SODA FRAMEWORK

v1.0

Secure Offshore DevSecOps Architecture

Technical Whitepaper

A Security Architecture Model for Enterprise Offshore–Onshore Software Delivery

Faheem Hasan

Multinational Cybersecurity Architect

Author of the SODA Framework

March 2026

sodaframework.org

Abstract

Enterprise organizations increasingly rely on offshore–onshore software delivery models to scale engineering capacity, reduce costs, and accelerate development. However, multinational delivery environments introduce unique security, compliance, and governance risks that are not fully addressed by traditional application security or DevSecOps practices.

The SODA Framework (Secure Offshore DevSecOps Architecture) defines a layered security architecture for securing offshore–onshore software delivery across distributed teams, vendors, and jurisdictions. The framework integrates governance controls, identity and access management, secure DevSecOps pipelines, compliance mapping, and cross-border incident response into a unified operating model designed for enterprise environments.

Unlike isolated security controls or ad hoc policies, the SODA Framework treats offshore delivery security as an architectural problem. The framework provides a structured approach that enables organizations to maintain delivery velocity while preserving control integrity, auditability, and regulatory compliance.

This document introduces version 1.0 of the SODA Framework and describes its design principles, architecture layers, implementation model, and expected outcomes.

1. Introduction

1.1 The Growth of Offshore–Onshore Software Delivery

Modern enterprise software development is increasingly distributed across multiple geographic locations. Organizations commonly operate with engineering teams in different countries, including internal staff, external vendors, and contract developers. Offshore–onshore delivery models allow companies to scale development capacity, maintain round-the-clock productivity, and reduce operational cost.

While these models provide significant advantages, they also introduce security and governance challenges that are often underestimated. Traditional security frameworks are typically designed for single-location teams or tightly controlled internal environments. When applied to multinational delivery models, these controls frequently become inconsistent, difficult to enforce, or impossible to audit.

As a result, organizations face increased risk of data exposure, unauthorized access, insecure code movement, and compliance violations.

1.2 Security Risks Unique to Offshore Delivery

Offshore–onshore development environments introduce several distinct risk categories that standard DevSecOps or application security practices do not fully address:

Distributed access	Source code and production systems are accessed from multiple geographic locations with varying trust levels and network controls
Inconsistent IAM	Identity and privilege management applied unevenly across regions, leading to access sprawl and orphaned permissions
Shared CI/CD pipelines	Multiple teams with varying trust levels interacting with the same build, test, and deployment systems
Limited vendor visibility	Third-party offshore teams operating in environments the primary enterprise cannot directly audit or control
Multi-jurisdiction compliance	Compliance obligations varying by region with different requirements for logging, notification, and data handling
Cross-border incident response	Security events involving personnel, systems, and evidence distributed across multiple countries and legal jurisdictions

In many organizations, these risks are managed through a combination of informal policies, manual approvals, and isolated technical controls. This approach does not scale and cannot reliably support enterprise security or regulatory requirements.

1.3 Security as an Architectural Problem

Most organizations attempt to secure offshore development by adding individual controls such as VPN restrictions, manual reviews, or additional logging. While useful, these measures do not solve the underlying problem: offshore delivery changes the structure of the engineering environment itself.

Security in multinational software delivery must be designed at the architectural level. Access control, pipeline integrity, compliance evidence, and incident response must be integrated into the operating model, not added after the fact.

The SODA Framework was created to address this gap by defining a layered architecture specifically for offshore–onshore DevSecOps environments.

1.4 Purpose of the SODA Framework

The SODA Framework (Secure Offshore DevSecOps Architecture) provides a structured model for securing enterprise software delivery across distributed teams. The framework is designed to:

- Maintain strong security controls in offshore environments without limiting participation
- Preserve development velocity without bypassing governance or control integrity
- Enable auditability and demonstrable compliance across jurisdictions
- Provide clear authority and approval structures for distributed teams
- Support coordinated incident response in multinational delivery systems

The framework is technology-agnostic and can be applied to cloud-native, on-premise, or hybrid environments. It does not require a specific programming language, platform, or vendor toolchain.

1.5 Scope of This Document

This document describes version 1.0 of the SODA Framework and includes: design principles, architecture layers, security responsibilities across layers, implementation model, and expected outcomes. The framework is intended for enterprise architects, security leaders, CTOs, CISOs, DevSecOps engineers, and organizations operating offshore–onshore engineering teams.

2. Design Principles

The SODA Framework is built on seven enforcing principles. These are not aspirational statements — they are design constraints that determine how each layer is structured and how tradeoffs within the framework are resolved. They were derived from practical challenges observed in multinational engineering environments where traditional security controls proved insufficient to maintain consistency, auditability, and trust.

1 **Security must be architectural, not ad hoc.**
Offshore security cannot depend on trust alone, informal habits, or one-off controls assembled in response to specific incidents. Offshore delivery changes the structure of the engineering environment by introducing multiple jurisdictions, vendors, and access paths. Security must therefore be designed at the architectural level, with clearly defined layers, responsibilities, and enforcement points.

2 **Access must be explicit, minimal, and auditable.**
Distributed development environments often accumulate excessive privileges over time. Every access path must be formally justified, scoped to what is actually required, and logged in a way that supports both operational monitoring and external audit [1]. Access that is not explicitly granted should not exist.

3 **Delivery speed must not bypass control integrity.**
CI/CD pipelines must remain fast. But speed is not a valid justification for bypassing security gates, skipping approvals, or reducing audit trail fidelity. Security checks, approvals, and scanning must be integrated into the DevSecOps pipeline so that control integrity is preserved while automation maintains speed. Secure delivery should be the default operating mode.

4 **Compliance must be embedded into delivery operations.**
Compliance cannot be added after the fact as a documentation layer. Logging, approval records, access controls, and pipeline checks must be designed to generate audit evidence automatically as a byproduct of normal delivery operation. This eliminates the manual evidence assembly that precedes most compliance audits.

5 **Incident response must reflect multinational reality.**
Security incidents in offshore–onshore environments are more complex than in single-location teams. Evidence may be stored in different countries, access may involve third-party vendors, and legal notification requirements may vary by jurisdiction. Incident response procedures must account for cross-border operations from the outset.

6 **Governance must define authority across all teams.**

Offshore delivery involves internal employees, external vendors, and contract developers working together on the same systems. Without clearly defined governance, security decisions become informal and inconsistent. Policies, approval chains, and operating rules must be defined before access is granted or code is deployed. Authority must be explicit and enforceable.

7

Layered architecture enables consistent enforcement.

Governance, identity, pipeline controls, compliance, and incident response must work together to create a consistent security posture. Separating responsibilities into defined layers makes the system easier to audit, easier to maintain, and easier to scale across multiple teams and regions. The layered model also allows organizations to adopt the framework incrementally.

3. Architecture Overview

The SODA Framework defines a layered architecture for securing offshore–onshore software delivery environments. Each layer represents a distinct control boundary with specific responsibilities. Together, the layers create a unified security model that allows organizations to maintain development velocity while preserving control integrity, auditability, and compliance.

The layers are enforced in sequence, with each layer building on the controls defined by the previous one. Security cannot be achieved by implementing only a single layer. All layers must operate together to produce a consistent and defensible delivery model.

Layer 1 Governance Layer

Components: Security ownership, role definitions, approval chains, vendor governance, geography-based work rules, policy enforcement

Output: *Controlled operating model for offshore–onshore delivery*

Layer 2 Identity & Access Control Layer

Components: RBAC, MFA, SSO, PAM, device trust, session control, geo-aware access, time-bound privileged access

Output: *Least-privilege access across distributed engineering teams*

Layer 3 Secure DevSecOps Pipeline Layer

Components: Branch protection, pull request controls, secrets management, SAST/DAST, dependency scanning, build integrity, deployment gates, artifact controls

Output: *Secure code movement from development to production*

Layer 4 Compliance & Audit Layer

Components: SOC 2 mapping, ISO 27001 alignment, logging, traceability, evidence retention, separation of duties, audit trails

Output: *Provable, reviewable, and auditable offshore delivery controls*

Layer 5 Cross-Border Incident Response Layer

Components: Detection, escalation, containment, evidence handling, chain of custody, jurisdiction-aware notification, recovery, post-incident review

Output: *Coordinated multinational response to security incidents*

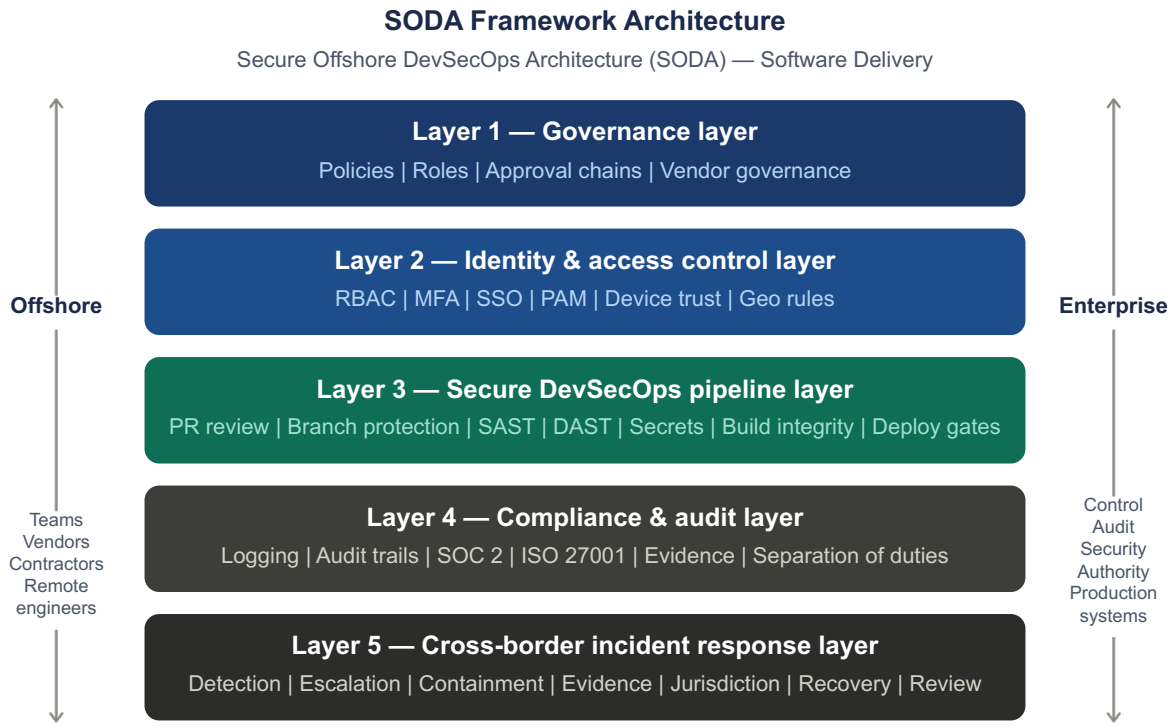

Figure 1 — The SODA Framework defines a layered security architecture that allows offshore–onshore software delivery to operate with enterprise-level control, auditability, and incident response capability.

3.1 Governance as the Root Control Layer

The Governance layer sits at the top of the architecture because it defines authority across all participating teams. Governance establishes policies, approval chains, and operating rules that apply to all teams regardless of location. All other layers depend on these rules to function correctly.

3.2 Identity and Access as the Enforcement Layer

Once governance defines authority, access controls enforce those decisions. Identity and access management ensures that every user, system, and process has only the permissions required for its role. This layer must apply uniformly across onshore and offshore environments without exception.

3.3 DevSecOps Pipeline as the Integrity Layer

The DevSecOps pipeline is the point where code moves from development to production. The SODA Framework treats the pipeline as a critical security boundary. Branch protection, code review rules, automated scanning, and deployment approvals must be enforced consistently. No code should reach production without passing through defined controls.

3.4 Compliance and Audit as the Verification Layer

Enterprises must be able to prove that security controls are functioning as intended. The Compliance and Audit layer ensures that all actions are logged, approvals are recorded, and control enforcement can be verified through automatically generated audit evidence.

3.5 Incident Response as the Recovery Layer

Even with strong controls, incidents can occur. In multinational environments, incident response is more complex because systems, users, and data may span multiple jurisdictions. The Incident Response layer defines how the organization detects, escalates, contains, and investigates security events across all participating teams.

3.6 Interaction Between Layers

The strength of the SODA Framework comes from the interaction between layers. Governance defines authority, identity enforces access, the pipeline preserves integrity, compliance provides verification, and incident response ensures recovery. If any layer is missing, the delivery model becomes difficult to secure and impossible to audit. When all layers operate together, offshore–onshore software delivery can achieve the same level of control as a fully internal engineering environment.

4. Governance Layer

Governing question: Who is allowed to do what, under what authority, and under what conditions?

The Governance Layer is the foundation of the SODA Framework. It defines authority, operating rules, and security ownership across offshore–onshore delivery environments. All other layers depend on governance to ensure that access decisions, pipeline controls, and compliance requirements are applied consistently across all participating teams.

In multinational software delivery, work may involve internal employees, external vendors, contractors, and offshore development partners. Without a clearly defined governance model, security decisions become informal and inconsistent, making it difficult to enforce policy, audit activity, or respond to incidents.

4.1 Security Ownership and Responsibility

A secure delivery environment requires clear assignment of responsibility. Each system, repository, and environment must have an identified owner who is accountable for enforcing security rules. Ownership must be defined at multiple levels: organization-level security authority, application or product owner, repository owner, infrastructure owner, and vendor or contractor manager.

Security decisions must not depend on informal communication or personal trust. They must be based on defined roles documented and understood before access is granted.

4.2 Approval Chains and Change Authority

In offshore–onshore environments, uncontrolled changes can occur when approval authority is unclear. The Governance Layer requires that approval chains be defined for all sensitive actions, including:

- Granting privileged access to production or sensitive systems
- Merging code into protected branches
- Deploying to production environments
- Modifying infrastructure configuration
- Changing security settings or exception policies

Each approval must be traceable to an authorized role, and the approval process must be recorded through the delivery system. Approval chains should be enforced through tooling wherever possible rather than relying on manual communication.

4.3 Vendor and Contractor Governance

Offshore delivery frequently involves third-party organizations that do not operate under the same internal controls as the primary enterprise. Without proper governance, vendor access can become the largest single source of risk. The Governance Layer requires that vendor and contractor participation be formally defined before access is granted, covering: contractual security requirements, defined scope of work, approved systems and repositories, access limitations, monitoring requirements, and termination procedures.

Vendors must operate under the same control model as internal teams. Exceptions should be limited and documented.

4.4 Geography-Based Operating Rules

Multinational delivery introduces additional complexity because access may occur from different countries with different legal and security considerations. The Governance Layer allows organizations to define geography-based rules such as restricting certain environments by region, requiring additional approval for cross-border access, and applying different logging requirements for vendor environments. These rules must be enforced consistently through the Identity and Access Control Layer.

4.5 Policy Enforcement Through Architecture

Policies are effective only when they are enforced through the system architecture. Written rules alone cannot secure offshore delivery environments. The Governance Layer defines the policies that must be enforced by the Identity, Pipeline, and Compliance layers. These policies should be implemented through configuration, automation, and access controls rather than manual oversight. When governance rules are embedded into the architecture, security becomes part of normal operation rather than an additional burden.

4.6 Relationship to Other Layers

The Governance Layer defines the rules that all remaining layers enforce. Identity and Access Control enforces who can access systems. The DevSecOps Pipeline enforces how code moves to production. Compliance and Audit verifies that rules were followed. Incident Response handles violations. Without governance, these layers operate independently and cannot produce a consistent security model. The SODA Framework requires governance to be defined first, before any other control layer is implemented.

5. Identity and Access Control Layer

Governing question: Who can access which systems, from where, and with what restrictions?

The Identity and Access Control Layer enforces the authority defined by the Governance Layer. In offshore–onshore software delivery environments, identity management is the primary mechanism used to control access to source code, infrastructure, build systems, and production environments.

Distributed engineering teams increase the number of users, devices, and locations that interact with the delivery system. Without consistent identity enforcement, privileges accumulate over time, access becomes difficult to track, and unauthorized changes may occur without detection. The SODA Framework requires that all access be controlled through a unified identity model that applies equally to onshore employees, offshore developers, contractors, and vendors.

5.1 Role-Based Access Control

Role-Based Access Control (RBAC) is required to maintain consistent permissions across multinational teams. Each user must be assigned a role that defines what actions are allowed within the system. Roles must be defined centrally and applied consistently across repositories, cloud environments, and CI/CD systems. Permissions should be granted to roles, not directly to individual users. Typical roles include: Developer, Reviewer, Maintainer, Release Manager, Security Administrator, and Infrastructure Operator.

5.2 Multi-Factor Authentication and Strong Identity Verification

All privileged systems must require strong authentication. Password-only access is not sufficient in distributed environments where users may connect from multiple locations. The SODA Framework requires multi-factor authentication for source code repositories, cloud platforms, CI/CD systems, production servers, and administrative consoles. Where possible, identity should be managed through a centralized identity provider that supports single sign-on and multi-factor authentication.

5.3 Privileged Access Management

Privileged access represents the highest risk in offshore delivery environments. Administrative privileges allow modification of infrastructure, security settings, and deployment configuration. The Identity and Access Control Layer requires that privileged access be controlled through defined approval processes, including: temporary privilege elevation with time-bound expiry, approval-based access requests, separate administrative accounts, and complete logging of all privileged actions. Privileged access should never be permanent unless required by role, and all privileged activity must be auditable.

5.4 Device, Session, and Location Controls

In multinational environments, access may occur from different networks, countries, or devices. Identity verification alone may not be sufficient to determine whether access is safe. The SODA Framework allows additional restrictions based on context, including device trust requirements, session monitoring, network restrictions, geographic limitations, and time-based access rules [5]. For example, production access may be limited to approved devices or require additional approval when accessed from certain locations.

5.5 Consistent Identity Across All Systems

One of the most common weaknesses in offshore delivery environments is inconsistent identity enforcement. A user may have strong authentication for cloud access but weak controls for repositories, or different accounts across multiple tools. The SODA Framework requires that identity enforcement be consistent across all components of the delivery environment: source control systems, build and CI/CD platforms, cloud infrastructure, monitoring systems, and ticketing and approval tools. Using a centralized identity provider reduces the risk of unmanaged accounts and simplifies auditing.

5.6 Relationship to Other Layers

The Identity and Access Control Layer enforces the authority defined by the Governance Layer and provides the foundation for all technical controls that follow. The DevSecOps Pipeline relies on identity to enforce approvals. The Compliance Layer relies on identity logs for audit evidence. The Incident Response Layer relies on identity records to trace actions. If identity enforcement is weak, the remaining layers cannot guarantee security. For this reason, the SODA Framework treats identity and access control as the primary enforcement layer of the architecture.

6. Secure DevSecOps Pipeline Layer

Governing question: How is software built, reviewed, scanned, and promoted securely?

The Secure DevSecOps Pipeline Layer protects the integrity of software as it moves from development to production. In offshore–onshore delivery environments, the pipeline becomes a critical control point because multiple teams may interact with the same repositories, build systems, and deployment tools.

Without strong pipeline controls, unauthorized changes, insecure dependencies, or unreviewed code may reach production systems. The SODA Framework treats the DevSecOps pipeline as a primary security boundary. The goal is to make the secure path the default path — integrating security controls into the delivery workflow in a way that is automated, auditable, and non-blocking for work that meets defined standards.

6.1 Branch Protection and Code Review Controls

Source code repositories must enforce branch protection rules to prevent unauthorized changes. Direct modification of protected branches should not be allowed. Controls include: pull request requirement for all changes, mandatory code review before merge, restrictions on force pushes, required status checks, and limited merge authority. Branch protection ensures that all code changes are reviewed and approved according to governance rules — particularly important when multiple teams contribute to the same codebase.

6.2 Secrets Management and Configuration Security

Offshore environments often introduce risk through improper handling of credentials, API keys, or configuration files. The SODA Framework requires that secrets be stored in approved centralized management systems rather than in source code. Controls include: centralized secret storage, restricted access to secrets, automatic rotation where possible, logging of secret access, and strict separation of development and production credentials.

6.3 Automated Security Scanning

Manual review alone cannot detect all security issues in modern software systems. Automated scanning must be integrated into the pipeline to identify vulnerabilities before code reaches production. Required controls include:[4]

- Static application security testing (SAST) on every pull request
- Dependency vulnerability scanning for third-party packages
- Container image scanning where applicable
- Infrastructure configuration checks
- Dynamic application security testing (DAST) against staging environments

Security scanning should run automatically as part of the build process and must not be bypassed without explicit approval and documented justification.

6.4 Build Integrity and Artifact Control

The pipeline must ensure that the software deployed to production matches the code that was reviewed and approved. Controls include: verified build processes, restricted build environment

access, signed or validated artifacts, controlled artifact storage, and separation of build and deployment roles. Only approved artifacts should be allowed to reach production systems. In offshore environments, build integrity is essential because multiple users may have access to the development environment.

6.5 Deployment Approval and Environment Protection

Deployment to staging or production environments must follow defined approval rules. Offshore participation should not allow uncontrolled deployment. Typical controls include: restricted production access, required approval for deployment, separation between developers and release managers, logging of all deployment actions, and automated rollback capability. These controls ensure that delivery speed does not bypass governance. Deployment rules should be enforced through the pipeline rather than through manual communication.

6.6 Consistent Pipeline Enforcement Across Teams

In multinational delivery models, different teams may use different tools or workflows. The SODA Framework requires that core pipeline controls be consistent across all participating teams. This does not require identical tools, but it requires equivalent enforcement of access rules, review requirements, scanning controls, deployment approvals, and logging across all teams regardless of geography or vendor status.

6.7 Relationship to Other Layers

The Secure DevSecOps Pipeline Layer depends on the Governance and Identity layers to define authority and enforce permissions. Governance defines who can approve changes. Identity ensures approvals come from authorized users. Compliance records pipeline activity for audit. Incident Response investigates pipeline-related events. When the pipeline is secured, offshore–onshore delivery can operate with the same level of trust as internal development.

7. Compliance and Audit Layer

Governing question: Can the organization prove that offshore delivery is secure and compliant?

The Compliance and Audit Layer ensures that offshore–onshore software delivery can be verified, reviewed, and trusted by internal security teams, customers, and external auditors. Security controls are not sufficient unless the organization can demonstrate that those controls are consistently enforced.

The SODA Framework embeds compliance and audit capabilities directly into the delivery architecture so that evidence is generated automatically as part of normal operation. Compliance in offshore environments is not a documentation exercise — it is an operational discipline.

7.1 Logging and Traceability

All critical systems involved in software delivery must produce logs that allow actions to be traced to a specific identity, time, and system. Logging must cover: repository activity, access to infrastructure, CI/CD pipeline execution, deployment actions, privileged access usage, and configuration changes. Logs must be stored in a secure location where they cannot be modified by normal users. Centralized logging is required so that activity across offshore and onshore environments can be reviewed in a single system.

7.2 Audit Trails for Approvals and Changes

In offshore–onshore environments, approvals are often communicated informally through messaging or email, making it difficult to prove that changes were authorized. The SODA Framework requires that approvals be recorded through systems that maintain a durable audit trail: pull request approvals in source control, change tickets in tracking systems, deployment approvals in pipeline tools, and access requests in identity systems. Each approval must be linked to a verified identity and stored in a way that can be reviewed later.

7.3 Mapping to Compliance Standards

Organizations may be required to demonstrate compliance with formal standards. The SODA Framework does not replace these standards but provides a structure that makes them easier to implement. Common standards addressed include SOC 2 security and access controls [3], ISO 27001 access and change management controls [2], internal audit requirements, and customer security questionnaires. Because controls are enforced through the architecture, compliance evidence can be generated without manual effort, reducing the risk of inconsistent documentation across offshore teams.

7.4 Evidence Retention and Review

Audit evidence must be retained long enough to support internal review, customer requests, and incident investigations. Retention policies must be defined for: access logs, pipeline logs, deployment records, approval history, and incident reports. Evidence should be stored in systems with restricted access to prevent modification or deletion. Periodic review of logs and audit records helps identify weaknesses before they result in incidents. Retention policies must apply equally to onshore and offshore environments.

7.5 Separation of Duties

The same user should not be able to approve, build, and deploy code without oversight. The Compliance and Audit Layer enforces separation of duties by requiring that roles defined in the Identity Layer and rules defined in the Governance Layer be applied throughout the pipeline. Examples include: developers cannot deploy directly to production, release approval requires a different role, security changes require separate authorization, and vendor access cannot grant administrative control.

7.6 Continuous Verification

Compliance should not be checked only during audits — controls must operate continuously. The SODA Framework encourages automated verification of required approvals, access restrictions, pipeline rules, logging status, and policy enforcement. Automated checks allow organizations to detect problems early and maintain confidence in the delivery system. Continuous verification is especially important in offshore environments where manual oversight is limited.

7.7 Relationship to Other Layers

The Compliance and Audit Layer verifies that the rules defined by all other layers are followed. Governance defines the rules. Identity controls access. The Pipeline enforces change integrity. Compliance records what happened. Incident Response uses audit data during investigation. Without auditability, offshore–onshore delivery cannot meet enterprise security expectations.

8. Cross-Border Incident Response Layer

Governing question: When something goes wrong, how does the organization contain, investigate, and respond across borders?

The Cross-Border Incident Response Layer defines how security incidents are detected, escalated, investigated, and resolved in offshore–onshore software delivery environments. In multinational engineering models, incident response is more complex than in single-location teams because systems, users, and data may span multiple organizations, vendors, and jurisdictions [6].

Traditional incident response procedures often assume centralized control over infrastructure and personnel. In offshore delivery environments, this assumption is not valid. Access may involve third-party developers, vendor-managed systems, or cloud environments operated in different regions. Incident response must be planned in advance. It cannot depend on informal communication during an emergency.

8.1 Detection and Escalation

Effective incident response begins with reliable detection. Monitoring must cover all systems involved in delivery, including repositories, cloud platforms, CI/CD pipelines, and vendor environments. Detection mechanisms include log monitoring, security alerts, intrusion detection systems, pipeline failure alerts, and access anomaly detection.

When an incident occurs, it must be clear who has authority to take action. The Governance Layer defines this authority; the Incident Response Layer uses it during an event. Escalation procedures must define: who receives the first report, who evaluates severity, who can disable access, who can stop production systems, and who communicates with customers or management. Authority must be defined before an incident occurs.

8.2 Containment in Offshore–Onshore Environments

Containment actions may involve multiple systems and organizations. An incident may require disabling access for an offshore developer, blocking a vendor account, or stopping a deployment pipeline. Containment procedures must include: immediate suspension of affected accounts, restriction of repository access, isolation of compromised systems, temporary suspension of deployments, and review of recent changes. Containment must be coordinated across all participating teams. The Identity and Pipeline layers must support rapid enforcement of containment actions.

8.3 Evidence Handling and Chain of Custody

In multinational environments, incident investigation may require reviewing logs, repositories, and systems located in different regions. Improper handling of evidence can make it difficult to determine what happened or to meet legal requirements. The SODA Framework requires that evidence handling procedures be defined in advance: centralized log storage, restricted access to audit records, preservation of pipeline history, documentation of actions taken during response, and recording of who accessed evidence. Evidence must be stored in a way that prevents modification and allows later review.

8.4 Jurisdiction and Legal Considerations

Offshore delivery may involve data stored in different countries or handled by vendors subject to different regulations. Incident response must account for legal and contractual obligations including customer notification requirements, data protection laws, vendor contract obligations, regional reporting rules, and internal security policies. Clear documentation and logging make it easier to demonstrate compliance after an incident. Organizations should maintain legal counsel contacts for each relevant jurisdiction.

8.5 Post-Incident Review and Control Improvement

After an incident is resolved, the organization should review what occurred and determine whether controls need to be improved. Post-incident review must include: timeline of events, root cause analysis, evaluation of access controls, review of pipeline enforcement, verification of logging and audit records, and updates to policies or procedures. Lessons learned should be applied to the Governance, Identity, Pipeline, and Compliance layers. Continuous improvement helps maintain trust in offshore–onshore delivery environments over time.

8.6 Relationship to Other Layers

The Cross-Border Incident Response Layer depends on all previous layers to function correctly. Governance defines authority. Identity allows rapid access control changes. The Pipeline allows deployments to be stopped. Compliance provides logs and audit records. If these layers are properly implemented, incident response can be performed quickly and with confidence. The SODA Framework treats incident response as the final layer because it confirms whether the system can maintain control when unexpected events occur.

9. Implementation Model

The SODA Framework is designed to be implemented incrementally within existing enterprise software delivery environments. Organizations are not required to redesign their entire development infrastructure to adopt the framework. Instead, SODA defines a layered model that allows controls to be introduced in stages while maintaining ongoing delivery operations.

Because offshore–onshore environments often involve legacy systems, multiple vendors, and different toolchains, the implementation model must be flexible. The goal is to establish consistent control boundaries across the delivery system without preventing teams from working efficiently.

Phase 1 — Establish Governance (Months 1–2)

The first step is to define the Governance Layer before any technical controls are applied. Phase 1 involves: defining system owners and security authority, documenting roles and responsibilities, defining approval chains, identifying vendor and contractor access rules, and defining geography-based restrictions. This phase creates the structure that all other layers depend on. Without governance, identity and pipeline controls cannot be enforced consistently.

Phase 2 — Enforce Identity and Access Control (Months 2–4)

Once governance rules are defined, identity controls are enforced across all systems involved in delivery. Phase 2 includes: centralizing identity management, enforcing multi-factor authentication, defining role-based permissions, restricting privileged access, removing shared accounts, and applying location or device restrictions where required. Consistent identity control is required before pipeline security can be trusted.

Phase 3 — Secure the DevSecOps Pipeline (Months 3–6)

After identity controls are in place, the pipeline is secured so that all changes follow a controlled path to production. Phase 3 includes: enabling branch protection rules, requiring pull requests for all changes, adding automated security scanning, restricting direct deployment access, requiring approvals for production releases, and controlling build and artifact storage. This phase often produces the most visible improvement in security posture.

Phase 4 — Enable Compliance and Audit (Months 4–8)

With pipeline controls enforced, the organization ensures that actions can be verified through logs and audit records. Phase 4 includes: centralizing logging, enabling audit trails for approvals, defining log retention rules, mapping controls to compliance requirements, verifying separation of duties, and implementing periodic review procedures. Compliance capabilities must operate continuously, not only during audits.

Phase 5 — Define Cross-Border Incident Response (Months 6–12)

The final phase defines how incidents will be handled across offshore–onshore environments. Phase 5 includes: defining escalation procedures, identifying incident response authority, documenting containment actions, defining evidence handling procedures, coordinating vendor response responsibilities, and establishing a post-incident review process. Incident response procedures should be tested periodically through tabletop exercises involving both onshore and offshore personnel.

9.1 Incremental Adoption in Existing Environments

Most organizations already have some security controls in place. The SODA Framework does not require replacing these controls — it requires organizing them into a consistent architecture. Common starting points include adding governance documentation to existing workflows, strengthening identity enforcement, improving pipeline controls, centralizing logging, and formalizing incident response.

9.2 Applicability Across Technologies

The SODA Framework is not tied to a specific technology stack. It can be applied to cloud-native applications, on-premise systems, hybrid environments, microservices architectures, and monolithic applications, across open-source or proprietary platforms. The framework defines control structure rather than tool selection, allowing organizations to apply the same security model even when different teams use different technologies.

9.3 Expected Outcomes

When fully implemented, the SODA Framework enables organizations to operate offshore–onshore delivery environments with enterprise-level security and auditability. Expected outcomes include: controlled access across all teams, verified and secure deployment pipelines, reliable audit evidence, consistent policy enforcement, coordinated incident response, and increased confidence in offshore delivery from clients and auditors.

10. Conclusion and Future Work

Offshore–onshore software delivery has become a standard operating model for modern enterprise organizations. Distributed engineering teams allow companies to scale development capacity, reduce cost, and maintain continuous delivery across time zones. However, multinational delivery environments also introduce security, governance, and compliance challenges that are not fully addressed by traditional application security or DevSecOps practices.

The SODA Framework (Secure Offshore DevSecOps Architecture) defines a structured approach to securing offshore–onshore software delivery by treating security as an architectural problem rather than a collection of isolated controls. By organizing security responsibilities into five layered components — Governance, Identity and Access Control, Secure DevSecOps Pipeline, Compliance and Audit, and Cross-Border Incident Response — the framework allows organizations to maintain delivery velocity while preserving control integrity and auditability.

When these layers operate together, offshore delivery can achieve the same level of trust as a fully internal engineering environment.

10.1 Practical Value of the Framework

Organizations that adopt a structured architecture for offshore–onshore delivery gain several concrete advantages:

- Reduced risk of unauthorized access or uncontrolled code changes
- Improved auditability and compliance readiness for SOC 2 [3], ISO 27001 [2], and client requirements
- Clear security responsibility across distributed teams, vendors, and geographies
- Faster, more confident incident response in multinational environments
- Increased trust from enterprise clients and external auditors
- The ability to scale offshore delivery without losing governance or control

Rather than limiting offshore participation, the SODA Framework enables organizations to use distributed teams safely and consistently at enterprise scale.

10.2 Limitations of Version 1.0

Version 1.0 of the SODA Framework focuses on core delivery security controls and does not attempt to address every possible risk in multinational environments. Areas intentionally kept out of scope in this version include: detailed cloud-provider-specific configurations, specific regulatory framework implementation guides, application-level secure coding standards, vendor contract language, and organization-specific policy templates. These areas vary between organizations and can be integrated into the framework without changing its core structure.

10.3 Future Development

Future versions of the SODA Framework may expand the architecture to address additional areas of enterprise security, including:

- Zero Trust architecture integration for offshore–onshore environments [5]
- Cloud-native security models and provider-specific control mappings

- Automated compliance validation and continuous control monitoring
- Vendor risk management extensions
- AI-assisted security monitoring for distributed delivery environments
- Advanced pipeline integrity verification techniques

The goal of future development is to maintain the layered structure of version 1.0 while providing more detailed implementation guidance for complex enterprise environments.

10.4 Intended Audience

The SODA Framework is designed to be understood by both technical and executive audiences while remaining precise enough for implementation. It is intended for: enterprise architects and security leaders, CTOs and engineering managers, CISOs and compliance officers, DevSecOps engineers, organizations operating offshore–onshore development teams, and auditors and compliance reviewers evaluating multinational delivery environments.

10.5 Final Statement

Secure offshore–onshore delivery is not achieved through trust alone, and it cannot rely on informal processes. It requires a defined architecture that integrates governance, identity control, pipeline integrity, auditability, and incident response into a single operating model.

The SODA Framework provides that architecture. By applying the layered structure described in this document, organizations can operate multinational software delivery environments with the same level of security, accountability, and reliability expected in enterprise systems.

SODA Framework v1.0 — Secure Offshore DevSecOps Architecture

sodaframework.org

References

[1] National Institute of Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations.* NIST Special Publication 800-53, Revision 5. Gaithersburg, MD: NIST, 2020.

[2] International Organization for Standardization. *Information Security Management Systems — Requirements.* ISO/IEC 27001:2022. Geneva: ISO, 2022.

[3] American Institute of Certified Public Accountants. *Trust Services Criteria for Security, Availability, Processing Integrity, Confidentiality, and Privacy.* SOC 2 TSC. New York: AICPA, 2022.

[4] OWASP Foundation. *OWASP Top Ten: The Ten Most Critical Web Application Security Risks.* Bel Air, MD: OWASP Foundation, 2021. https://owasp.org/Top10

[5] National Institute of Standards and Technology. *Zero Trust Architecture.* NIST Special Publication 800-207. Gaithersburg, MD: NIST, 2020.

[6] National Institute of Standards and Technology. *Computer Security Incident Handling Guide.* NIST Special Publication 800-61, Revision 2. Gaithersburg, MD: NIST, 2012.

Disclaimer

This document describes the SODA Framework v1.0, a conceptual security architecture model for offshore–onshore DevSecOps environments.

The framework is intended for educational, architectural, and advisory purposes. It does not represent a certification standard, regulatory requirement, or formal compliance program.

Organizations adopting the framework should evaluate their own legal, regulatory, and security requirements before implementation.