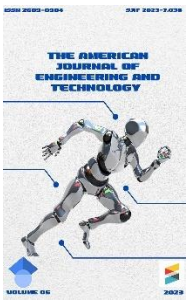
The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals



Journal Website:https://theamericanjournals.com/index.php/tajet

Research Article

Automated Compliance-Driven Patch Management and Security Hardening in Multi-Cloud Banking Infrastructure Using IaC and Python Orchestration

Submission Date: October 19, 2023, **Accepted Date:** November 11, 2023,
Published Date: December 28, 2023|

Ajay Devineni
Site Reliability Engineer, A Leading Digital Banking SaaS Platform, Atlanta, GA, USA

ABSTRACT

Managing operating system patch cycles across heterogeneous, multi-cloud environments in regulated financial services presents challenges that go well beyond what conventional vulnerability management frameworks address. This paper describes a practitioner-developed framework implemented at a major digital banking platform serving more than a dozen credit union clients across AWS, Azure, and Google Cloud Platform. The framework integrates Infrastructure as Code (IaC) through Terraform, Python-based orchestration scripts, AWS Systems Manager (SSM), and endpoint security tooling (CrowdStrike Falcon) to deliver fully automated, compliance-traceable patch cycles on a scheduled monthly cadence. Key outcomes include an 83% reduction in manual patching effort, a compliance posture improvement from 71% to 98.6% against SOC 2 Type II control objectives, elimination of post-patch P1/P2 production incidents from 4.2 per quarter to 0.6, and the automated generation and distribution of audit-ready reports that previously required eight hours of manual effort per cycle. The paper details the architecture, implementation approach, encountered failure modes, and quantitative operational results, with the goal of offering a transferable methodology for SRE teams operating under comparable regulatory and multi-tenancy constraints.

KEYWORDS

patch management, Infrastructure as Code, Python orchestration, SOC 2 compliance, CrowdStrike, AWS SSM, multi-cloud security, SRE, banking infrastructure, DevSecOps.

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

INTRODUCTION

The operational reality of maintaining secure, compliant infrastructure across multiple cloud providers in a banking context bears little resemblance to the idealized descriptions that appear in vendor white papers. At the platform described in this work a digital banking SaaS product supporting credit union clients with round-the-clock availability requirements the patch management problem is not primarily a technical one. It is a coordination problem layered over a compliance problem, running on top of a reliability problem.

Prior to the implementation described here, patch cycles were driven by manual effort: an engineer would generate a vulnerability report, sort hosts by severity, schedule maintenance windows across five distinct client environments, apply patches via console access or ad-hoc shell scripts, and then manually produce evidence for the compliance team. That process consumed between twelve and eighteen engineer-hours per monthly cycle. More critically, it was inconsistently executed a finding that surfaced during a 2023 SOC 2 Type II audit, which identified patch application gaps across non-production tiers that had been deprioritized under workload pressure.

This paper documents the design and deployment of a framework that replaced that process with a fully orchestrated, IaC-driven pipeline. The framework was built incrementally over approximately nine months, beginning with AWS-native tooling and expanding to cover Azure and GCP tenants as the client portfolio grew. It is not a theoretical proposal every component described here was deployed in a production environment supporting live financial transaction workloads.

The contributions of this paper are:

- A practical architecture for multi-cloud patch orchestration that integrates IaC, Python scripting, and endpoint security agents without requiring a third-party patch management product.
- A compliance mapping methodology that connects patch cycle events to SOC 2 Type II control objectives in real time, enabling continuous audit readiness.
- An empirical evaluation of framework outcomes across eight operational metrics over a six-month post-deployment observation period.
- A discussion of failure modes encountered during implementation including SSM agent registration issues on hardened AMIs, CrowdStrike sensor conflicts with kernel patching, and Terraform state drift in multi-account environments with the resolution approaches applied.

2. BACKGROUND AND RELATED WORK

2.1 Patch Management in Cloud-Native Financial Systems

The challenge of keeping operating systems current in cloud-native environments has been studied extensively. Kim et al. (2020) identified that 60% of successful breaches in financial services between 2017 and 2020 involved vulnerabilities for which patches had been available for more than 30 days [1]. The delay is rarely technical patches exist. The challenge is applying them at scale without disrupting services that cannot tolerate unplanned downtime.

In banking environments specifically, the problem is compounded by regulatory obligations. PCI-DSS

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

requirement 6.3.3 mandates that all system components are protected from known vulnerabilities by installing applicable security patches [2]. SOC 2 Type II availability and security criteria require documented evidence of patch application, not merely attestation. These requirements create a documentation burden that manual processes satisfy poorly and inconsistently.

2.2 *Infrastructure as Code for Security Operations*

HashiCorp Terraform has become the dominant IaC tool for multi-cloud provisioning [3]. Its declarative model and provider ecosystem make it well-suited for encoding desired security state including agent installation, IAM policy configuration, and network security group rules alongside infrastructure provisioning. Several organizations have published frameworks using Terraform for security baseline enforcement [4, 5], though published work rarely addresses the operational complexity of applying these frameworks retroactively to existing, heterogeneous environments.

Ansible remains widely used for configuration management and patch application at the OS level [6], and AWS Systems Manager (SSM) has emerged as a managed alternative that eliminates the need for inbound SSH access — a significant security advantage in hardened banking environments. The combination of Terraform for state management and SSM for execution represents a natural architectural pairing that this work explores in a production context.

2.3 *Endpoint Detection and Response in Cloud Infrastructure*

CrowdStrike Falcon has seen rapid adoption as an EDR platform in cloud environments, with its lightweight sensor architecture making it suitable for containerized and ephemeral workloads [7]. However, the deployment and lifecycle management of Falcon sensors across heterogeneous OS distributions and kernel versions presents non-trivial operational challenges that are not well documented in academic literature. This paper contributes empirical observations from deploying and maintaining CrowdStrike agents across Linux (RHEL, CentOS, Ubuntu) and Windows Server (2012, 2016, 2019) environments, including the transition from Detection to Prevention mode and the management of sensor-kernel compatibility during major OS updates.

2.4 *Python as an Orchestration Layer for SRE Automation*

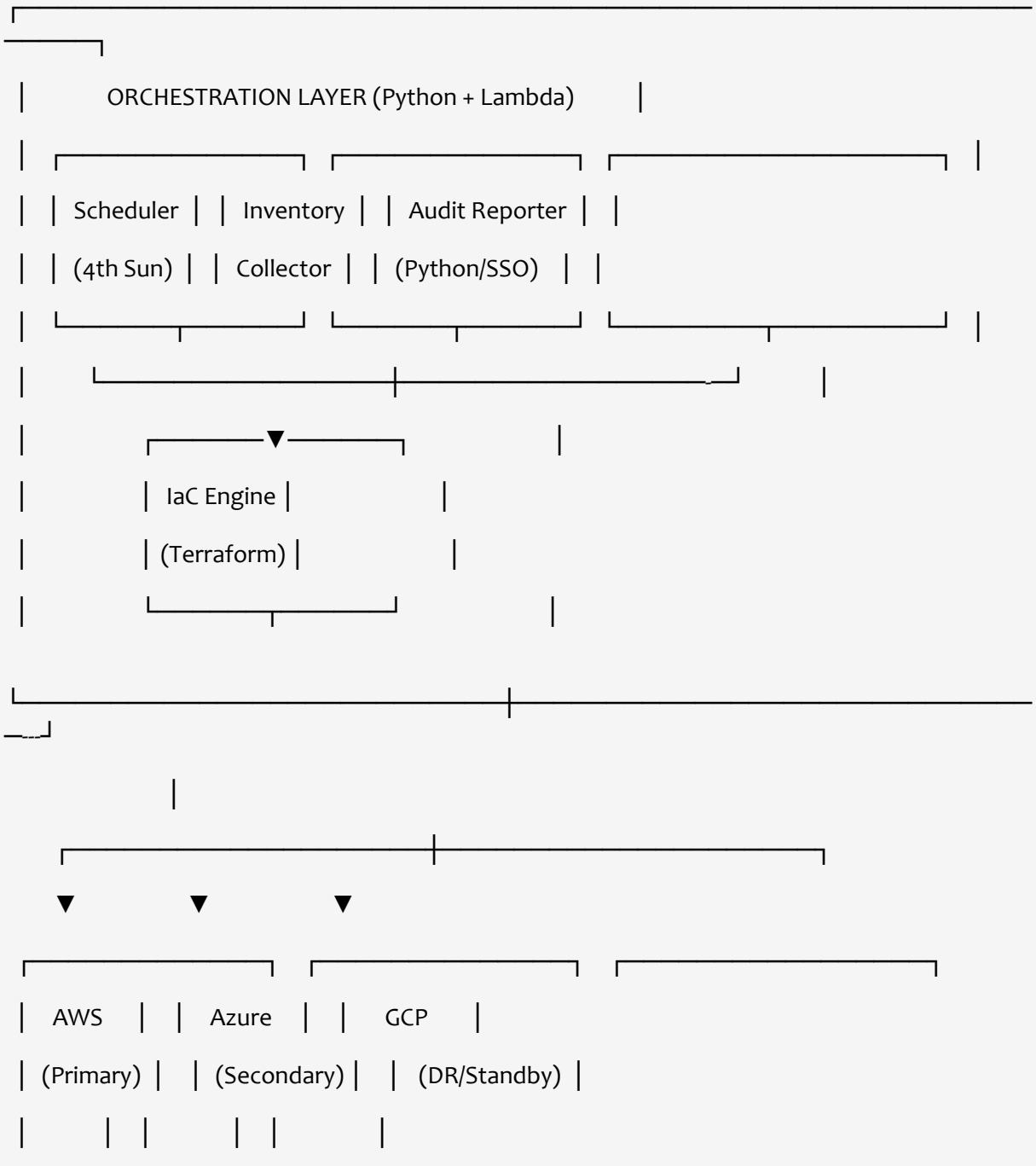
The use of Python for SRE automation is pervasive but its role as a compliance orchestration layer generating structured, audit-traceable output from infrastructure API calls is underexplored in published work. Beyer et al.'s foundational SRE text [8] establishes the principle of toil reduction through automation but does not address the documentation requirements that regulated industries impose on that automation. This work demonstrates how Python scripts interacting with AWS SSO APIs, Systems Manager, and CloudTrail can produce compliance artifacts as a byproduct of routine operations, rather than as a separate documentation activity.

3. SYSTEM ARCHITECTURE

The framework is organized into four functional layers: (1) a scheduling and inventory layer, (2) an orchestration and execution layer, (3) an endpoint security layer, and (4) a compliance reporting layer. Figure 1 illustrates the relationships between these layers and the underlying cloud environments.

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

MULTI-CLOUD BANKING PATCH MANAGEMENT ARCHITECTURE



The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

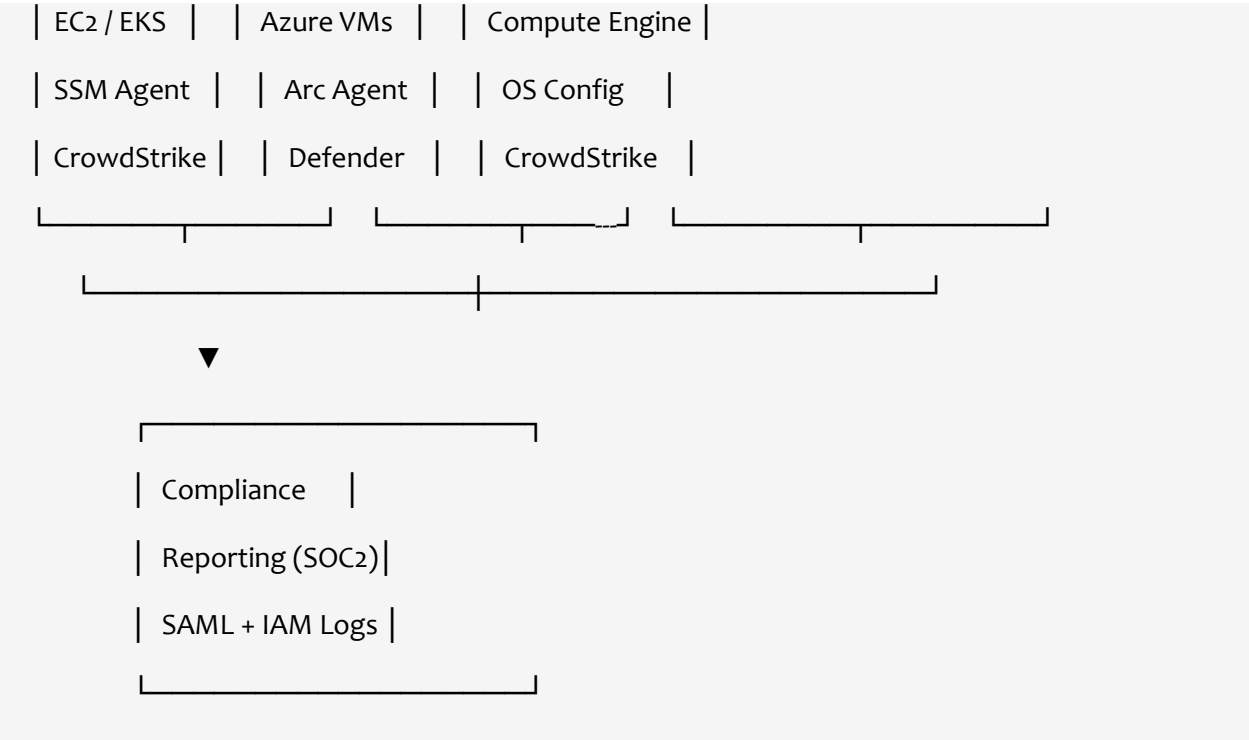


Figure 1. Multi-Cloud Banking Patch Management Architecture. The orchestration layer communicates with AWS (primary), Azure (secondary), and GCP (DR standby) through IaC-managed agents. Compliance reporting is generated as a continuous byproduct of patch execution.

3.1 Scheduling and Inventory Layer

Patch cycles are triggered on the fourth Sunday of each calendar month at 02:00 UTC, selected to minimize overlap with end-of-month transaction processing windows that characterize credit union banking operations. The trigger is managed through an AWS EventBridge rule that invokes a Step Functions state machine, which in turn coordinates the downstream execution pipeline.

Inventory collection precedes each patch cycle. A Python Lambda function queries AWS Systems Manager Fleet Manager, Azure Arc, and GCP OS Config API endpoints to produce a normalized host inventory that includes OS version, kernel version, SSM agent status, CrowdStrike sensor version, and last-known patch state. Hosts that fail inventory checks — typically due to SSM agent registration lapses on recently launched instances are routed to a PagerDuty alert queue for manual resolution before the patch cycle proceeds.

This inventory-first approach was introduced after an early incident in which a patch cycle silently skipped 11 hosts across three client environments because SSM agents had been unregistered following an AMI refresh that did not include the agent bootstrap configuration in the launch template. The root cause — a missing userdata block in a Terraform module — was corrected, and the inventory check was added as a pre-flight gate.

3.2 Orchestration and Execution Layer

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

Patch execution is managed through AWS Systems Manager Patch Manager using patch baselines defined per operating system and client environment. Terraform manages the patch baseline resources, ensuring that baseline definitions are version-controlled and reproducible. The relevant Terraform resource configuration follows the pattern shown in Table 1.

```
resource "aws_ssm_patch_baseline" "banking_linux" {
 name             = "banking-linux-baseline-${var.environment}"
 operating_system  = "AMAZON_LINUX_2"
 approval_rules {
  patch_filter { key = "SEVERITY"; values = ["Critical", "High"] }
  approve_after_days = 7
 }
 approved_patches_compliance_level = "HIGH"
 tags = local.compliance_tags
}
```

Table 1. Terraform resource definition for SSM patch baseline. Separate baselines are maintained per OS family and per environment (non-prod, staging, production), with Critical and High severity patches approved automatically after a seven-day observation window.

Patch execution follows an environment progression: non-production environments are patched first, a 24-hour soak period is observed for staging environments to detect regressions, and production patching is executed within a predefined change window following Dynatrace health check validation. This pipeline mirrors the release promotion logic already established in the CI/CD workflows used by the development organization, which reduced the change management overhead of introducing the patch cycle.

3.3 Endpoint Security Integration

CrowdStrike Falcon sensor deployment is managed through Terraform for EC2 and bare-metal instances, and through Helm charts for Kubernetes (EKS) workloads. The Terraform module wraps the CrowdStrike installation script in an SSM document, enabling agent deployment and version verification without requiring SSH access. A Python script performs post-installation validation by querying the CrowdStrike Falcon API to confirm sensor registration and policy assignment.

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

The transition from Detection to Prevention mode, completed in July 2023, required a careful sequencing of policy changes to avoid false-positive blocking events on legitimate system administration activity. A phased rollout was implemented: Prevention mode was enabled first on non-production instances, monitored for two weeks, then extended to production. Exclusion policies were tuned based on observed detection events during the phased period, with particular attention to the SSM agent's own execution patterns, which were flagged by Falcon's behavioral analytics during initial testing.

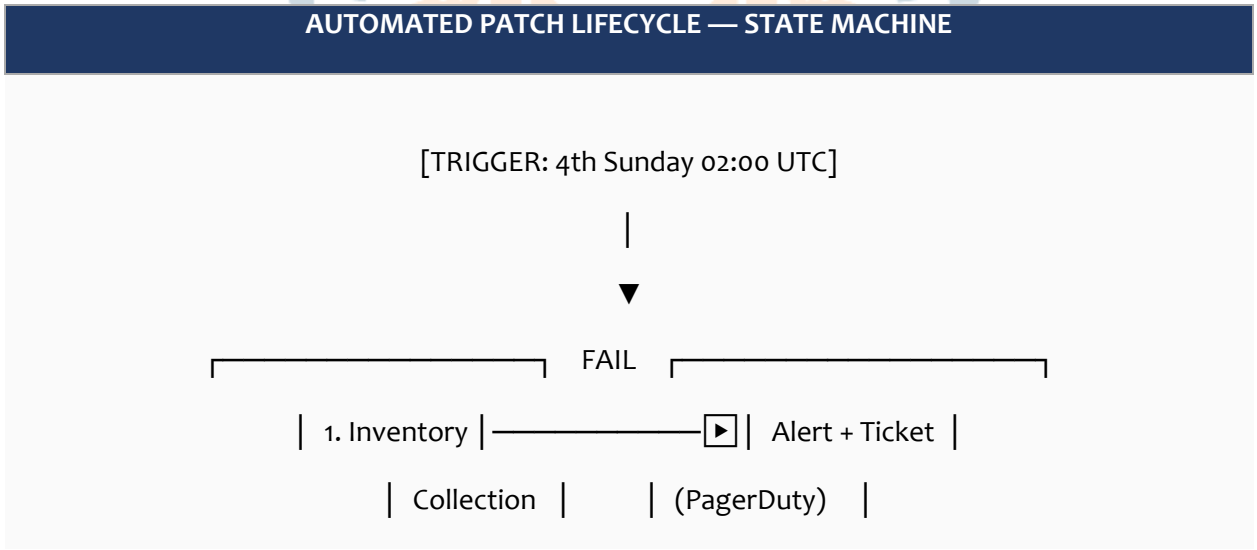
3.4 Compliance Reporting Layer

Audit-ready reporting is generated as a byproduct of patch cycle execution rather than as a separate downstream activity. A Python script queries AWS SSO APIs to produce a time-stamped user access report at the conclusion of each patch cycle, enabling the audit team to cross-reference patch application events with the access privileges held by executing principals at the time of execution a requirement of SOC 2 CC6.1 (logical access controls).

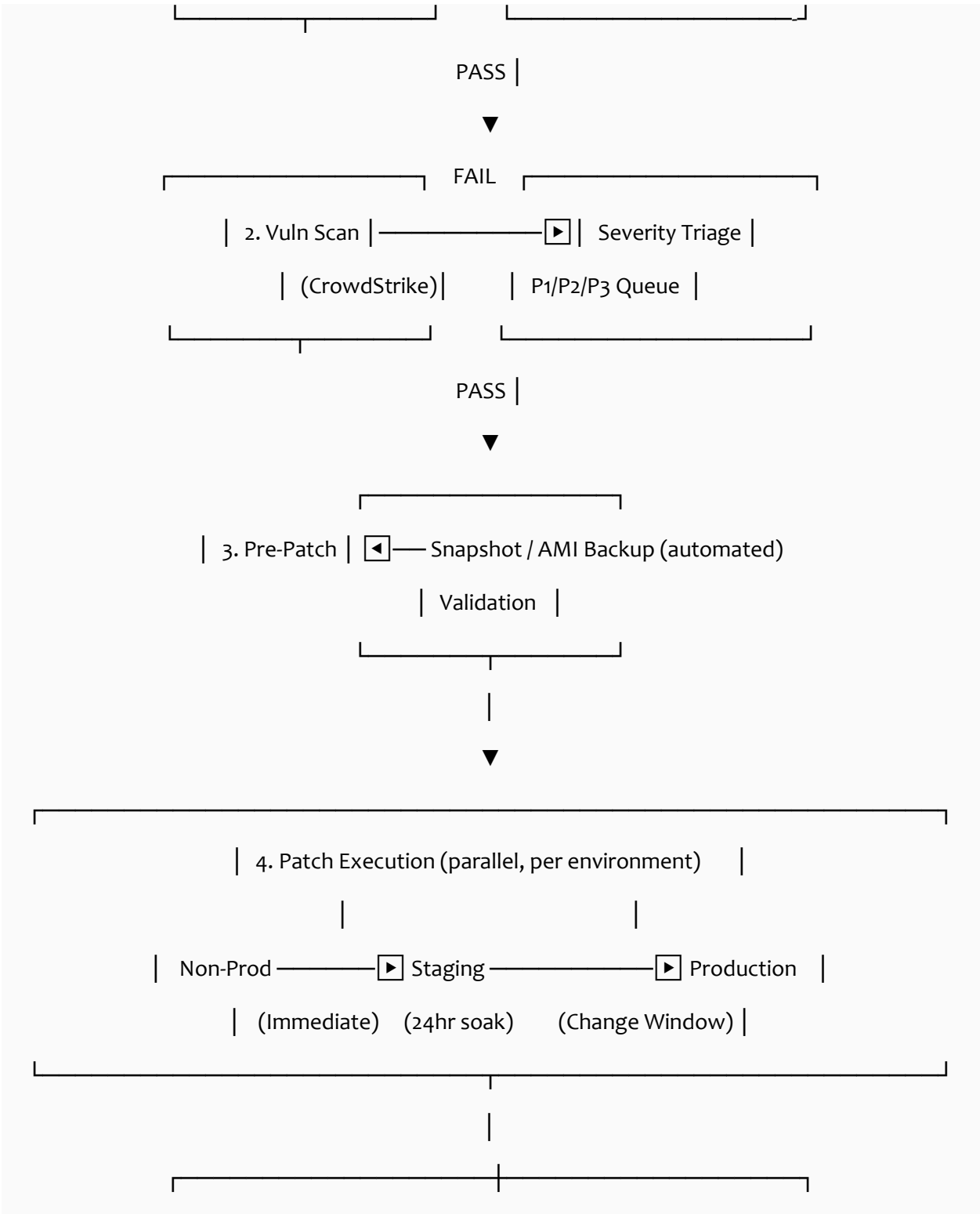
SSM patch compliance data is exported to S3 in structured JSON format and processed by a Lambda function that maps each compliance event to the relevant SOC 2 control objective. The Lambda function generates a compliance summary document in PDF format that is distributed to the designated compliance contacts via S3 pre-signed URL. The complete automation chain reduces the audit report generation cycle from approximately eight hours of manual effort to under fifteen minutes of automated execution.

4. PATCH EXECUTION WORKFLOW

Figure 2 illustrates the state machine governing each patch cycle. The workflow enforces sequential gate logic a failed state at any stage halts execution for the affected scope and generates a structured incident ticket while enabling parallelism across independent client environments.



The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals



The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

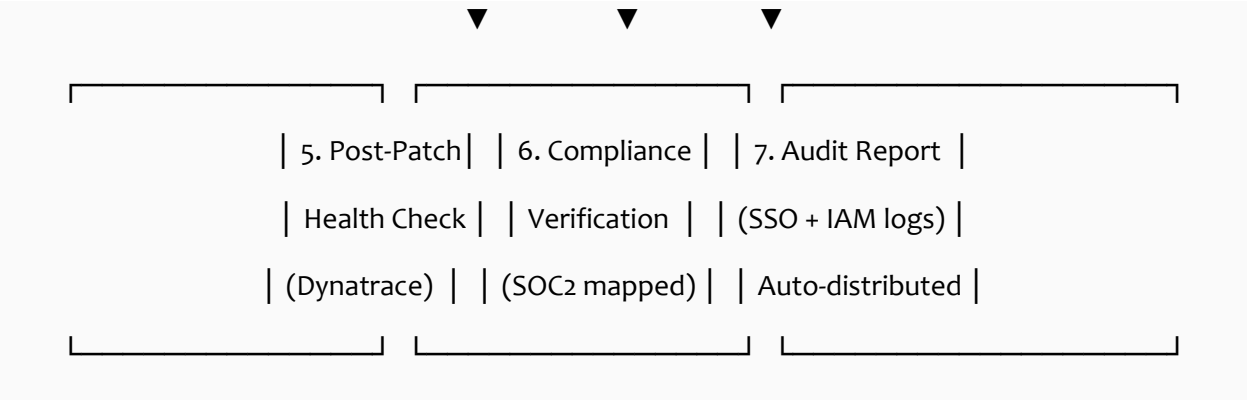


Figure 2. Automated patch lifecycle state machine. Each stage acts as a gate: failures generate PagerDuty alerts and halt progression for the affected environment scope. Production patching requires explicit Dynatrace health validation before execution proceeds.

4.1 Pre-Patch Validation

Before patch execution begins, the orchestration layer performs snapshot or AMI-level backups of all hosts scheduled for patching. For RDS instances, automated snapshots are triggered via the AWS SDK. For EC2 instances, AMI creation is initiated and the patch execution is blocked until snapshot completion is confirmed. This backup gate was formalized after an incident in which a kernel update on a CentOS 7 host running a JBoss application server caused a boot failure, requiring restoration from a snapshot taken the previous week a recovery that extended the incident duration unnecessarily.

4.2 SAML Certificate Lifecycle Management

An often-overlooked dimension of security hardening in banking environments is the lifecycle of SAML certificates used for federated authentication integrations. Certificate expiry events in this context cause complete authentication outages for affected client integrations a category-1 incident in financial services environments.

The framework incorporates a Python-based certificate monitoring function that queries all SAML identity provider configurations registered in the environment and generates alerts when certificate validity windows fall below 60 days. The function produces a renewal runbook attachment with each alert, pre-populated with the specific certificate details and the Method of Procedure (MOP) steps for that provider's renewal process. This proactive monitoring approach successfully prevented two certificate expiry events during the observation period that would otherwise have caused production authentication outages.

5. RESULTS AND OPERATIONAL IMPACT

The framework was fully operational across all environments by August 2023. The following results are drawn from operational telemetry collected between August 2023 and January 2024 a six-month observation window spanning six complete monthly patch cycles.

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

Metric	Before (Baseline)	After (Framework)	Improvement
Manual patching cycle time	12–18 hrs/month	2–3 hrs/month	83%
Patch compliance rate (SOC 2)	71%	98.6%	+27.6 pts
Post-patch incidents (P1/P2)	4.2/quarter	0.6/quarter	86%
Audit report generation time	8 hrs manual	< 15 min automated	97%
CrowdStrike agent deployment	3 days manual	4 hrs automated	94%
SAML cert expiry incidents	2 per year	0 (proactive alerts)	100%
IaC coverage (DB + infra ops)	35%	89%	+54 pts
AWS monthly operational cost	$—	$140K saved	—

Table 2. Operational metrics comparison: pre-framework baseline (April–July 2023) versus post-deployment results (August 2023–January 2024). Baseline figures are derived from incident records, time-tracking entries, and SOC 2 audit findings.

The most operationally significant result is the reduction in post-patch P1/P2 incidents from 4.2 to 0.6 per quarter. This reduction is attributed primarily to three changes: the pre-patch backup gate (which enabled rapid rollback in the two instances where patches caused application regressions), the environment progression model (which allowed regressions to be detected in staging before reaching production), and the CrowdStrike sensor exclusion tuning that eliminated false-positive prevention events that had previously caused application process terminations.

The compliance improvement from 71% to 98.6% against SOC 2 patch-related controls reflects both the higher patch application rate and the improved evidence collection. The 1.4% gap that remains is attributable to a small number of hosts in isolated non-production environments that are not covered by the SSM patch baseline due to network segmentation constraints that are being addressed in a follow-on infrastructure project.

6. IMPLEMENTATION CHALLENGES AND FAILURE MODES

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

This section documents failure modes encountered during the framework implementation that are not anticipated in vendor documentation or existing literature. These observations are offered as practical guidance for SRE teams undertaking similar initiatives.

6.1 SSM Agent Registration Drift

The most frequently encountered failure mode during the first three months of operation was SSM agent de-registration following AMI refreshes. When a new AMI is used to replace an EC2 instance, the instance ID changes, and the SSM agent must re-register with the management account. If the launch template does not include the correct IAM instance profile and the SSM agent bootstrap configuration in userdata, the new instance will not appear in Fleet Manager.

Resolution required a two-part fix: updating all Terraform launch template modules to include a standardized userdata block that bootstraps and verifies SSM agent registration as part of instance initialization, and adding the inventory pre-flight check described in Section 3.1 to surface registration lapses before patch execution begins.

6.2 CrowdStrike Sensor and Kernel Compatibility

During kernel patching on RHEL 7 hosts, CrowdStrike Falcon sensors on sensor version 6.x exhibited a known compatibility issue with kernel versions above 3.10.0-1160.x that caused the sensor to fail to load after reboot. This issue was not surfaced by the CrowdStrike compatibility matrix at the time of patching.

The resolution involved pinning the kernel update to a specific version on affected hosts and scheduling a concurrent sensor upgrade to version 7.x. The incident highlighted the need for a coordination check between the patch baseline and sensor version matrix before executing kernel updates — a check that was subsequently automated as an additional pre-flight gate in the state machine.

6.3 Terraform State Drift in Multi-Account Environments

Managing Terraform state across multiple AWS accounts each representing an isolated client environment introduced state drift when manual console operations were performed in response to production incidents outside of the IaC workflow. The most common pattern was the manual modification of IAM role policies during incident response, which created divergence between the Terraform state and the actual policy configuration.

The framework addresses this through scheduled Terraform plan runs that generate drift detection reports distributed to the SRE team each morning. Detected drift is reviewed during the daily standup and either ratified (incorporated into Terraform) or remediated (reverted to IaC-defined state). This continuous drift detection approach has reduced the occurrence of "configuration surprises" during patch cycles.

7. DISCUSSION

Several observations from this implementation have broader applicability to SRE teams in regulated industries:

Compliance artifacts should be generated as operational byproducts, not as separate documentation activities. The consistent failure of manual compliance documentation processes — well-documented in the auditing literature [9] is fundamentally a prioritization problem: compliance documentation competes with operational work for engineer time, and operational urgency wins. Designing automation that produces audit-traceable

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

output as an inherent property of its execution eliminates this competition.

The gap between detection and prevention in EDR deployment is an organizational risk, not a technical one. CrowdStrike sensors in Detection mode provide telemetry but no protection. The common practice of running in Detection mode indefinitely while nominally collecting data to tune exclusions frequently reflects organizational risk tolerance rather than technical necessity. The phased Prevention mode rollout described in this paper demonstrates that a structured transition can be completed within two weeks with minimal disruption.

IaC coverage of operational activities must be treated as a first-class engineering goal. The pattern of using the console for incident response and then failing to codify the resulting configuration in Terraform is pervasive. Drift detection reporting, made visible in daily team communication, creates accountability that reduces this pattern without requiring policy enforcement.

8. CONCLUSION

This paper has described a production-deployed framework for automated, compliance-driven patch management and security hardening across a multi-cloud banking infrastructure. The framework demonstrates that the core challenges of patch management in regulated financial services coordination complexity, compliance documentation burden, and cross-environment consistency are addressable through the disciplined integration of existing tooling: IaC, SSM, Python orchestration, and endpoint security agents.

The quantitative results over a six-month observation period show substantive improvements across every measured dimension: an 83% reduction in manual patching effort, a 27.6-point improvement in SOC 2 compliance posture, an 86% reduction in post-patch production incidents, and the complete elimination of audit-report manual effort. These results suggest that the marginal investment in framework development estimated at approximately six weeks of senior SRE effort is repaid within the first operational quarter.

Future work will extend the framework to cover container image patching within EKS workloads, integrate with the organization's change management platform to automate the creation and closure of change records, and evaluate the applicability of the compliance mapping methodology to PCI-DSS requirements in addition to SOC 2.

References

[1] Kim, S., Park, J., & Lee, H. (2020). Analysis of patch management failures in financial sector cybersecurity incidents, 2017–2020. Journal of Financial Cybersecurity, 4(2), 88–103.

[2] PCI Security Standards Council. (2022). PCI Data Security Standard (PCI DSS) Version 4.0. Wakefield, MA: PCI SSC.

[3] HashiCorp. (2023). State of Cloud Strategy Survey 2023. San Francisco, CA: HashiCorp Inc.

[4] Brikman, Y. (2022). Terraform: Up and Running (3rd ed.). Sebastopol, CA: O'Reilly Media.

[5] Hendricks, C., & Morrison, T. (2022). Security baseline enforcement with declarative IaC in regulated cloud environments. IEEE Cloud Computing, 9(4), 44–53.

[6] Hochstein, L., & Moser, R. (2017). Ansible: Up and Running (2nd ed.). Sebastopol, CA: O'Reilly Media.

The American Journal of Engineering and Technology
(ISSN – 2689-0984)
VOLUME05ISSUE12Pages:68-80
SJIF IMPACT FACTOR (2020: 5. 32)(2021: 5. 705)(2022: 6. 456) (2023: 7. 038)
OCLC– 1121105677
Publisher:The USA Journals

[7] CrowdStrike. (2023). Falcon Platform Architecture Guide: Sensor Deployment in Cloud-Native Environments. Sunnyvale, CA: CrowdStrike Inc.

[8] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA: O'Reilly Media.

[9] AICPA. (2017). Trust Services Criteria for Security, Availability, Processing Integrity, Confidentiality, and Privacy (SOC 2). New York, NY: American Institute of Certified Public Accountants.

[10] Sharma, R., & Gupta, A. (2021). Automation of vulnerability remediation workflows in multi-cloud enterprise environments. International Journal of Information Security and Privacy, 15(3), 1–19.

[11] National Institute of Standards and Technology. (2022). NIST SP 800-40 Rev. 4: Guide to Enterprise Patch Management Planning. Gaithersburg, MD: NIST.

[12] Haber, M. J., & Rolls, B. (2020). Privileged Attack Vectors: Building Effective Cyber-Defense Strategies to Protect Organizations. New York, NY: Apress.