Security Policies and DevSecOps at enVisiAIn: A Comprehensive Framework

Course: Security Policies & Procedures **Instructor:** Arlecia Tyus **Student:** Douglass Brown **Evergreen State College — Spring 2026**

Introduction

When I started writing about enVisiAIn earlier this semester, I framed the company around an idea borrowed from cyberpunk fiction — the notion that a technology capable of giving sight back to someone is also, by design, a technology that can take it away. That tension hasn't left the work. It has only gotten sharper.

enVisiAIn is a near-future health technology startup operating in 2030. The company produces AI-augmented smart glasses, a companion mobile application, and a cloud-connected clinical dashboard, all purpose-built for blind and low-vision users. The product suite processes live video streams, retinal imaging, biometric readings, GPS location data, medical records, and real-time AI recommendations — often simultaneously, often at the edge, always on behalf of people who depend on the output for physical navigation and clinical care. In Paper 1, I established the security policies governing how the company's systems may be used, how risks are formally assessed, and under what conditions remote access is permitted. In Paper 2, I examined how HIPAA and GDPR compliance frameworks must be woven into the fabric of daily operations rather than maintained as a passive checklist. This paper brings those threads together under a single organizing principle: DevSecOps.

DevSecOps — the practice of embedding security into every phase of the software development lifecycle rather than treating it as a gate at the end — is not a new idea. But for a company like enVisiAIn, it is not merely good practice. It is an ethical obligation. A compromised firmware update pushed to a user's smart glasses could cause that person to misread a crosswalk signal. A poisoned AI model retrained on unvalidated data could produce navigation guidance that is systematically wrong. The stakes are not theoretical. They are the specific, embodied stakes of people who navigate the world with less margin for error than most of us. That reality is what makes DevSecOps at enVisiAIn a framework worth examining carefully.

From Waterfall to DevSecOps: The Methodology Evolution

To understand where DevSecOps fits, it helps to trace how development methodologies have evolved and why that evolution matters for a company like enVisiAIn.

The traditional Waterfall model treated security as a phase — something that happened after design, after development, and ideally before deployment. The approach had a certain logic to it in an era when software shipped on physical media and update cycles were measured in years. Security review teams operated at the end of the pipeline as quality gates. The problem, as the industry learned repeatedly through the 1990s and 2000s, was that by the time security reviewers saw code, the cost of fixing what they found was enormous. Architectural flaws discovered in the testing phase require re-engineering the architecture. Dependency vulnerabilities identified at deployment require rebuilding the build.

Agile methodology disrupted the Waterfall sequence by breaking development into shorter iterative sprints, placing working software over comprehensive documentation, and responding to change rather than following a fixed plan. This was an improvement for velocity, but Agile's relationship with security was initially uneasy. Security could feel like friction — a set of external requirements that slowed sprints down.

Many Agile shops continued to treat security as a post-sprint concern, batching security work into separate "hardening" sprints that felt like a return to Waterfall thinking.

DevOps resolved the tension between development velocity and operational reliability by treating operations as a shared responsibility of the development team — collapsing the wall between "we build it" and "we run it." CI/CD pipelines automated the path from code commit to production deployment. Infrastructure-as-code made environments reproducible. Monitoring closed the feedback loop. But DevOps, in its early forms, still tended to treat security as someone else's problem — the security team's job, applied somewhere in the pipeline as a scan or a review.

DevSecOps is the acknowledgment that this is not good enough, and the practical commitment to doing something about it. Security is shifted left — introduced at the earliest stages of design and planning — and distributed across the entire team rather than delegated to a separate function. Every developer becomes a security stakeholder. Every commit is an opportunity for automated security validation. Every deployment is preceded by verified compliance with the policies that govern the system.

For enVisiAIn, this evolution is not optional. The company operates in a regulatory environment (HIPAA, GDPR) that demands demonstrable security controls. It processes sensitive health data on the bodies of its users. It pushes software updates to devices that people wear on their faces. The question was never whether to take security seriously — the question was always how to operationalize seriousness at the speed that a competitive health technology startup requires.

The DevSecOps Culture at enVisiAIn

Before touching the toolchain, DevSecOps requires a cultural commitment. This is the piece that organizations most frequently skip, and it is the piece that most frequently causes security programs to fail.

At enVisiAIn, a DevSecOps culture means three things concretely.

Shared responsibility, not siloed expertise. Security is not the security team's job — it is every engineer's job. When a firmware engineer pushes a change to the smart glasses' object recognition module, that engineer is responsible for understanding the threat surface of that change: what new inputs does it accept, what data does it touch, what failure modes does it introduce? The security team provides tools, guidance, and review — but the primary author of a piece of code is the primary security owner of that code. This is a significant cultural shift from the traditional model, where security teams were treated as gatekeepers whose job was to find problems that developers were not expected to anticipate.

No-blame post-mortems. When a vulnerability is discovered — and in a company processing live biometric data with an AI inference pipeline, vulnerabilities will be discovered — the response cannot be punitive. Blame cultures suppress the honest reporting that security programs depend on. If the engineer who accidentally introduced a data serialization bug fears professional consequences for reporting it, the bug does not get reported promptly, it gets quietly fixed and quietly forgotten, and the organizational learning that could prevent the next bug never happens. enVisiAIn's security culture must create the conditions for engineers to surface problems early and loudly, with the confidence that the system's response will be to understand and improve rather than to assign fault.

Security education as ongoing practice. enVisiAIn engineers are not cybersecurity specialists — they are machine learning engineers, embedded systems developers, clinical software architects, mobile application developers. Each discipline has a different threat landscape. The ML engineer needs to understand data poisoning, model extraction attacks, and the security implications of training pipelines that touch real patient data. The firmware engineer needs to understand secure boot, code signing, and the attack surface of a device that receives over-the-air updates. The clinical dashboard developer needs to understand session management, audit log integrity, and the specific HIPAA technical safeguard requirements that apply

to systems accessing ePHI. Security training at enVisiAIn is not a once-a-year compliance checkbox — it is a rolling program of discipline-specific education tied to the actual threats the team encounters in its work.

Integrating Existing Policies into the DevSecOps Pipeline

The policies established in Paper 1 — the Acceptable Use Policy, the Risk Assessment Policy, and the Remote Access Policy — were written as governance documents. In a DevSecOps framework, governance documents only have value if they are operationalized. A policy that lives in a shared drive is not a security control. A policy that is encoded into automated pipeline enforcement is.

The Acceptable Use Policy in the pipeline. The AUP established that raw patient video and biometric data may not be used for AI model training without formal approval. In a DevSecOps pipeline, this constraint is implemented technically, not just declared in writing. The training data management system enforces approval workflows before any dataset containing identifiable patient data can be flagged as training-eligible. Data lineage tooling — such as Apache Atlas or a custom metadata layer on top of the clinical data warehouse — tags datasets with their consent and approval status, and the ML training pipeline rejects unapproved datasets at runtime as an automated enforcement gate, not a manual review step. The AUP's prohibition on credential sharing is similarly operationalized: the CI/CD pipeline uses short-lived machine credentials issued through HashiCorp Vault or AWS IAM roles rather than long-lived shared API keys, making credential sharing structurally impossible rather than merely prohibited.

The Risk Assessment Policy in the pipeline. The Risk Assessment Policy requires formal risk assessment annually and after major changes. In a DevSecOps context, "major changes" needs a precise definition and an automated trigger. enVisiAIn's pipeline incorporates threat modeling as a required artifact for any pull request that modifies the attack surface: new API endpoints, changes to the smart glasses' network communication stack, modifications to the AI inference pipeline's data handling, or updates to the clinical dashboard's authentication flow all require a threat model update before the PR is mergeable. This is not a heavyweight process — lightweight threat modeling tools like OWASP Threat Dragon can be integrated into the PR workflow so that the threat model is attached to the code change that necessitated it and reviewed alongside it. The annual formal risk assessment draws on the accumulated threat model history from the prior year's development cycle, making it a synthesis of ongoing work rather than a periodic scramble to document what everyone already knows.

The Remote Access Policy in the pipeline. The Remote Access Policy mandated VPN or zero-trust access, MFA, and company-managed devices for all remote work. In a DevSecOps pipeline, these requirements translate into infrastructure controls: developers without a passing device compliance check — up-to-date OS patches, active endpoint protection, current MDM enrollment — cannot authenticate to the development environment at all. The zero-trust model enforces continuous verification, not just at login but throughout the session. Pipeline secrets (deployment keys, signing certificates, production database credentials) are never accessible to developer workstations directly; they are injected into CI/CD runners by the secrets management system at runtime, scoped to the specific job that needs them and expired immediately after.

HIPAA and GDPR in the pipeline. The compliance work from Paper 2 maps onto the DevSecOps pipeline most concretely in three areas. First, static analysis rules encoding HIPAA technical safeguard requirements — encryption at rest and in transit, audit logging on all ePHI access, automatic session timeout — are incorporated into the SAST toolchain so that code violating these requirements fails the build rather than reaching review. Second, GDPR data minimization requirements are enforced by a data classification layer in the development environment: fields tagged as personal data are automatically scrutinized by linting rules that flag patterns likely to log, cache, or expose that data unnecessarily. Third, the right to deletion is implemented as a tested, versioned, deployed service — not a manual operational procedure — with integration tests that verify deletion requests propagate correctly across the clinical data store, the AI training dataset registry, and the smart glasses' cached local data.

DevSecOps Practices for enVisiAIn's Unique Environment

Generic DevSecOps tooling covers a lot of ground. But enVisiAIn's environment has three components that require practices tailored to their specific threat profiles.

Smart glasses firmware updates. The smart glasses run an embedded operating system with a real-time AI inference engine. Firmware updates are pushed over-the-air to devices worn by people who may have no other reliable technology for navigation. The DevSecOps requirements here go beyond what standard web application security covers.

Every firmware build is code-signed with a certificate managed by the engineering PKI team and stored in a hardware security module (HSM) — not on a developer workstation, not in a CI/CD environment variable. The device's secure boot implementation verifies the firmware signature before execution. The update delivery system implements a staged rollout: updates are pushed first to internal test devices, then to a small percentage of opted-in pilot users, with automated rollback triggers that revert the staged population to the previous firmware version if error rates or crash reports exceed defined thresholds within the first 24 hours. No firmware update reaches full population without passing this staged validation. This is not merely a quality control measure — it is a safety-critical risk management protocol. A blind user should never arrive at a firmware version that degrades their ability to navigate safely because the company pushed too fast.

AI model retraining pipeline. enVisiAIn's AI models are not static — they are retrained as new data becomes available, as edge cases are identified, and as the clinical evidence base for low-vision navigation evolves. The retraining pipeline is itself a software artifact with a software security profile.

Data ingestion into the training pipeline is gated by the consent and approval controls described above. The pipeline is additionally hardened against data poisoning: statistical validation checks run on each new training dataset to identify anomalous distributions that could indicate an attempt to bias model behavior — a concern that becomes acute when the training data is derived from real-world usage by vulnerable users. Model evaluation prior to production deployment includes not only standard performance metrics but adversarial robustness testing: does the model behave safely when inputs are perturbed in the ways a malicious actor might attempt? Model artifacts are versioned and signed using the same PKI infrastructure as firmware, and deployment to the inference serving layer requires the same staged rollout process. Model rollback is a first-class capability, not an afterthought.

Clinical dashboard deployments. The clinical dashboard is the interface through which optometrists, ophthalmologists, and vision rehabilitation specialists access patient data, review AI-generated assessments, and adjust device configurations for individual patients. It is a HIPAA-covered application with a relatively traditional web architecture, but its DevSecOps requirements are demanding because the data it exposes is among the most sensitive enVisiAIn handles.

Every deployment of the clinical dashboard is preceded by a DAST scan using OWASP ZAP or a commercial equivalent, run against a staging environment populated with synthetic patient data. The scan checks for OWASP Top 10 vulnerabilities and for the clinical-dashboard-specific risk of insecure direct object references — a vulnerability class that in this context could allow one clinician to access another's patients' records. Dependency scanning runs on every pull request to catch newly published CVEs in third-party libraries before they reach production. Audit logs capturing every ePHI access event are written to an append-only, tamper-evident log store that satisfies both the HIPAA audit control requirement and provides the forensic trail needed to investigate any future breach or unauthorized access incident.

Security Hierarchy Across the Methodology Evolution

Looking back across the three methodology eras, a pattern emerges in how security responsibility is distributed and where it sits in the hierarchy.

In Waterfall, security authority was top-down and late: a security team reviewed completed work and had the authority to block release, but limited ability to influence design. Security was hierarchically superior but operationally marginal.

In Agile, security responsibility became more distributed — security requirements could appear as user stories, as acceptance criteria, as items in the product backlog. But without dedicated tooling and pipeline enforcement, this distribution was largely nominal. The security person in the room had influence; security as an embedded system property did not.

In DevSecOps, the hierarchy flattens into a network. Security policy (governance layer) informs security tooling (pipeline layer) which enforces security requirements automatically (enforcement layer) and feeds security telemetry back to the governance layer for continuous improvement. At enVisiAIn, this network looks like: the CISO and security team own the policy and tooling standards; the platform engineering team implements those standards as pipeline controls; every product engineering team operates within those controls as part of their normal workflow; and the security telemetry — vulnerability scan results, failed build counts, incident reports, threat model updates — flows back to the CISO's team as evidence that the controls are working and as input to the next cycle of policy refinement.

This network model is more resilient than the Waterfall hierarchy because it does not depend on a single point of enforcement. A developer who bypasses a code review cannot bypass an automated SAST scan. A product team under deadline pressure cannot skip the staged rollout gate for a firmware update without an explicit override approved by the security team and logged in the change management system. Security is not a person who can be persuaded or a meeting that can be skipped — it is a property of the pipeline itself.

Conclusion: Why DevSecOps Is Not Optional

I want to return to where I started this semester: the idea that enVisiAIn's technology is, by its nature, a technology of trust. The company's users are people who have entrusted the company with something irreplaceable — their ability to navigate the world safely.

A compromised firmware update to a smart glasses device is not, in the abstract, a data breach. It is a navigation failure. It is a person who misreads a staircase. It is a clinician who receives a falsified AI recommendation for a patient who cannot independently verify what the device is telling them. The downstream consequences of a security failure at enVisiAIn are not measured in leaked credit card numbers or embarrassing PR cycles — they are measured in physical safety outcomes for a population that is already navigating a world not designed with them in mind.

DevSecOps is the answer to the question: how does a company that moves fast enough to remain competitive also move carefully enough to remain trustworthy? The answer is that speed and safety are not opposites — they are both properties of a well-engineered pipeline. Automated security scanning is faster than manual review. Staged rollout with automated rollback is safer than big-bang deployment and faster to recover from than a manually managed incident. Threat models attached to pull requests catch design flaws in minutes rather than the weeks a post-development security audit would require.

The policies from Paper 1 and the compliance frameworks from Paper 2 were never meant to be documents that enVisiAIn's employees read once and then filed away. They were always meant to be the specification for a system of technical controls that would make secure behavior the default, the automated, the normal. DevSecOps is the methodology that bridges the specification and the implementation — that turns governance documents into pipeline enforcement, that turns compliance requirements into code, that turns shared responsibility from a cultural aspiration into a structural reality.

For a company where a bad software deployment could affect a blind person's ability to navigate safely, DevSecOps is not a best practice. It is the minimum viable commitment to the people the technology is supposed to serve.

References

- National Institute of Standards and Technology (NIST). (2018). *Framework for Improving Critical Infrastructure Cybersecurity* (Version 1.1). https://www.nist.gov/cyberframework
- U.S. Department of Health and Human Services. (2013). *HIPAA Security Rule.* 45 C.F.R. Parts 160 and 164.
- European Parliament and of the Council. (2016). *General Data Protection Regulation (GDPR).* Regulation (EU) 2016/679.
- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution Press.
- OWASP Foundation. (2021). *OWASP Top Ten.* https://owasp.org/www-project-top-ten/
- Coles-Kemp, L., & Theoharidou, M. (2010). Insider threat and information security management. In C. W. Probst, J. Hunker, D. Gollmann, & M. Bishop (Eds.), *Insider Threats in Cyber Security* (pp. 45–71). Springer.
- Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous delivery and deployment: A systematic review of the state of research. *IEEE Access, 5*, 3909–3943.
- Ross, R., Pillitteri, V., Graubart, R., Bodeau, D., & McQuaid, R. (2019). *Developing Cyber-Resilient Systems: A Systems Security Engineering Approach* (NIST SP 800-160 Vol. 2). National Institute of Standards and Technology.

Word count: approximately 1,850 words (body text, excluding header and references)