches and the Proposed Framework.

Approach	Primary strength	Typical limitation	Required integration
DevSecOps	Continuous security within delivery workflows	Findings may remain disconnected from runtime context	Persist build and release evidence into operations
Supply-chain assurance	Component transparency, integrity, and provenance	Often treated as a pre-deployment gate	Map SBOM and provenance data to running services
Runtime observability	Detailed operational health and behaviour	May lack software-origin and dependency context	Correlate telemetry with deployment and component evidence
AI-based cybersecurity	Scalable correlation and prioritisation	Probabilistic output, opacity, and model risk	Constrain AI to explainable, monitored decision support
Cyber-resilience practices	Continuity, containment, and recovery	Can be weakly linked to development controls	Return incident evidence to pipeline policy and design
Proposed framework	Lifecycle-wide evidence and governed decision support	Conceptual and not yet empirically validated	Integrates all capabilities with policy precedence and human authority

Research Methodology

The study follows a design-science research (DSR) methodology, as the primary research product is a design that addresses an identified organisational and technical problem in the form of an architectural artifact. DSR emphasizes problem relevance, purposeful design of artifacts, evaluation, and communication [20, 21]. In line with the principle that a conceptual artifact can be evaluated using ex ante and scenario-based methods prior to the deployment of a prototype and field testing [22], the evaluation strategy is developed following this logic. Figure 1 summarises the process used in this study

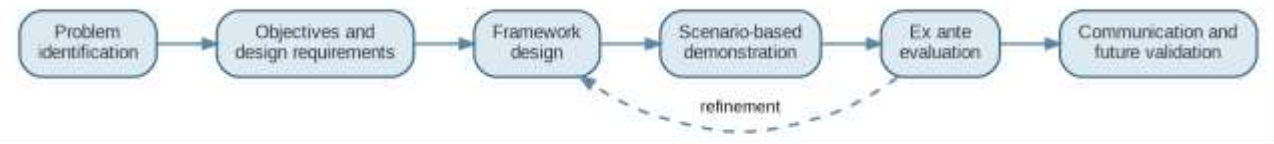

Figure 1. Design-science process used to construct and evaluate the framework.

The first activity is problem identification: development-time software assurance and runtime evidence, AI analysis, and operational authority are often disjointed. The second activity develops the design requirements based on U.S. standards and technical literature. The artifact needs to offer traceability from source to deployed services, tie together vulnerability intelligence with the operational perspective, maintain the precedence of deterministic policies, ensure accountable

human decision-making for high-impact activities, offer explanations and audit trails, and allow feedback from operations to the development process.

The third activity presents the six-layer framework and the established information flows. The fourth activity demonstrates the artifact via the use of a hypothetical healthcare scenario due to its mix of software-supply-chain exposure, high availability, and accountability standards. The fifth activity evaluates the scenario in terms of the design requirements, but not in terms of quantitative performance metrics. The analysis asks questions such as: Can the framework detect the impacted service? Can the framework pull together evidence in an adequate manner? Can the framework differentiate between an AI recommendation and an authoritative approach? Can the framework enable a targeted response? Can the framework enable a trackable learning process?

This study has two limitations. Firstly, the scenario does not measure the accuracy of the model, the mean time to detection, or the mean time to recovery. Secondly, the architecture is technology-neutral; the specific products and algorithms are implementation options and not part of the conceptual contribution. These limits make it impossible to offer illustrative details as empirical results.

Design principles and architecture

The framework relies on six principles: all protection till the end of the life cycle; persistent evidence of the software-supply-chain; AI is used only as a decision support tool and in no other role; action taken is deterministic; high impact actions are human-authorised; and compliance with accepted standards. These principles work together to separate analytical automation from governance authority. The six layers and the feedback path to validate the operation knowledge and place it back in the pipeline are shown in Figure 2.

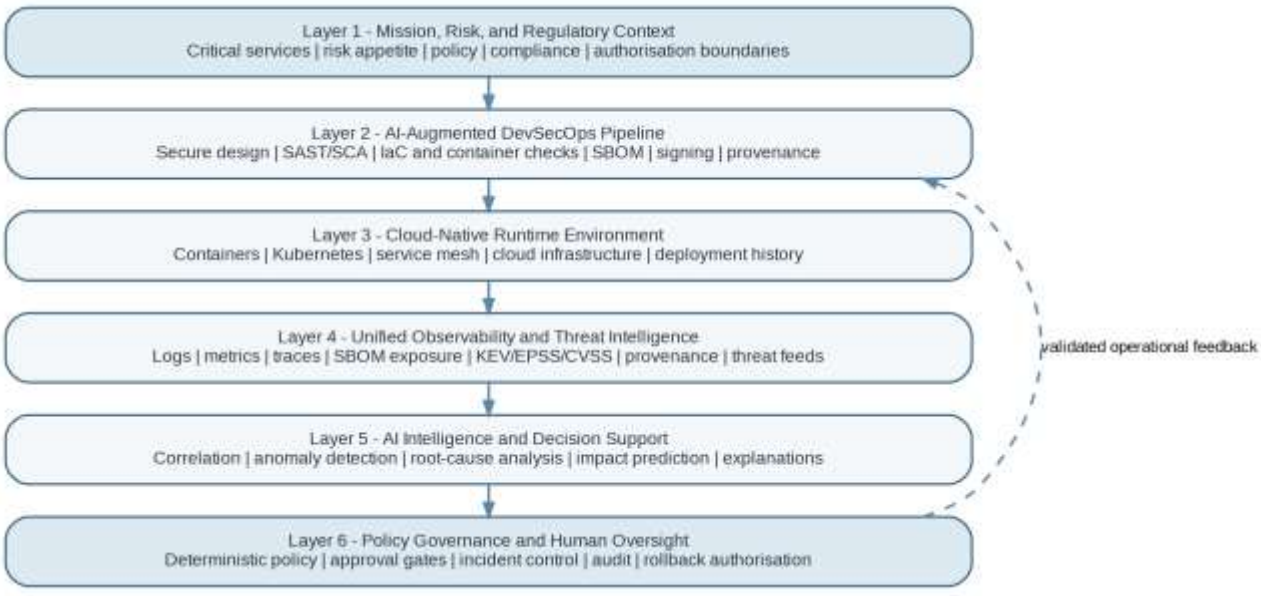

Figure 2. Governance-oriented AI-augmented DevSecOps architecture.

Layer responsibilities

Layer 1 – Mission Risk & Regulatory context. This layer identifies the services, data, and operations to be protected. The inputs include business objectives, service criticality, risk appetite, regulatory requirement, security policy, authorised perimeter etc. Outputs – Rules, control objects, escalation parameters, authorization rules etc. that have been made binding. All layers focus on aligning enterprise governance toward the NIST CSF 2.0 and provides a way to interpret technical findings based upon the impact to the mission in lieu of just the severity of the finding [5].

Layer 2: Augmented DevSecOps Pipeline. The pipeline can securely design and build products, including code review, secure part of SAST, software composition analysis, secret detection, infrastructure-as-code validation, container scanning, dependency checking and release policy evaluation. While AI can help identify potential risks, prioritize them based on context and make suggestions for remediation, it cannot release without human intervention. Signed artefacts, SBOMs, provenance records, attestations and validation reports are generated by successful execution [4, 6–8].

Layer 3 - Cloud-Native Runtime Environment. Verified artefacts are published to containers, Kubernetes clusters, service meshes, and cloud infrastructure. Jobs are run on this layer and maintain isolation and availability, deployment versions, configuration, service relationships and resource state are recorded here. Logs, metrics and traces, authentication events, and dependency observations are generated by runtime systems for later analysis [23]

Layer 4: Centralized Observability & Threat Intelligence. This layer is an evidence plane and not a decision engine. Runtime Telemetry is correlated with SBOM entries, SBOM Provenance, Deployments, Vulnerability Advisories, which includes Vulnerability KEVs, EPSS Vulnerability Probability, CVSS data and relevant threat intelligence. Immediately after a vulnerability has been reported, the layer is able to determine which artefacts include the component, where these artefacts are deployed and whether there are services associated with these artefacts which show related anomalies. An evidence package with source references, data-quality indicators and timestamps becomes the output.

Layer 5: AI Intelligence and Decision Support. AI decodes evidence and identifies abnormalities, links related events, provides root-cause hypotheses, prioritizes the exposures, estimates the operational impact, and compares response options. In each recommendation, indication of the evidence that has been relied on in making the recommendation, assumptions, information regarding uncertainty or confidence and an explanation sufficient for analyst analysis, should be provided. Because the AI component is itself a security relevant system [12], [13] model monitoring, data lineage and version record are necessary.

Layer 6: Policy Governance and Human Oversight. This layer is authoritative control layer. A policy engine simulates actions against the mandatory controls, service criticality, change-management rules, and regulatory obligations. It consists of two main features: prohibitions on actions and exceptions that are specifically beforehand authorizations of reversible actions that can be routed to explicitly designated human decision makers. Results include approvals, denials, policy reasons, action records, audit log, and incident documentation. While AI can provide reasons for a recommended change, it cannot make a change that is necessary but not in line with recommendations or make a self-authorised change with a high impact.

Table 2. Responsibilities and Outputs of the Six Framework Layers.

Layer	Primary function	Role of AI	Primary outputs
1. Mission, Risk, and Regulatory Context	Define criticality, risk, policy, and authority	None	Security objectives, thresholds, control and approval rules
2. AI-Augmented DevSecOps Pipeline	Secure design, build, verification, and release	Advisory code analysis and prioritisation	Trusted artefacts, SBOMs, signatures, provenance, attestations
3. Cloud-Native Runtime Environment	Deploy and operate verified services	Optional low-risk optimisation under policy	Running services, deployment state, telemetry
4. Centralized Observability and Threat Intelligence	Collect, normalise, and correlate evidence	Evidence preparation only	Contextual evidence packages and exposure maps
5. AI Intelligence and Decision Support	Analyse risk, impact, anomaly, and likely cause	Explainable recommendation with uncertainty	Prioritised findings, options, confidence/uncertainty, rationale
6. Policy Governance and Human Oversight	Enforce rules, authorise actions, and preserve accountability	Explanation support only	Approvals, blocks, controlled actions, audit and incident records

Standards and evidence mapping

The architecture does not supercede existing guidance. Adopts standards as supplementary sources of controls and evidence. CSF 2.0 contributes governance and enterprise outcomes, SSDF and SP 800-204D contribute to pipeline practices, guidance and SLSA contribute to component and provenance records, Zero Trust contributes to continuous verification, and KEV, EPSS and CVSS contribute various types of vulnerability evidence [3–10], [12], [14–19]. Table III lists the mapping used by the framework that has been reduced.

Table 3. Standards and Evidence Mapping.

Source	Primary purpose	Framework use	Representative evidence/control
NIST CSF 2.0 [5]	Enterprise cybersecurity governance and outcomes	Layers 1 and 6	Risk ownership, policy, profiles, governance outcomes
NIST SSDF and SP 800-204D [4], [6]	Secure development and CI/CD supply-chain integration	Layer 2	Secure practices, pipeline gates, release evidence
CISA 2025 SBOM draft guidance [7]	Component transparency	Layers 2 and 4	Component, version, supplier, dependency relationships
SLSA and in-toto [8], [9]	Build integrity and provenance	Layers 2 and 4	Attestations, build source, process and material records
NIST AI RMF [12]	AI risk management	Layers 5 and 6	Model governance, monitoring, validity, transparency
KEV, EPSS, and CVSS [15]–[17]	Observed exploitation, likelihood, and severity	Layers 4 and 5	Exploitation status, probability, severity vectors
NIST Zero Trust Architecture [19]	Continuous verification and least privilege	Layers 1 and 3	Identity, access, device and workload verification

Information flow and feedback

The forward flow starts with the mission requirements and then moves towards secure design and development, artefact verification, deployment, and operation. The feedback flow starts with runtime evidence and threat intelligence, and continues through contextual correlation and AI analysis to policy-governed action and validated learning. Preservation of evidence identity, and not just the flow of data in both directions, is the key design aspect. The audit trail shall include links for an SBOM component, signed artefact, deployment record, anomaly, recommendation, policy decision, and final action. Feedback can involve updating dependency allow/deny rules, modifying test cases, deployment strategies, monitoring coverage, model features, response playbooks and policy thresholds. Outcomes must be validated and only those outcomes should affect controls. Model evaluation should record an AI hypothesis that an analyst has rejected, but should not automatically promote it to an automated pipeline rule. This will stop the learning loop from becoming a validated control.

Decision hierarchy

The rule of authority within the framework is in this order: organisational policy is compulsory, human decision is authorised, and AI recommendation is provided. This order doesn't mean AI is not important, it's just what it could be best used for. AI can streamline and accelerate the analysis process; evidence can be assembled and interpreted by AI and policy, combined with personnel held accountable, will dictate whether or not a consequential action is allowed and appropriate

Let $E = \{S, O, T, M\}$ denote the contextual evidence package, where S is software-assurance evidence, O is operational telemetry, T is vulnerability and threat intelligence, and M is mission and service criticality. The AI component produces an assessment $A = h(E)$, including findings, recommended actions, explanations, and uncertainty. The final decision is $D = \pi(E, A, P)$, where P

is the deterministic policy set and π is the governed decision procedure. The model intentionally avoids collapsing evidence into a single opaque risk score.

The operation has three possible results. Mandatory policy violation equals rejection, blocking or containment. Any proposed action that is high impact, irreversible, not in an approved playbook or not well supported by evidence is brought to human review. Low-risk, reversible action can be decided upon and performed automatically only if there is an explicit policy allowing it, and monitoring is active after the action takes place. The workflow is presented in figure 3.

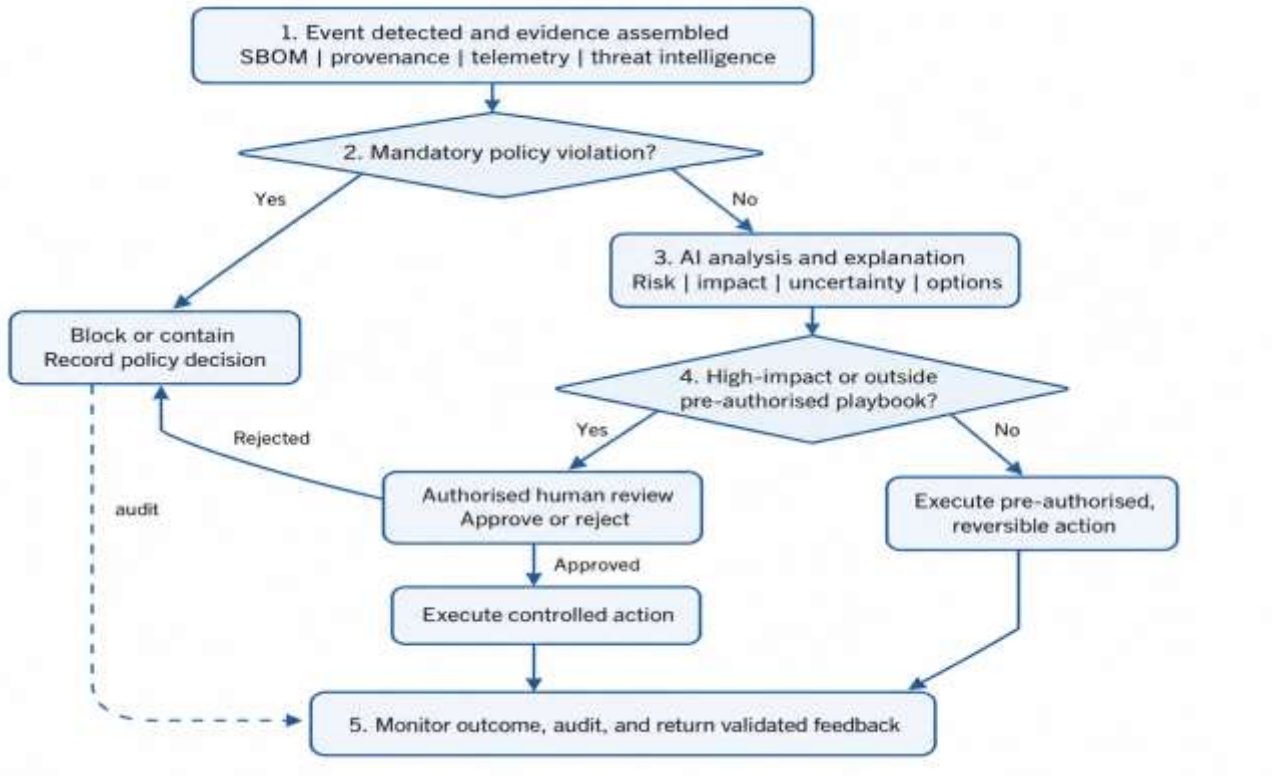

Figure 3. Policy-governed operational decision workflow.

Decision categories and controls

Automatic decisions. Low impact actions that have been approved ahead of time where no damage will be caused when executed and that do not require rolling back to a previous state based on previous operations but instead it will simply be "restarting" an unhealthy, non-critical instance, scaling within fixed limits, refreshing a cache, increasing logging, etc. Automation is based on a set of conditions: policy compliance, limited scope, social and economic impact, and a rollback strategy.

Assisted decisions. The ones include moderate risk and need the operator to be able to confirm the AI-assisted suggestion. Examples would be separating a workload, turning on or off a configuration change, running a remediation procedure, or undoing a configuration change from a recent rollout, if there is no loss of service. The interface should include showing of evidence, expected impact, alternatives, uncertainty and policy.

Governed decisions. These include mission-critical services, regulatory responsibilities, emergency powers, policy exemptions, disaster recovery scenarios, and major service shutdowns. They are at all times to be accompanied by named staff in appropriate manner. AI will always be advisory, no matter how high the confidence is.

Each category must have an audit record that includes the triggering event, data sources, versions of model and policy, steps recommended, author, and approval or denial, action, outcome and any

feedback added to pipeline. This record is used for incident review, regulatory accountability and subsequent model calibration and analyst reliance.

Scenario-Based Demonstration

A regional health care organization is running an e-health record application (EHR), along with a patient-scheduling application, as containerized microservices within the Kubernetes container orchestration system. Services are made available via an automated CI/CD pipeline. The EHR service is deemed a mission critical service, the scheduling service is deemed important but can tolerate brief downtime during an emergency rollback. The scenario is a hypothetical one and is provided only to illustrate the flow of information and separation of authority. During an update to the scheduling service, a developer adds another open source dependency. The pipeline conducts code dependency analysis, builds the container, creates an SBOM and verifies provenance, signs the artefact and keeps the proof. No policy violations are detected at this stage. The app is released via a rolling deployment process called a canary deployment, during which a small percentage of traffic is directed toward the new version of the app alongside logs, latency metrics, distributed traces, authentication events, and deployment data.

A new Vulnerability Disclosure is reported after deployment with the dependency rating as high-risk with high exploitation risks. Observability, at about the same time, identifies abnormal authentication requests, storage anomaly – abnormal database queries, increased latency, and increased resource utilization. The evidence layer associates the vulnerability record with the stored SBOM and provides verification that the canary contains the affected version and associates the onset of anomalies with the deployment window. The AI layer looks at the difference between the new deployment and the previous one applying the baseline and associates the anomaly cluster with the critical component, thus recommending to rollback to the last validated release. The recommendation will contain the component and version, impacted workload, supporting traces, change history, impact to the service, options for containment, and uncertainty. Illustrative confidence demonstrated in the operator interface would not be an actual, experiment-derived result in this conceptual study. The policy layer starts by making sure it is not involving any unsigned or unverified artefacts, and that the previous release is still approved. This then states that rollback can be done, but must be approved by a human since the scheduling service is participating in the environment of the mission-critical EHR system. The on-call security engineer and application owner review the evidence and approve a controlled rollback. Once executed, latency and authentication behaviour goes back to normal, the audit record is completed and a rule is added to the pipeline to block until the insecure dependency version is no longer used.

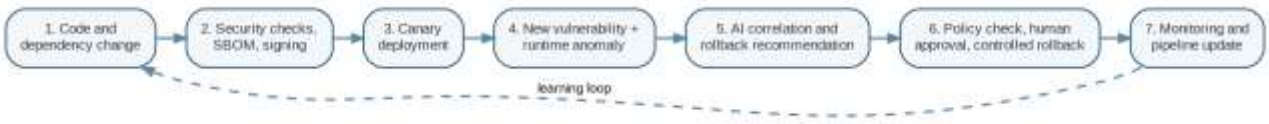

Figure 4. Healthcare scenario timeline and continuous learning loop.

Table 4. Scenario Evidence and Framework Response.

Stage	Evidence	Framework response	Authority
Code and build	Source change, dependency metadata, tests	Scan, create SBOM, verify provenance, sign artefact	Pipeline policy
Canary deployment	Signed artefact, deployment manifest, baseline	Deploy limited traffic and collect telemetry	Pre-authorised deployment control
Disclosure and anomaly	Vulnerability data, SBOM match, logs, metrics, traces	Create contextual exposure package	Evidence layer; no action authority
AI analysis	Evidence package, history, service graph	Correlate likely cause, compare options, recommend rollback	Advisory only
Policy and approval	Recommendation, service criticality, change rules	Confirm permitted response and request approval	Policy engine and authorised humans
Rollback and learning	Action record, post-change telemetry, incident outcome	Restore verified release, audit outcome, update pipeline rule	Controlled execution under approval

The scenario is meeting all six design requirements at a conceptual level. Dependency introduction to deployed workload maintains traceability. Operational evidence along with threat evidence are fused together and the observability layer does not make a decision. Advise: AI expedites the process of correlating and explaining. Deterministic policy is used to decide whether to allow rollback, and when human approval is needed. The controlled response is auditable and the result has to be confirmed and its result will feed a downstream pipeline gate. This is a product of internal coherence of the artefact, not real world effectiveness.

Discussion

The framework adds an explicit software-supply-chain-assurance-to-operations-resilience connection. SBOMs and provenance and signatures are often talked about as release controls, in this architecture these are still "in use" after deployment. The further deployment of these allows an organisation to rapidly respond to three questions: ‘Which services use an affected component?', ‘Are any of these services exhibiting relevant abnormal behaviours?', ‘What verified alternate services can be deployed?'. Another is separation of concerns. Unified observability prepares evidence, AI interprets that evidence, policy enforces non-negotiable rules and humans are responsible for consequential choices. This facilitates explainability as an operator can tell what was observed, what the model is thinking, what the policy might want to do, and who was responsible for taking the final action. It also helps to make failure analysis more precise; a poor outcome may be due to the quality of evidence involved, to the reasoning in the model, to policy design, or to policy approval or execution, rather than to a vague “AI system.”

The third is a practical pattern for integrating standards. Architecture features CSF 2.0 - governance, SSDF / SP 800-204D - securing dev process and CI/CD pipeline, SBOM / provenance - evidence for components / builds, AI RMF - governance of models, and vulnerability intelligence - prioritisation. The standards do not occur in an emergent, single conformance checklist; vision and the evidence of control for each standard are placed at the layer at which they have operational value. For implementation, it's important to have maturity as a foundation. Organisations must have reliable CI/CD controls, component inventories, signing and provenance, centralised

observability, consistent deployment identifiers, policy-as-code capability as well as clearly assigned decision roles. Most legacy systems don't generate the identifiers to relate source, build, deploy and runtime evidence. In these systems, the first implementation goal is to achieve evidence completeness, not to use "high tech" AI. Poor data quality is a big challenge. An incomplete SBOM, an outdated service list, no service inventory provided or an improperly normalised vulnerability feed can lead to bad exposure mapping. AI cannot resolve these foundational data issues; furthermore, AI-generated outputs can make poor data appear deceptively valid, making errors even more difficult to detect. Evidence packages will then have provenance metadata, evidence freshness metadata, coverage metadata, and confidence metadata. Model inputs and outputs need to be recorded, and access-control and privacy and retention measures need to be in place appropriate for the service. Organisational changes can be more challenging than technical changes. Different tools and escalations for the business of development, security, operations, risk, legal, and service owners. The structure relies on consensus regarding criticality, consented thresholds, rollback ownership, exception during emergency and acceptable automation. Designing policies should start with a limited number of reversible actions and be expanded if outcomes are monitored.

Limitations and future validation

The only restriction is lack of a feasible prototype and empirical findings. The healthcare scenario demonstrates that while the architecture was able to model the information and authority relationships required, it wasn't possible to measure the detection accuracy, latency, scalability, analyst workload reduction and improved incident outcomes. The paper also relies on the assumption that organisations are able to create reasonably complete SBOMs and provenance records, and that human decision makers can be called upon when needed.

The framework is targeted to enterprise and cloud-native software. Different applications like operational technology, industrial control systems, embedded platforms or safety-critical real-time environments might demand different telemetry, change windows, fail-safe controls, and/or certification. The decision categories also demand thresholds for each sector – what may be a low-risk action in a development service could be an unacceptable action for a healthcare or energy business.

Future work is needed involving a prototype built with a CI/CD platform, SBOM generator, signing and provenance service, Kubernetes environment, observability stack, policy engine, and AI decision-support component. It is advisable to have controlled experiments to measure time to recognize exposed services, mean time to detection and recovery, false positive / false negative rate, confidence calibration, analyst decision time, completeness of evidence gathered, policy-compliance rate, success of rollback, and post-incident control improvement. The cross-sector case studies should then be used to check the scalability, the fit within the governance structure, and the generalisability of the approaches. Examples of adversarial telemetry, poisoned threat intel, model manipulation, or attempting to evade model policy or boundaries should be tested with red teams.

Conclusion

Integrating secure development, supply chain proof of origin, runtime protection monitoring, AI analysis, and authority in operations into isolated workflows is an ineffective way of managing software supply chain risk. To resolve this, this paper proposed an AI-augmented DevSecOps framework with a governance perspective which has six layers and continuous evidence loop. The evidence is correlated and exposures are prioritised and estimated for impact, via the application of artificial intelligence, and response options are explained to the entire chain, but manifestation of impact within it is under deterministic control of the policy and authorised humans.

The healthcare scenario illustrated how a dependency add at development time could be found in a deployed service via SBOM and provenance data, then be correlated with runtime anomalies and vulnerability intelligence, be assessed with AI, be evaluated against policy, and finally get resolved with an approved rollback. Ultimately, the core value lies in the traceable separation between evidence, inference, authority, and action. Its contribution now is of the conceptual kind, and prototype implementation and empirical testing are needed for making claims about its operational performance. Nonetheless, the architecture offers a concentrated foundation for organizations that want to boost software-supply-chain resilience while implementing AI without compromising accountability.

References

[1] Cybersecurity and Infrastructure Security Agency, "Advanced persistent threat compromise of government agencies, critical infrastructure, and private sector organizations," Alert AA20-352A, Apr. 15, 2021.

[2] Cybersecurity and Infrastructure Security Agency, "Mitigating Log4Shell and other Log4j-related vulnerabilities," Alert AA21-356A, Dec. 2021.

[3] The White House, "Executive Order 14028: Improving the Nation's Cybersecurity," Washington, DC, USA, May 12, 2021.

[4] National Institute of Standards and Technology, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities, NIST SP 800-218, Feb. 2022, doi: 10.6028/NIST.SP.800-218.

[5] National Institute of Standards and Technology, The NIST Cybersecurity Framework (CSF) 2.0, NIST CSWP 29, Feb. 2024, doi: 10.6028/NIST.CSWP.29.

[6] National Institute of Standards and Technology, Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines, NIST SP 800-204D, Feb. 2024, doi: 10.6028/NIST.SP.800-204D.

[7] Cybersecurity and Infrastructure Security Agency, 2025 Minimum Elements for a Software Bill of Materials (SBOM): Guidance and Request for Comment, Aug. 2025.

[8] SLSA Community, SLSA Specification, Version 1.2, Nov. 2025.

[9] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in 28th USENIX Security Symposium, 2019, pp. 1393–1410.

[10] Z. Newman, J. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in Proc. 2022 ACM SIGSAC Conf. Computer and Communications Security, 2022, pp. 2353–2367, doi: 10.1145/3548606.3560596.

[11] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," Information and Software Technology, vol. 141, art. 106700, 2022, doi: 10.1016/j.infsof.2021.106700.

[12] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, Jan. 2023, doi: 10.6028/NIST.AI.100-1.

[13] F. Charmet, H. Tanuwidjaja, S. Ayoubi, P. Gimenez, Y. Han, H. Jmila, G. Blanc, T. Takahashi, and Z. Zhang, "Explainable artificial intelligence for cybersecurity: A literature survey," Annals of Telecommunications, vol. 77, nos. 11–12, pp. 789–812, 2022, doi: 10.1007/s12243-022-00926-7.
[14] National Institute of Standards and Technology, Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations, NIST SP 800-161 Rev. 1, May 2022, doi: 10.6028/NIST.SP.800-161r1.
[15] Cybersecurity and Infrastructure Security Agency, "Known Exploited Vulnerabilities Catalog," continuously updated.
[16] Forum of Incident Response and Security Teams, Common Vulnerability Scoring System Version 4.0: Specification Document, 2023.
[17] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, "Exploit Prediction Scoring System (EPSS)," Digital Threats: Research and Practice, vol. 2, no. 3, art. 17, 2021, doi: 10.1145/3436242.
[18] J. Jacobs, S. Romanosky, B. Edwards, and M. Roytman, "Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights," in 2023 IEEE European Symposium on Security and Privacy Workshops, 2023, pp. 194–206, doi: 10.1109/EuroSPW59978.2023.00027.
[19] National Institute of Standards and Technology, Zero Trust Architecture, NIST SP 800-207, Aug. 2020, doi: 10.6028/NIST.SP.800-207.
[20] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," MIS Quarterly, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
[21] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," Journal of Management Information Systems, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.
[22] J. Venable, J. Pries-Heje, and R. Baskerville, "FEDS: A framework for evaluation in design science research," European Journal of Information Systems, vol. 25, no. 1, pp. 77–89, 2016, doi: 10.1057/ejis.2014.36.
[23] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016, doi: 10.1145/2890784.
[24] Cybersecurity and Infrastructure Security Agency, Secure by Design: Shifting the Balance of Cybersecurity Risk, Oct. 2023.
[25] M. S. Melara and M. Bowman, "What is software supply chain security?" arXiv:2209.04006, 2022.