NYCPS TMS: Prescriptive Security & Compliance Strategy

I. Introduction: The Security & Compliance Imperative

This document mandates the comprehensive, hyper-detailed, and **non-negotiable** Security and Compliance Strategy for the NYCPS Transportation Management System (TMS) project. Given the system's critical function in transporting students, the extreme sensitivity of the data processed (including student PII protected under FERPA and NY Ed Law 2-d, and potentially health information), deployment within the regulated AWS GovCloud environment, and the stringent requirements imposed by NYC3, OTI, DIIT, and contractual obligations, security and compliance are the absolute top priorities, overriding all other concerns where conflicts arise.

Our approach is Zero Trust and Defense-in-Depth. We assume no implicit trust based on network location or system origin. Every user, device, and service interaction must be authenticated and explicitly authorized based on the principle of least privilege. Security controls are layered throughout the architecture, application stack, operational processes, and data lifecycle. Compliance is not treated as a checklist item but as an integral outcome of robust, verifiable security practices.

This strategy details the specific policies, procedures, technical controls, automation, governance, and ongoing vigilance required to build, operate, and maintain a TMS system that is demonstrably secure, resilient against threats (internal and external), and perpetually audit-ready. **Failure in any aspect of this strategy is unacceptable.**

> **Any security vulnerability or compliance failure could lead to severe consequences, including harm to students, data breaches triggering legal/financial penalties, loss of public trust, project termination, and significant reputational damage to NYCPS.**

II. Guiding Security & Compliance Principles (Mandatory)

Core Security & Compliance Principles:

- **Security & Compliance by Design/Default:** Controls and requirements *must* be integrated from the earliest** design stages and be the default configuration.

- Zero Trust Architecture:** Never trust, always verify. Authenticate and authorize every access request based on identity, device posture (where possible), and least privilege, regardless of network location.
- **Defense-in-Depth:** Implement multiple, layered security controls (Network, Host, Application, Data, Identity, Operations). A breach of one layer should not compromise the entire system.
- **Least Privilege Access:** Grant users, roles, and services *only* the minimum permissions necessary to perform their required functions. Regularly review and revoke unnecessary access.
- **Data Minimization & Confidentiality:** Collect, process, and retain only essential data. Classify data rigorously and apply encryption and access controls appropriate to its sensitivity (especially PII).
- **Immutable Infrastructure & Automation:** Utilize Infrastructure as Code (IaC) and automated pipelines to build, deploy, and manage infrastructure and applications consistently and securely, reducing manual errors and configuration drift. Treat infrastructure components as ephemeral where possible.
- **Continuous Monitoring & Rapid Response:** Implement comprehensive monitoring to detect security threats and compliance deviations in near real-time. Have

practiced incident response plans ready for immediate execution.

- **Verifiable Compliance:** All controls and processes *must* generate auditable evidence (logs, configuration records, test results, attestations) to demonstrate compliance with all applicable mandates (FERPA, 2-d, NYC3, etc.).
- **Secure SDLC Integration:** Embed security activities (threat modeling, SAST, DAST, SCA, security reviews) throughout the entire software development lifecycle ("Shift Left").
- **Shared Responsibility (Explicit):** While the Security Team leads strategy and provides expertise, *everyone* involved in the project (Dev, Ops, QA, PM, Users) shares responsibility for adhering to security policies and practices.

III. Security & Compliance Governance Structure

Effective security and compliance require dedicated roles and clear governance integrated with project oversight.

Implementation How-To:

1. Establish/Leverage a formal **Security Review Board (SRB)** comprising:
    - NYCPS CISO / Designee (Chair)
    - NYCPS CPO / Designee
    - Project Security Lead (Vendor)
    - Lead Cloud Architect (Vendor)
    - DIIT Security Representative
    - NYC3 Representative (as required/invited)
    - Legal Counsel Liaison (as required)
    - Project Manager (Facilitator/Scribe)

    The SRB meets regularly (e.g., monthly and ad-hoc) to review/approve security architecture, threat models, significant control implementations, risk assessments, penetration test results, compliance reports, and major security incident post-mortems. Decisions are formally documented.

2. Define explicit security responsibilities within project roles (beyond those listed in the HR Plan):

- **Project Security Lead (Vendor):** Owns implementation of security strategy, coordinates security team activities, primary security POC, reports security status/risks, ensures DevSecOps integration.

- **Application Security (AppSec) Engineer(s):** Perform secure code reviews, manage SAST/DAST/SCA tools, provide secure coding training/guidance, assist development teams with vulnerability remediation.

- **Cloud Security Engineer(s) (DevSecOps):** Implement/manage security controls in AWS GovCloud (IAM, SGs, WAF, KMS, Config, GuardDuty, Security Hub), automate security checks in IaC/CI/CD, respond to cloud security alerts.

- **Compliance Analyst:** Maintains CRTM, tracks compliance evidence, coordinates audit responses, monitors regulatory changes.

- **All Developers:** Responsible for writing secure code according to standards, writing tests for security features, remediating vulnerabilities identified in their code.
- **All SRE/Ops:** Responsible for secure configuration/patching of infrastructure, monitoring security alerts, participating in incident response.

3. Integrate security reviews and compliance checks as mandatory quality gates within the overall SDLC and Change Management processes.

Responsibility: Project Leadership, NYCPS CISO, CPO, Legal, Security Team Lead.

IV. Compliance Framework

Implementation

We will implement a proactive framework to ensure and demonstrate adherence to all relevant mandates.

Implementation How-To:

1. ****Maintain Comprehensive CRTM:** Continuously update the Compliance Requirements Traceability Matrix (CRTM in Confluence) mapping FERPA, NY Ed Law 2-d, CIPA, HIPAA (GovCloud context), WCAG 2.0 AA, NYC3/OTI/DIIT policies, and contractual security/privacy clauses to specific technical controls, operational procedures, test cases, and evidence artifacts (logs, reports, configurations). Assign owners for verifying each mapping.**
2. ****Policy Development & Alignment:** Ensure the TMS Data Governance Policy, Incident Response Plans (SIRP & general), BCP, and other project policies are explicitly aligned with and reference applicable regulations and NYCPS standards. Obtain necessary CPO/CISO/Legal approvals.**
3. ****Automated Compliance Checks:****

- Leverage **AWS Config** with managed and custom rules (aligned with FedRAMP High / NIST 800-53 baselines suitable for GovCloud, plus specific NYCPS requirements) to continuously monitor infrastructure configuration compliance (e.g., encryption enabled, public access blocked, logging enabled).

- Utilize **AWS Security Hub** to aggregate findings from Config, GuardDuty, Inspector, Macie, and potentially partner tools against specific compliance frameworks (e.g., CIS Benchmarks, NIST).

- Implement **automated checks within CI/CD pipelines** (e.g., `tfsec`, `checkov` for IaC; linters for policy adherence in code) to catch potential compliance issues before deployment.

- Configure alerts for critical compliance deviations reported by Config/Security Hub.

4. **Regular Compliance Reviews:** Conduct quarterly internal reviews led by the Compliance Analyst/Officer, using the CRTM and automated tool reports to verify control effectiveness and evidence completeness. Report findings to project leadership and SRB.

5. **Mandatory Compliance Training:** Implement role-based training covering general privacy/security awareness (all personnel) and specific requirements (FERPA/2-d for data handlers, secure coding for developers, WCAG for frontend devs/designers, incident reporting for all). Track completion rigorously.

Responsibility: Compliance Analyst, Security Team, DevOps Team (Automation), PM, Training Lead.

> ***Proactive, automated monitoring and meticulous evidence management via the CRTM are key to demonstrating continuous compliance.***

V. Embedding Security Throughout the DevSecOps Lifecycle (Prescriptive)

"Shift Left" Security: Security activities are integrated into every phase, not delayed until the end.

1. Phase 1: Planning & Requirements
    - Mandatory Activity: Identify high-level security & compliance requirements (privacy, availability, integrity needs based on data classification). Incorporate into NFRs.
    - Mandatory Activity: Conduct initial Security Risk Assessment; add security risks to project Risk Register.
    - Mandatory Activity: Define data classification for key data elements anticipated.

    Gate: Security/Compliance NFRs included in requirements baseline.

2. Phase 2: Architecture & Design
    - Mandatory Activity: Conduct detailed **Threat Modeling** (STRIDE/equivalent) for all new services/components and critical workflows. Document threats and required mitigations.

    - **Mandatory Activity:** Design specific security controls mapped to threats and compliance requirements (AuthN/Z, Encryption, Logging, Input Validation, Network Segmentation).
    - **Mandatory Activity:** Define secure configurations for AWS services (IAM policies, SG rules, KMS usage).
    - **Mandatory Activity:** Design secure API contracts.
    - **Mandatory Activity:** Include security considerations in all Architecture Decision Records (ADRs).

   Gate: Security architecture & design formally reviewed and approved by Security Review Board (SRB) / CISO designee. Threat model documented and reviewed.

3. Phase 3: Development (Implementation)
    - **Mandatory Activity:** Strict adherence to Secure Coding Standards (NYCPS, OWASP).
    - **Mandatory Activity:** Use approved, secure libraries and frameworks. Track dependencies.

- **Mandatory Activity:** Implement security controls as designed (AuthN/Z checks, input validation, output encoding, logging).
- **Mandatory Activity:** Write unit/integration tests specifically for security controls (e.g., testing access denial for incorrect roles).
- **Mandatory Activity:** Run linters and basic security checks via pre-commit hooks.
- **Mandatory Activity:** Conduct security-focused peer code reviews using checklists.

4. Phase 4: Testing & QA
    - **Mandatory Activity:** Execute automated SAST, SCA, Container Scanning in CI pipelines (block on critical/high findings).
    - **Mandatory Activity:** Execute automated DAST scans against QA/Staging environments.
    - **Mandatory Activity:** Develop and execute specific test cases validating security requirements and controls (AuthN/Z

rules, input validation effectiveness, error handling security).

- Mandatory Activity: Conduct formal Penetration Testing (internal/3rd party) against Staging environment before major releases.
- Mandatory Activity: Track and manage remediation of all identified security vulnerabilities based on severity SLAs.

Gate: All required security tests passed. No unresolved Critical/High vulnerabilities without documented risk acceptance from SRB/CISO.

5. Phase 5: Deployment & Release
    - Mandatory Activity: Deploy infrastructure using reviewed and approved IaC (Terraform) enforcing secure configurations (Security Groups, IAM, encryption).
    - Mandatory Activity: Use secure deployment strategies (Blue/Green, Canary) managed via CI/CD.
    - Mandatory Activity: Verify production environment configuration compliance using AWS Config Rules post-deployment.

- **Mandatory Activity:** Include security sign-off in the Production Readiness Review / Go/No-Go decision.
- **Mandatory Activity:** Securely manage secrets/credentials needed for deployment using Secrets Manager/Vault integration in CI/CD.

Gate: Security sign-off obtained for production release. Automated config checks pass.

6. Phase 6: Operations & Maintenance
    - **Mandatory Activity:** Continuous security monitoring (GuardDuty, Security Hub, CloudTrail, WAF logs, SIEM).
    - **Mandatory Activity:** Timely vulnerability patching according to defined SLAs.
    - **Mandatory Activity:** Regular IAM access reviews and least privilege enforcement.
    - **Mandatory Activity:** Execute Security Incident Response Plan (SIRP) when needed.
    - **Mandatory Activity:** Participate in periodic security audits and compliance reviews.

VI. Detailed Security Control Domains & Implementation

This section details the specific technical and procedural controls implemented across key security domains.

A. Identity & Access Management (IAM) - Zero Trust Foundation

Implementation How-To:

1. **Centralized Identity (Human Users): Federate using SAML/OIDC with NYCPS primary IdP (Azure AD/ADFS/NYCSA) for staff/parents/students where feasible, mapping IdP groups/attributes to TMS roles. Use AWS IAM Identity Center for streamlined federation setup if possible in GovCloud. For users without federation (SBC staff), use AWS Cognito User Pools with mandatory MFA (TOTP Authenticator App preferred over SMS) and strong password policies.**

2. **Machine/Service Identity (IAM Roles):** *Mandate* use of IAM Roles for *all* AWS service access (Lambda Execution Roles, ECS Task Roles, EC2 Instance Profiles). **Strictly prohibit use of long-lived IAM user access keys** for applications/services.

3. **Least Privilege Policies:** Craft fine-grained IAM policies attached to roles/users. Grant only the specific actions (`iam:Action`) needed on specific resources (`Resource ARN`). Use condition keys (`iam:Condition`) extensively to restrict access based on source IP, VPC endpoint, time of day, tags, MFA status, etc. Regularly review policies using IAM Access Analyzer.

4. **Role-Based Access Control (RBAC) in Application:** Implement application-level authorization checks based on user roles derived from IdP groups/Cognito groups or internal TMS role assignments, enforcing access to specific APIs and data scopes.

5. **Secrets Management:** Store *all* secrets (DB passwords, API keys, certs) securely in **AWS Secrets Manager**. Configure automatic rotation where supported. Grant access to secrets via least-privilege IAM policies attached to service roles. Applications

retrieve secrets at runtime via SDK calls. **No secrets in code, config files, or environment variables.**

6. **Access Reviews:** Implement mandatory quarterly user access reviews. Managers/Data Stewards review and recertify access for their direct reports/data domains. Automate reporting of current access lists. Revoke unnecessary access promptly via defined offboarding process.

7. **Auditing:** Log all authentication attempts (success/failure), authorization failures, IAM policy changes (CloudTrail), and Secrets Manager access (CloudTrail). Alert on suspicious IAM activity (privilege escalation, root usage).

Tools: AWS IAM (Roles, Policies, Identity Center), AWS Cognito, AWS Secrets Manager, AWS KMS, SAML/OIDC IdPs (NYCPS), Terraform (for IaC).

Responsibility: Security Team (Policy/Design/Audit), DevOps (IaC Implementation), Developers (App RBAC/Secret Retrieval).

B. Network Security (Layered Defense)

Implementation How-To:

1. **VPC Design:** Use multiple VPCs (Prod/NonProd). Implement multi-AZ design with distinct public and private subnets. Place all sensitive resources (databases, backend compute) in **private subnets** with no direct internet ingress/egress.

2. **Network Segmentation:**
    - **Security Groups (Stateful):** Define granular SGs per application tier/service. Allow ingress *only* from specific source SGs on specific ports (e.g., DB SG allows port 5432 only from App Tier SG). Default deny all ingress. Egress rules should also be restricted to only necessary destinations where feasible (e.g., VPC endpoints, specific external APIs via NAT GW). Manage SGs via Terraform.

    - **Network ACLs (Stateless):** Use as a secondary, broader layer of defense. Define NACL rules allowing necessary traffic between subnets but denying common malicious traffic types. Default deny implicit.

3. **Edge Security:**
    - Deploy **AWS WAF** with ALBs and CloudFront distributions. Use AWS Managed Rulesets (Core rules, SQLi/XSS prevention, Bot Control) and develop custom rules specific to TMS application vulnerabilities or traffic patterns. Configure WAF to block or log suspicious requests.
    - Utilize **AWS Shield Advanced** for enhanced DDoS protection on critical public endpoints (CloudFront, ALBs, Route 53). Configure proactive engagement with AWS DDoS Response Team (DRT).
4. **Private Connectivity:** Use **VPC Endpoints (Interface & Gateway)** extensively for accessing AWS services (S3, DynamoDB, KMS, Secrets Mgr, ECR, SQS, SNS, etc.) privately

from within the VPC without traversing the internet.

5. **Traffic Flow Logging:** Enable **VPC Flow Logs** (potentially sampled or targeted at critical subnets due to volume/cost) delivered to CloudWatch Logs or S3 for network traffic analysis and security incident investigation.

6. **Egress Control:** Route outbound internet traffic from private subnets through **NAT Gateways**. Consider implementing **AWS Network Firewall** or proxy solutions for more granular egress filtering and inspection if required by compliance/security policy.

Tools: AWS VPC, Subnets, Security Groups, Network ACLs, NAT Gateway, Internet Gateway, VPC Endpoints, AWS WAF, AWS Shield Advanced, AWS Network Firewall (optional), Route 53, CloudTrail, VPC Flow Logs, Terraform.

Responsibility: Cloud Architect, Network Security Engineer, DevOps Team (IaC), Security Team (WAF/Firewall Rules).

C. Data Security & Privacy Controls

Implementation How-To:

1. **Encryption Everywhere:**

    - **At Rest:** Mandate SSE-KMS with dedicated, environment-specific CMKs (with key rotation enabled) for S3 buckets, EBS volumes, RDS instances/snapshots, DynamoDB tables, ElastiCache (where available), SQS/SNS queues containing Confidential/Highly Restricted data. Enforce via IaC and AWS Config rules.

    - **In Transit:** Mandate TLS 1.2+ with strong ciphers for *all* communication (external user -> AWS Edge, Edge -> Internal Services, Service-to-Service, Service -> Database/Cache/Queue). Configure ALBs, API Gateway, CloudFront, application clients accordingly.

2. **Data Classification Enforcement:** Technical controls must reflect data classification (from Data Governance Policy/Matrix). E.g., Highly Restricted data requires CMK encryption, more

stringent access logging, potentially masking in more scenarios.

3. **PII Handling:** Implement specific procedures for handling PII according to FERPA/NY Ed Law 2-d: Strict access control (RBAC/ABAC), purpose limitation enforcement in code, secure logging (avoid logging PII directly), mandatory masking/anonymization in non-prod, verifiable destruction after retention period.

4. **Data Masking/Anonymization:** Implement automated, robust masking/anonymization pipelines (Glue, Lambda) using approved techniques for populating non-prod environments. Verification of effectiveness is mandatory.

5. **Data Loss Prevention (DLP):** Explore using **AWS Macie** to automatically discover and classify sensitive data (PII) in S3 buckets and alert on unusual access patterns or potential data leakage. Configure Macie jobs and alerts appropriately.

6. **Secure Backup/Restore:** Ensure backups are encrypted with the same (or stronger) keys as source data. Restrict access to backup/restore functions via IAM. Regularly

test restore procedures *and* validate data integrity/security post-restore.

7. **Secure Destruction:** Implement automated S3 Lifecycle expiration policies for 7-year deletion. Implement verified, logged procedures for deleting data from active databases/EBS if needed before retention expiry (requires formal approval). Obtain/generate Certificates of Destruction.

Tools: AWS KMS, AWS Secrets Manager, AWS S3 (SSE, Lifecycle, Policies), AWS RDS (Encryption, Audit Logging), AWS DynamoDB (Encryption, PITR), AWS EBS (Encryption), TLS Certificates (ACM), AWS Config, AWS Macie (optional), AWS Backup, Terraform, Application Code (Masking logic).

Responsibility: Security Team (Encryption/Key Policy), DevOps (IaC Config), Developers (PII Handling/Masking Logic), DBA (DB Security/Backup), Compliance Officer (Verification).

Data security controls must directly map to and verifiably enforce the TMS Data Governance Policy and all regulatory requirements (FERPA, 2-d).

D. Application Security (AppSec)

Implementation How-To:

1. **Secure SDLC Integration:** Embed SAST, SCA, DAST, Container Scanning, Threat Modeling, Secure Code Reviews throughout the development lifecycle (as detailed in SDLC/DevSecOps sections).

2. **Input Validation:** Mandate rigorous server-side validation of *all* inputs (APIs, UI forms, file uploads) against allow-lists for type, length, format, range using framework features and custom logic. Sanitize data before passing to interpreters (SQL, OS, etc.).

3. **Output Encoding:** Mandate contextual output encoding for all data reflected back to users (HTML, JavaScript) or used in downstream systems (e.g., generating reports) to prevent XSS. Use framework features (React's default JSX encoding, template engine escaping) and specific encoding libraries where needed.

4. **Authentication/Authorization Implementation:** Securely implement AuthN/Z checks server-side for every request to protected resources, verifying tokens/sessions and checking permissions/scope against defined RBAC/ABAC rules.

5. **API Security:** Use API Gateway for public/internal APIs. Implement rate limiting, throttling, request validation (based on OpenAPI spec), and appropriate authorizers (Lambda Authorizer for custom logic, Cognito Authorizer, IAM Authorizer).

6. **Dependency Management:** Continuously monitor third-party libraries (SCA) and promptly update components with known critical/high vulnerabilities. Maintain a Software Bill of Materials (SBOM).

7. **Secure Session Management:** Use secure, HttpOnly, SameSite cookies; short timeouts; regenerate session IDs on login; implement secure logout.

8. **Error Handling:** Implement generic error messages for users; log detailed error information (excluding PII) securely on the backend for debugging.

Tools: GitLab CI (SAST, DAST, SCA, Container Scanning), SonarQube, OWASP ZAP, Snyk, Code Review Checklists, API Gateway, Secure Coding Libraries/Framework Features.

Responsibility: Developers (Implementation), AppSec Engineers (Reviews, Tooling, Guidance), QA (Testing Controls).

E. Infrastructure Security (Host, Container, Serverless)

Implementation How-To:

1. **Hardening:** Use hardened base Operating System images (e.g., CIS Benchmarked AMIs for EC2, minimal base images like distroless or Alpine for containers). Remove unnecessary packages/services. Configure OS security settings (firewall, auditd).
2. **Vulnerability Management:** Use AWS Inspector (for EC2) and ECR Scanning/Trivy (for containers) to continuously scan for OS/library vulnerabilities. Integrate results with patching process/CI quality gates.
3. **Patch Management:** Implement automated patching for OS using AWS Systems Manager Patch Manager for EC2 instances. For containers, mandate rebuilding images regularly with updated base images/dependencies via CI/CD pipeline. Track patching compliance against SLAs.
4. **Configuration Management:** Use AWS Config rules to monitor infrastructure

configurations against defined security baselines (e.g., EBS encryption enabled, S3 public access blocked, specific ports closed in SGs). Alert on non-compliance. Use Terraform/IaC to enforce desired state.

5. **Container Security:**
    - Scan images for vulnerabilities before deployment.
    - Run containers as non-root users.
    - Use read-only file systems where possible.
    - Configure minimal resource limits (CPU/Memory).
    - Use Security Groups and potentially network policies (in EKS) for network segmentation.
    - Monitor container runtime behavior (e.g., using GuardDuty EKS Runtime Monitoring or tools like Falco).
6. **Serverless Security (Lambda):**

- **Assign least-privilege IAM execution roles per function.**
- **Store secrets in Secrets Manager, access via IAM role (never in environment variables directly unless encrypted with KMS).**
- **Validate input event schemas.**
- **Include dependencies securely; scan deployment packages for vulnerabilities (SCA).**
- **Configure appropriate memory/timeout limits to prevent abuse.**
- **Monitor execution logs and errors via CloudWatch.**

Tools: AWS Inspector, AWS Systems Manager (Patch Manager, State Manager), ECR Scanning, Trivy/Clair, AWS Config, GuardDuty, Terraform, Docker Security Best Practices, Lambda Best Practices.

Responsibility: SRE/Ops Team, DevOps Team (IaC/Patch Automation), Security Team (Scanning/Policy), Developers (Container/Lambda config).

VII. Security Operations (SecOps) & Incident Response

This details the processes for continuously monitoring the security posture and responding rapidly and effectively to potential security incidents.

Implementation How-To:

1. **Centralized Security Monitoring & SIEM:**
    - Aggregate logs (CloudTrail, VPC Flow Logs, WAF, GuardDuty findings, Security Hub findings, critical application security logs) into a dedicated SIEM (e.g., AWS OpenSearch Service with Security Analytics, Splunk).
    - Develop correlation rules and threat detection logic within the SIEM specific to TMS threats and compliance requirements.

- Create security-focused dashboards for visualizing security posture, active threats, and compliance status.

Responsibility: Security Team (SecOps).

2. **Security Alerting:**
   - Configure high-fidelity alerts based on SIEM correlations, GuardDuty/Security Hub critical findings, critical AWS Config non-compliance, and key application security events (e.g., repeated AuthZ failures, potential injection attempts detected by WAF/App logs).
   - Route critical security alerts *immediately* to the Security Team on-call rotation via PagerDuty/Opsgenie.
   - Route lower-priority security notifications to dedicated Slack/Teams channels.

Responsibility: Security Team, SRE/Ops (Alerting Infra).

3. **Security Incident Response Plan (SIRP):**

    - Maintain and regularly rehearse (via tabletop exercises) the dedicated SIRP covering specific threat scenarios (Malware, Ransomware, DDoS, Unauthorized Access, Insider Threat, Data Breach/Exfiltration).

    - SIRP *must* include phases: Preparation -> Detection & Analysis -> Containment -> Eradication -> Recovery -> Post-Incident Activity (RCA, Lessons Learned).

    - Define clear roles and responsibilities during a security incident (Security Incident Lead, Forensics Lead, Comms Lead, Legal Liaison, etc.).

    - Mandate immediate engagement protocols between Security, SRE/Ops, Legal, Comms, and NYCPS Leadership (CISO, CPO) based on incident severity/type, especially for potential data breaches.

- **Include specific procedures for evidence preservation and forensic analysis (leveraging CloudTrail, logs, snapshots).**
- **Integrate SIRP communication steps with overall Incident Management and BCP communication plans.**

Responsibility: Security Team (Owner/Lead), SRE/Ops, Legal, Comms, CISO/CPO.

> ***Having a well-defined and practiced SIRP is critical for meeting data breach notification requirements (e.g., under NY Ed Law 2-d) and minimizing incident impact.***

VIII. Vendor & Third-Party Security Management

Security risks extend to the entire supply chain. We will enforce strict security requirements for all external parties interacting with TMS data or systems.

> *Implementation How-To:*
>
> 1. **Mandatory Pre-Contract Due Diligence:** Perform rigorous security assessments (reviewing policies, certifications like SOC2/ISO27001, questionnaires, potentially vulnerability scans/interviews) for *all* vendors handling TMS data or providing critical system components *before* contract signing. Document findings and required remediations.
> 2. **Contractual Security Requirements:** Ensure contracts include mandatory clauses requiring adherence to NYCPS security/privacy policies (flow-down), specific security controls (e.g., encryption), audit rights, breach notification obligations (within 24 hours), and secure offboarding/data destruction requirements.
> 3. **Least Privilege Access for Vendors:** Grant vendor personnel/systems only the absolute minimum necessary access to NYCPS environments or data, using time-bound, audited mechanisms (e.g., temporary IAM roles, restricted VPN access).

4. **Ongoing Monitoring (Where Applicable):** For critical SaaS vendors, periodically review their compliance documentation (updated SOC2 reports) and security posture. Monitor logs for anomalous vendor access patterns.

5. **Secure Offboarding:** Rigorously execute the access revocation checklist for all vendor personnel/systems immediately upon contract termination or personnel departure. Ensure data destruction certification is received.

Responsibility: Security Team (Assessments), Contract Manager, Legal (Contracts), PM/VRM (Ongoing Monitoring), IT/Ops (Access Control).

IX. Security Automation & Automated Guardrails

Automation is key to consistently enforcing security policies and providing rapid feedback.

> *We will automate security checks and controls wherever feasible to reduce manual effort and ensure consistent enforcement.*

Implementation How-To:

1. **CI/CD Security Gates:** Automate SAST, SCA, Container Scanning, and potentially basic DAST checks within GitLab CI/CD pipelines, configured to *fail the build* based on severity thresholds.

2. **IaC Security Validation:** Integrate tools like `tfsec`, `Checkov`, or `CloudFormation Guard` into CI pipelines to scan Terraform/CloudFormation code for security misconfigurations *before* deployment. Fail builds on critical findings.

3. **AWS Config Rules (Preventive & Detective):** Deploy AWS Config rules (managed and custom) to continuously monitor resource configurations against security baselines (e.g., check for unencrypted volumes, public S3 buckets, permissive SGs). Configure *auto-remediation* actions (via Systems Manager Automation) for specific, low-risk violations where appropriate and approved.

4. **Automated Alerting:** Configure CloudWatch Alarms/EventBridge rules based on Security Hub findings, GuardDuty alerts, Config non-compliance,

and critical security logs to trigger automated notifications or incident response workflows.

5. **IAM Policy Validation:** Use IAM Access Analyzer to continuously review IAM policies for unintended external or cross-account access and alert on findings.

6. **Patch Management Automation:** Utilize AWS Systems Manager Patch Manager to automate OS patching schedules and compliance reporting for EC2 instances.

Responsibility: DevOps Team, Security Team, SRE/Ops Team.

X. Security Training & Awareness Program

Technology and processes are only effective if personnel understand their security responsibilities.

Implementation How-To:

1. **Mandatory Onboarding Training:** All project personnel (including vendor staff) *must* complete initial training covering: NYCPS Acceptable Use Policy, Data Classification, PII Handling (FERPA/2-d focus), Phishing Awareness, Secure Password Practices, Incident Reporting Procedures. Track completion.

2. **Role-Based Security Training:** Provide specific, mandatory training based on roles:
    - **Developers:** Secure Coding Practices (OWASP Top 10, language specifics), Threat Modeling basics, Using security tools in CI/CD.

    - **Ops/SRE/DevOps:** Secure AWS Configuration (IAM, Networking, KMS), Secure IaC practices, Incident Response procedures, Log analysis basics.

    - **QA:** Security testing concepts, Identifying common vulnerabilities, Testing security controls.

    - **PMs/POs/BAs:** Understanding security requirements, Risk management related to security, Secure design principles overview.

- **Support Staff:** Recognizing security incidents, Secure handling of user data, Escalation procedures.

3. **Annual Refresher Training:** Mandate annual completion of core security awareness and relevant role-based training modules.

4. **Phishing Simulations:** Conduct periodic simulated phishing campaigns to test and reinforce awareness.

5. **Security Champions Program:** Identify security-minded individuals within development teams to act as local advocates, mentors, and liaisons with the central security team.

6. **Documentation:** Make security policies, standards, and training materials easily accessible via Confluence.

Responsibility: Security Team (Content/Delivery), Training Lead (Logistics/Tracking), Managers/Leads (Ensuring team completion).

XI. Compliance Audit Readiness (Security Focus)

We maintain a state of perpetual readiness for security-focused audits.

> ***Implementation How-To:***
>
> 1. ****Evidence Collection Automation:** Leverage AWS Config, Security Hub, CloudTrail, GuardDuty, IAM Access Analyzer, and GitLab CI/CD logs/reports as primary sources of automated evidence for control effectiveness. Configure delivery to central logging/S3 archives.**
> 2. ****CRTM for Security Controls:** Ensure the Compliance Requirements Traceability Matrix explicitly maps security regulations/policies (NYC3, FERPA, 2-d) to specific technical controls (KMS keys, SG rules, WAF rules, IAM policies, SAST checks, etc.) and links to automated evidence sources or manual test results.**
> 3. ****Regular Internal Security Audits:** Conduct quarterly internal audits focusing on specific security domains (e.g., IAM configuration, Network segmentation, Vulnerability management process, Incident Response readiness) using the CRTM and automated tool outputs. Track findings and remediation.**

4. **Documentation Repository:** Maintain the organized Confluence space containing all security policies, standards, architecture diagrams, threat models, security test reports (pen tests, scan summaries), incident post-mortems, training records, and internal/external audit reports/responses.

5. **Audit Response Playbook:** Maintain a specific playbook for responding to external security audits, detailing roles (Audit Liaison, SMEs), evidence gathering process, communication protocols, and review/approval steps for responses.

Responsibility: Security Team, Compliance Analyst, PM, DevOps/SRE (Evidence generation).

XII. Conclusion: Building a Foundation of Trust

This Prescriptive Security & Compliance Strategy establishes an uncompromising framework designed to protect sensitive student data, ensure regulatory adherence, and build operational resilience for the NYCPS TMS. By embedding security into every phase of the

lifecycle ("Shift Left"), leveraging extensive automation and AWS GovCloud's capabilities, implementing defense-in-depth across all layers, mandating rigorous testing and validation, and fostering a security-aware culture through continuous training and clear accountability, we create a demonstrably secure and compliant system.

The hyper-detailed controls, processes, governance structures, and focus on verifiable evidence outlined herein provide the necessary assurance for a critical public sector application handling PII. Adherence to this strategy is mandatory and fundamental to maintaining the trust of NYCPS, students, parents, and the public, ensuring the TMS project's long-term success and integrity.