SunBeam Institute of Information Technology

Hinjawadi, Pune

A Project Report on

Autonomous DevSecOps Architecture

with Active Threat Mitigation

Submitted in partial fulfillment of the requirements for the

Post Graduate Diploma in

IT Infrastructure, Systems & Security (PG-DITISS)

Submitted by:

Rahul Datta

PRN: 250844223033

February 2026

SunBeam Infotech Private Limited, "Sunbeam IT Park", Second Floor, Phase 2 of Rajiv Gandhi Infotech Park,Hinjawadi, Pune - 411057, MH-INDIA

SunBeam Institute of Information Technology

Hinjawadi, Pune

CERTIFICATE

This is to certify that the project report entitled **"Autonomous DevSecOps Architecture with Active Threat Mitigation"** is a bona fide record of work carried out by **Rahul Datta** (PRN: 250844223033), in partial fulfillment of the requirements for the **Post Graduate Diploma in IT Infrastructure, Systems & Security (PG-DITISS)** at SunBeam Institute of Information Technology, Pune, during the academic year 2025–2026 (AUG-2025 Batch).

To the best of my knowledge, the matter embodied in this project report has not been submitted to any other University or Institute for the award of any degree or diploma.

Mr. Vishal Salunkhe

Course Coordinator

Acknowledgement

I would like to express my sincere gratitude to my lab mentors, **Mr. Meet Patel** and **Mr. Gajanan Taur**, for their technical guidance, valuable suggestions, and continuous support throughout the development of this project. Their practical insights were instrumental in overcoming implementation challenges.

I am thankful to **Mr. Vishal Salunkhe**, Course Coordinator, PG-DITISS, for providing the necessary infrastructure, laboratory facilities, and a conducive academic environment.

I also extend my gratitude to the **Executive Director**, SunBeam Infotech, and all faculty members for their support and for fostering a structured and industry-oriented learning environment.

I would like to acknowledge and thank my group members for their cooperation, discussions, and contributions during the project:

- **Ashish Kumar Yadav** (PRN: 250844223002)
- **Kapse Shriyash Dhananjay** (PRN: 250844223018)
- **Prateek Srivastava** (PRN: 250844223031)

Finally, I thank my family and batchmates for their constant encouragement, patience, and support throughout the project work.

Date: February 2, 2026
Place: Pune, India

Rahul Datta
(PRN: 250844223033)

Abstract

Modern enterprise infrastructure faces the challenge of maintaining rapid deployment velocity while ensuring strict security. This project presented an autonomous **DevSecOps architecture** operating on **Defense-in-Depth** principles, designed to reconcile automation with isolation.

The system was constructed upon a **three-zone security model**: an untrusted public sector, a hardened DMZ acting as a protective buffer, and a strictly isolated internal vault. The DMZ enforced security through **active threat mitigation**, utilizing intrusion prevention systems and deception mechanisms to neutralize attacks in real-time before they reached critical assets. Behind this defensive layer, the internal vault hosted a zero-touch CI/CD pipeline that orchestrated containerized deployments and managed Enterprise PKI, ensuring that code transitioned from commit to production without compromising the air-gap.

Operational integrity was maintained through a dual strategy of **centralized observability** and **active alerting**. While a unified dashboard provided deep visibility into logs and metrics, an automated intelligence engine proactively monitored system health and security events, dispatching real-time notifications to administrators to ensure rapid response capabilities.

Contents

Chapter 1

Introduction

1.1 Motivation

Organizations increasingly adopt DevOps methodologies to accelerate software delivery, yet security often becomes an afterthought. This creates production vulnerabilities, compliance violations, and delayed incident response. Traditional perimeter-based security proves inadequate in modern distributed systems where a single compromise can enable unrestricted lateral movement across flat network architectures.

Simultaneously, regulatory frameworks mandate demonstrable security controls, audit trails, and data sovereignty. Organizations in regulated industries require air-gapped environments that complicate traditional DevOps tooling designed for cloud platforms.

Foundational Concept: The DevSecOps Challenge

Security teams demand isolation and manual gates while development teams require rapid iteration and automation. Traditional approaches create organizational silos where security "approves" deployments, creating bottlenecks. True DevSecOps requires architectural solutions where security enables velocity rather than hindering it.

This project addresses: *Can enterprise-grade security coexist with agile development practices in resource-constrained, isolated environments?*

1.2 Problem Statement

Modern organizations need IT infrastructure satisfying contradictory requirements:

1. **Security Isolation:** Critical systems must operate in air-gapped segments to prevent internet exposure and limit blast radius of compromises
2. **Deployment Velocity:** Development teams need automated CI/CD enabling multiple daily deployments without manual security reviews
3. **Autonomous Threat Response:** Security systems must detect and neutralize threats

in sub-second timeframes; human response latencies prove inadequate against automated attacks

4. **Resource Efficiency:** Edge deployments and development environments require solutions operating within commodity hardware constraints
5. **Audit Trail Integrity:** Forensic logs must persist in tamper-resistant storage for compliance and post-incident analysis

◎ Key Objective: Project Objectives

Design and implement an integrated architecture demonstrating that security isolation and deployment automation are complementary, not competing priorities. Prove the concept works on resource-constrained hardware (8GB total RAM) using entirely open-source components.

1.3 Objectives

1.3.1 Primary Objectives

1. **Design Three-Zone Security Architecture:**
   - DMZ hosting public services with hardened configurations
   - Air-gapped Internal Vault for management infrastructure
   - Controlled proxy-based internet gateway
2. **Build Automated Threat Detection System:**
   - Real-time traffic analysis using Suricata IDS
   - Automated blocking via Fail2Ban integration
   - Custom threat intelligence engine
   - High-interaction honeypot for behavioral analysis
3. **Establish Zero-Touch CI/CD Pipeline:**
   - Jenkins-based automation on isolated network
   - Container deployment via K3s orchestration
   - Git-based workflow with polling mechanism
4. **Implement Enterprise PKI:**
   - Three-tier certificate authority (Root, Intermediate, Server)
   - Trusted HTTPS without commercial certificates
5. **Deploy Comprehensive Observability:**
   - Centralized logging via Loki

- Metrics collection through Prometheus
- Visualization dashboards in Grafana

1.3.2 Scope and Limitations

In Scope:

- Single-node K3s cluster demonstrating orchestration concepts
- Active defense against common reconnaissance attacks (port scans, brute force)
- Air-gap implementation with controlled proxy access
- Basic CI/CD pipeline for containerized applications

Out of Scope:

- Multi-node Kubernetes clustering and high availability
- DDoS mitigation and advanced threat detection
- Compliance automation (SOC2, ISO 27001 controls)
- Production-scale load balancing and auto-scaling

1.4 Technology Overview

The project integrates open-source components into a cohesive security architecture:

Table 1.1: Core Technology Stack

Component	**Technology**	**Purpose**
Operating System	Ubuntu 22.04 LTS	Base platform for both VMs
Virtualization	Oracle VirtualBox	Isolated network simulation
Container Runtime	Docker	Application containerization
Orchestration	K3s	Lightweight Kubernetes
Intrusion Detection	Suricata	Network-based IDS/IPS
Automated Response	Fail2Ban	Dynamic firewall management
Honeypot	Cowrie	SSH deception system
CI/CD	Jenkins	Automation server
Reverse Proxy	Nginx	TLS termination, routing
Forward Proxy	Squid	Whitelist-based access
DNS	BIND9	Split-horizon resolution
Log Aggregation	Loki	Centralized log storage
Metrics	Prometheus	Time-series monitoring
Visualization	Grafana	Dashboards and alerting

Why This Matters: Technology Selection Rationale

Each technology was selected for specific technical reasons: Suricata provides multi-threaded performance while maintaining Snort rule compatibility. K3s reduces Kubernetes memory footprint from 4GB to 512MB through SQLite instead of etcd. Loki's label-based indexing drastically reduces storage versus Elasticsearch's full-text indexing. These aren't arbitrary choices; they're optimized for resource-constrained, air-gapped deployment.

Chapter 2

System Architecture

2.1 High-Level Design

Strategic Vision: Defense-in-Depth Philosophy

The architecture implements security through network segmentation rather than relying solely on perimeter defense. By physically isolating management infrastructure from public services, we create a system where compromising one zone does not grant access to the entire infrastructure. This "Zero-Trust" model assumes breach is inevitable and designs for containment.

2.1.1 Three-Zone Security Model

The network is divided into three distinct security zones, each serving a specific purpose:

Zone A: Public (Untrusted)

- Windows 11 host machine representing the internet
- Source of legitimate users and potential attackers
- No direct access to Internal Vault

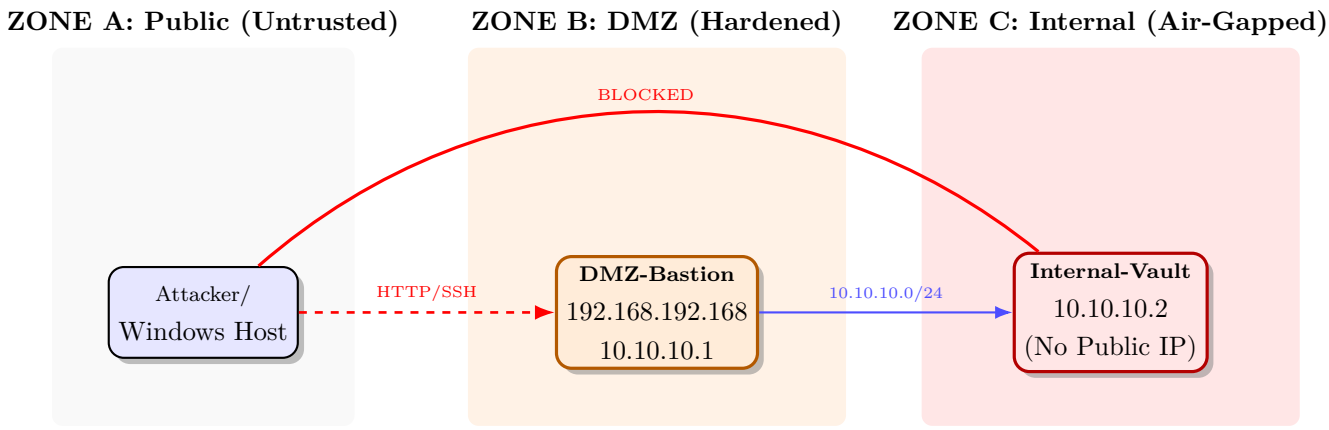
Zone B: DMZ (Semi-Trusted)

- Publicly accessible services (web applications)
- Active security stack (Suricata, Fail2Ban, Cowrie)
- Reverse proxy (Nginx) for TLS termination
- Forward proxy (Squid) for controlled internet access

Zone C: Internal Vault (Trusted)

- Management infrastructure (Jenkins, K3s)
- Observability stack (Prometheus, Loki, Grafana)

- Split-horizon DNS (BIND9)
- No direct internet connection
- Initiates all connections to DMZ



2.1.2 Network Topology

Technical Detail: VirtualBox Network Configuration

Three network types provide distinct connectivity:
NAT Network (enp0s3 on DMZ): Internet access for package installation. Uses VirtualBox's internal DHCP.
Host-Only Network (enp0s8 on DMZ): Management plane from Windows host. Static IP 192.168.192.168/24 with DHCP disabled.
Internal Network (enp0s9 on DMZ, enp0s3 on Vault): Isolated segment (10.10.10.0/24) connecting DMZ and Vault. VirtualBox enforces isolation; no routing to external networks even if misconfigured.

2.1.3 Security Zones and Trust Boundaries

Table 2.1: Trust Boundary Matrix

From → To	Public	DMZ	Vault
Public →	N/A	HTTPS/SSH	Blocked
DMZ →	Via Proxy	N/A	Allowed (Proxy)
Vault →	Via Proxy	SSH/HTTP	N/A

2.2 Component Architecture

2.2.1 DMZ Bastion Components

The DMZ hosts public-facing services and security enforcement:

Table 2.2: DMZ Bastion Component Stack (192.168.192.168 / 10.10.10.1)

Category	Component	Purpose
Network Services	Nginx	Reverse proxy for HTTPS traffic
	Squid	HTTP/HTTPS proxy with domain whitelist
Security Stack	Suricata	Network IDS with deep packet inspection
	Fail2Ban	Automated IPS with IP blocking
	Cowrie	SSH honeypot on port 22
Infrastructure	Docker	Container runtime for security tools
	Promtail	Log shipper to Loki
	Node Exporter	System metrics exporter

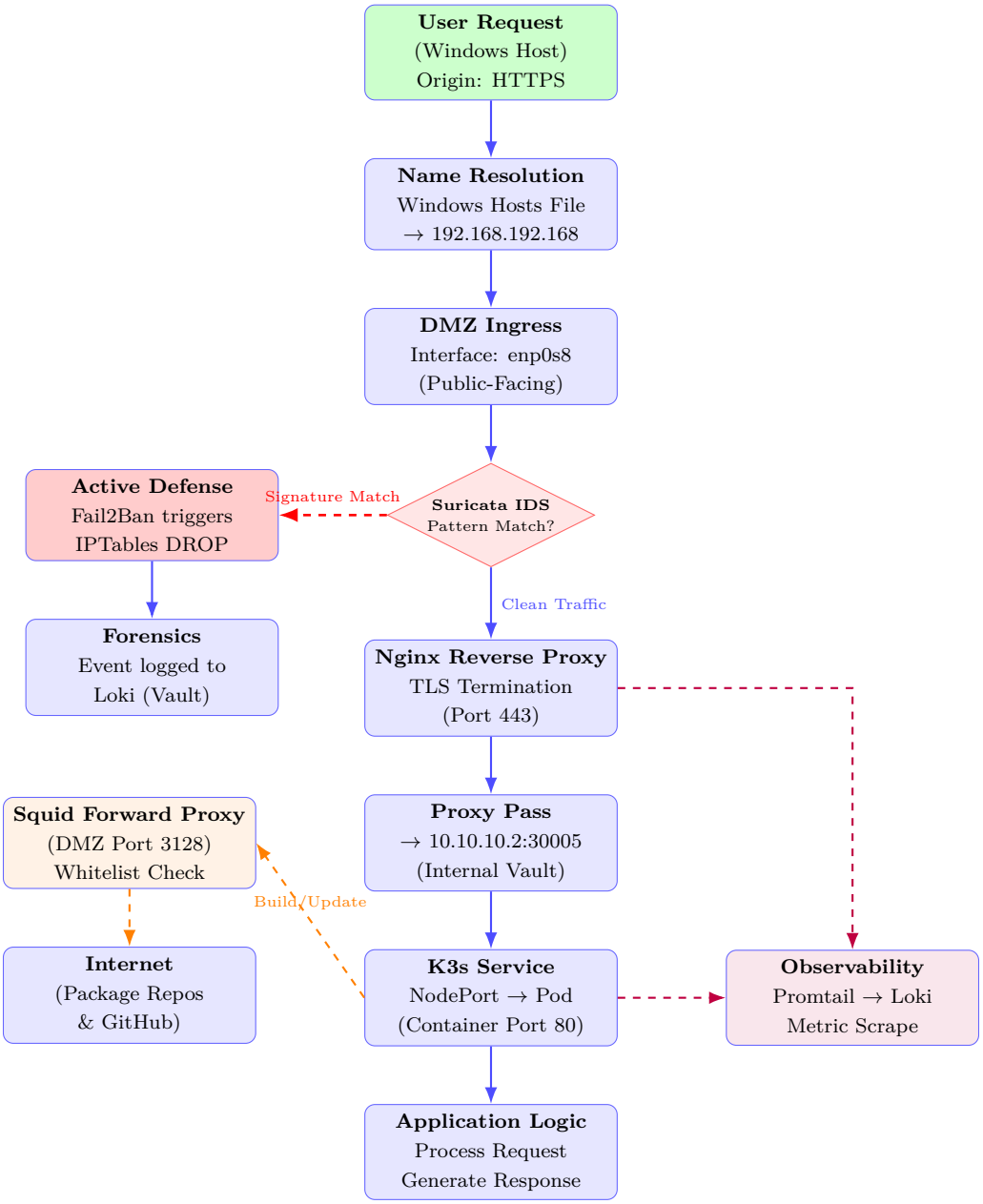
2.2.2 Internal Vault Components

The Vault hosts management and observability infrastructure:

Table 2.3: Internal Vault Component Stack (10.10.10.2 - Air-Gapped)

Category	Component	Purpose
CI/CD Pipeline	Jenkins	Automation server and build orchestrator
	Git	Local source code repository
	Ansible	System maintenance and health reporting
	BIND9	Split-horizon DNS server
Container Platform	K3s	Lightweight Kubernetes cluster
	Docker	Container runtime and image building
Observability	Loki	Centralized log aggregation and storage
	Prometheus	Time-series metrics database
	Grafana	Unified visualization and dashboards

2.2.3 Data Flow Architecture



2.3 CI/CD Pipeline Architecture

◎ Key Objective: Zero-Touch Deployment

The CI/CD pipeline enables automated deployments from code commit to production without manual intervention, while maintaining complete air-gap isolation. Jenkins polls a local Git repository, builds containers, and deploys to K3s; all within the Internal Vault.

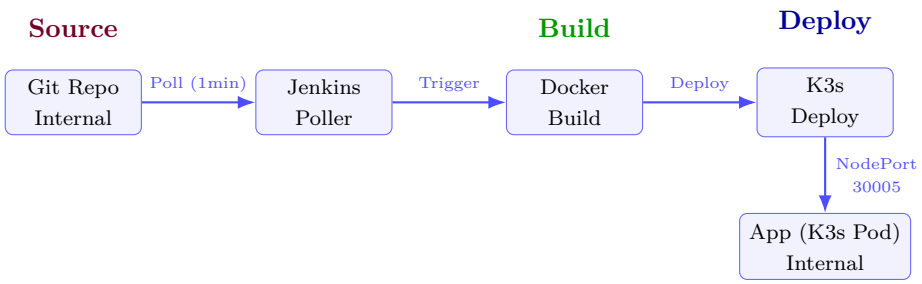


Figure 2.1: CI/CD Pipeline Flow

Why This Matters: Git Polling vs Webhooks

The pipeline uses Git polling (1-minute intervals) rather than webhooks to maintain air-gap. Webhooks require inbound network connections, which would compromise isolation. Trade-off: Maximum 1-minute deployment delay versus immediate triggers, but this preserves network security.

Chapter 3

Implementation Strategy

3.1 Network Infrastructure Setup

3.1.1 Air-Gap Bootstrap Challenge

Critical: The Bootstrap Paradox

The fundamental challenge: Internal Vault needs internet to install K3s and packages, but final state must have zero direct internet access. Cannot configure proxy before proxy exists; a circular dependency.

Solution: Staged Network Configuration

1. Initial setup with NAT enabled on Internal Vault
2. Install all required packages (Docker, K3s, monitoring tools)
3. Disable NAT interface, configure proxy-only access via DMZ
4. Verify air-gap integrity through connectivity tests

This pragmatic approach avoids the circular dependency. In production environments, this would be handled through offline installers or internal package mirrors.

3.1.2 VirtualBox Network Configuration

Technical Detail: Network Interface Assignment

DMZ Bastion (3 interfaces):

- enp0s3: NAT (internet access, DHCP)
- enp0s8: Host-Only 192.168.192.168/24 (management from Windows)
- enp0s9: Internal 10.10.10.1/24 (connection to Vault)

Internal Vault (1 interface):

- enp0s3: Internal 10.10.10.2/24 (isolated connection to DMZ)

The Internal Network type is critical; VirtualBox enforces isolation at the hypervisor level, preventing routing to external networks even if routing tables are misconfigured.

3.2 Security Stack Integration

3.2.1 Suricata IDS Deployment

Suricata monitors network interfaces in AF_PACKET mode for kernel-level packet capture, configured to monitor both management (enp0s8) and internal (enp0s9) interfaces.

Key Configuration Decisions:

- Multi-threaded mode enabled (utilizes all CPU cores)
- EVE JSON output for structured logging compatible with Loki
- Community ruleset (30,000+ signatures) plus custom port scan detection
- Fast.log output for Fail2Ban integration

Why This Matters: EVE JSON Format

Traditional syslog output requires complex regex parsing. JSON provides structured data with source IP, destination, signature ID, and timestamp as discrete fields; trivial to query in Loki and process programmatically.

3.2.2 Fail2Ban Integration

Fail2Ban monitors Suricata's fast.log file for specific alert patterns. When custom signature 1000001 (Nmap stealth scan) appears:

1. Extracts source IP using regex pattern matching
2. Invokes iptables to add DROP rule in f2b-suricata chain
3. Logs action to systemd journal
4. Maintains ban for 10 minutes (configurable)
5. Auto-removes ban after timeout

Foundational Concept: Signature-Based vs Anomaly Detection

Signature-based detection provides deterministic behavior. When signature 1000001 triggers, we know exactly what attack occurred. Anomaly detection produces probabilistic scores requiring data science expertise to interpret. For a basic project with limited resources, signature-based detection is the pragmatic choice.

3.2.3 Cowrie Honeypot Setup

Cowrie emulates SSH service on port 22 while real SSH moved to port 2222. Provides realistic interaction:

- Fake filesystem with /etc/passwd, /var/log, typical directories
- Command emulation for ls, cat, wget, curl
- Fake user accounts (root/admin, postgres/postgres for easy compromise)
- Session recording capturing all attacker commands
- JSON logging to Loki for forensic analysis

Why This Matters: Port Placement Strategy

Port 22 is the first target for SSH attacks. By placing the honeypot there, we absorb brute-force attempts while real management SSH remains undiscovered on port 2222. Attackers waste time exploring a fake system, giving us time to analyze their methodology.

3.3 Kubernetes in Air-Gap

3.3.1 K3s Proxy Configuration

Critical: Multi-Layer Proxy Configuration

K3s pulls container images from DockerHub. Air-gap blocks this. Solution requires configuration at multiple points; shell environment, systemd service, and kubeconfig permissions.

Configuration Layers:

1. Shell Environment (Installation):

```
export HTTP_PROXY=http://10.10.10.1:3128
export NO_PROXY=localhost,10.10.10.0/24,.svc,.cluster.local
```

2. Systemd Service Environment:
Created `/etc/systemd/system/k3s.service.env` with proxy settings. K3s systemd service loads this on startup for persistent proxy configuration.

3. Kubeconfig Permissions:
Modified K3s service to generate kubeconfig with 644 permissions (default 600 prevents Jenkins user access for automated deployments).

3.3.2 Squid Proxy Whitelist

Squid on DMZ enforces strict domain whitelist:

- docker.io, docker.com (container images)

- github.com, githubusercontent.com (K3s binary, scripts)
- ubuntu package repositories (security updates)

Requests to non-whitelisted domains receive 403 Forbidden, creating "controlled air-gap"; strong isolation with selective connectivity for essential services only.

3.4 CI/CD Pipeline Implementation

3.4.1 Jenkins Pipeline Structure

Jenkins runs on Internal Vault, orchestrating the complete deployment lifecycle:

◎ Key Objective: Deployment Pipeline Stages

Stage 1 - Source: Poll local Git repository every 1 minute for new commits
Stage 2 - Build: Execute Docker build, tag with timestamp
Stage 3 - Deploy: Update K3s deployment with new image
Stage 4 - Verify: Check pod status and readiness

All stages execute on Internal Vault. The application is exposed to DMZ via K3s NodePort (30005), which Nginx reverse-proxies to HTTPS port 443.

3.5 DNS and Certificate Infrastructure

3.5.1 Split-Horizon DNS

BIND9 on the Internal Vault provides split-horizon DNS functionality:

- Authoritative for corp.local zone (jenkins.corp.local → 10.10.10.2, app.corp.local → 10.10.10.2)
- Forwards external queries via NAT/routing through the DMZ
- DMZ and Vault systems use the Internal Vault (10.10.10.2) as DNS resolver

? Why This Matters: Split-Horizon Necessity

Internal services need local names (app.corp.local, jenkins.corp.local) but also need to resolve external domains for package updates and container registry access. A single DNS server handles both local zone authority and external forwarding.

3.5.2 PKI Certificate Generation

Strategic Vision: Three-Tier Certificate Authority

Enterprise PKI requires a chain of trust: Root CA (offline, signs Intermediate) → Intermediate CA (operational, signs server certs) → Server Certificate (used by applications). This hierarchy enables certificate revocation and key rotation without re-trusting all clients.

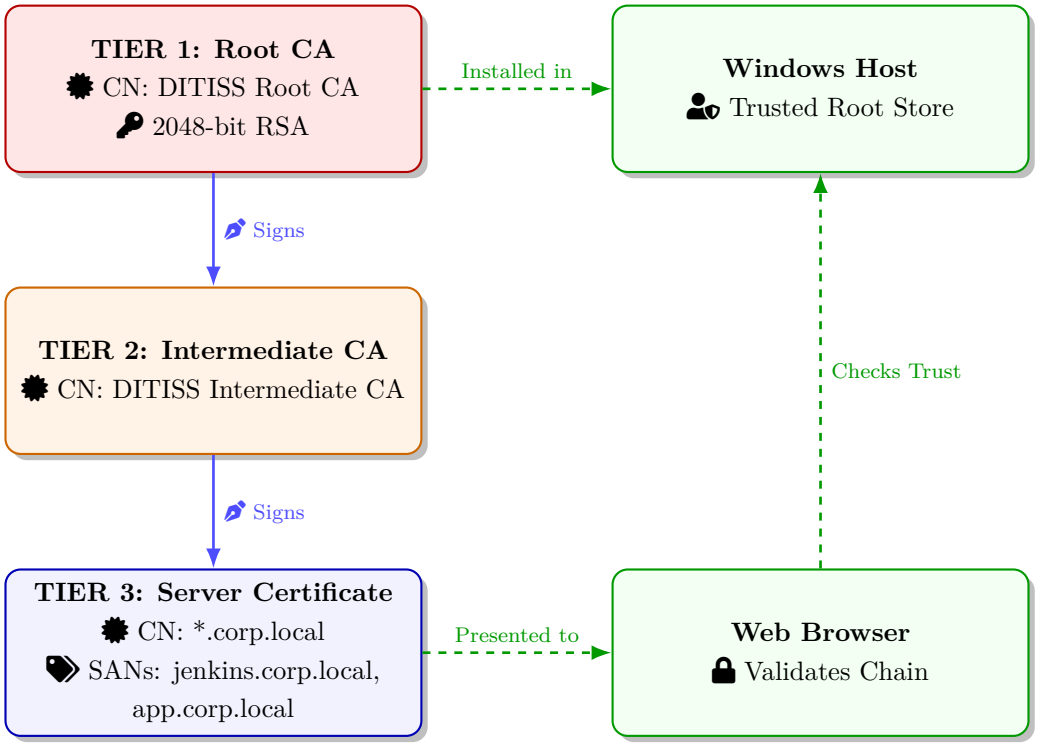
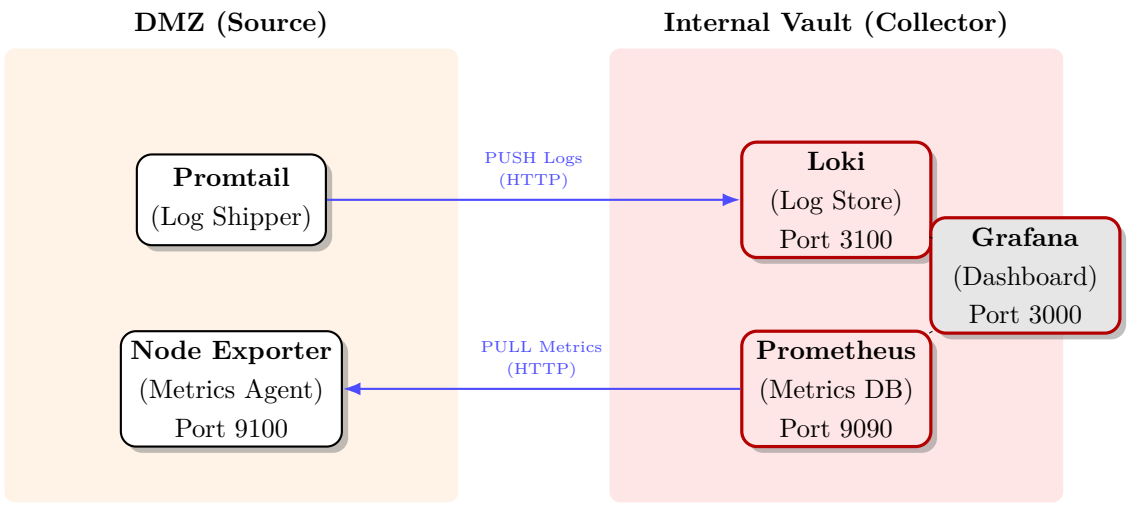


Figure 3.1: 3-Tier Enterprise PKI Architecture and Browser Trust Chain

3.6 Observability Stack



3.6.1 Loki Log Aggregation

Loki stores logs with label-based indexing instead of full-text indexing. Labels include: host, service, severity.

Promtail agent on the DMZ VM ships logs to Loki:

- Suricata EVE JSON logs (security events)
- Cowrie JSON logs (honeypot sessions)

Why This Matters: Loki vs Elasticsearch

Storage efficiency drives the choice. Elasticsearch indexes every word in every log, requiring 10-20GB for typical retention. Loki indexes only labels, requiring 1-2GB for the same period. In resource-constrained environments, this difference is critical.

3.6.2 Prometheus Metrics Collection

Prometheus scrapes metrics every 15 seconds from:

- Node Exporter on the DMZ (system metrics: CPU, memory, disk)

15-day retention configured to balance disk usage and historical analysis needs.

3.6.3 Grafana Dashboards

Two primary dashboards provide operational visibility:

Security Dashboard: Attack timeline, blocked IPs, honeypot sessions, Suricata alert counts

Operations Dashboard: System resources, deployment frequency, service health

Grafana queries both Prometheus (metrics) and Loki (logs) in a unified interface, accessible from Windows host via SSH tunnel to Internal Vault.

Chapter 4

Security Features

4.1 Active Defense Mechanism

4.1.1 Detection Strategy

Suricata evaluates every packet against a comprehensive ruleset, including custom rules for common reconnaissance attacks:

- Nmap SYN scans (half-open connections to multiple ports)
- Nmap stealth scans (FIN, NULL, Xmas scan patterns)
- SSH brute force (rapid connection attempts)
- Web vulnerability scanners (Nikto, SQLMap signatures)

Technical Detail: Detection Logic Example

Nmap Stealth Scan Detection:
Rule: If (SYN packets to >100 ports within 5 seconds) from same source IP, then generate alert with signature ID 1000001.
This signature triggers on the characteristic behavior of port scanners rather than relying on specific packet patterns, making it resilient against evasion techniques.

4.1.2 Automated Response

Fail2Ban translates detection into enforcement through a simple but effective workflow:

1. Monitors Suricata fast.log for signature 1000001
2. Regex extracts source IP from alert message
3. Executes: `iptables -I f2b-suricata -s <IP> -j DROP`
4. Attacker's subsequent packets dropped at kernel level
5. Ban persists 10 minutes, then auto-removed

Why This Matters: Temporary vs Permanent Bans

Permanent bans risk blocking legitimate users behind NAT who share IPs with attackers. 10-minute timeout forces attackers to wait (disrupting automated tools) while preserving access for genuine users who might have triggered false positives. This balances security with operational availability.

4.1.3 Forensic Trail

Every security event is logged to Loki on Internal Vault, creating a tamper-proof audit trail:

- Suricata alert with full packet metadata (source, destination, signature)
- Fail2Ban ban action with IP and timestamp
- IPtables rule addition and removal events
- Honeypot session if attacker probed port 22

Key Objective: Tamper-Proof Logging

Logs stored on Internal Vault cannot be modified by attackers who compromise the DMZ. Even if an attacker gains root access on the DMZ Bastion, they cannot delete or modify evidence stored on the air-gapped Vault. This separation ensures forensic integrity.

4.2 Honeypot Deception System

4.2.1 High-Interaction Design

Cowrie provides a realistic SSH environment designed to deceive attackers:

- Accepts common credentials (root/admin, admin/admin, postgres/postgres)
- Presents fake shell prompt with hostname "internal-db-prod" (suggesting a production database server)
- Emulates bash commands with realistic output
- Fake /etc/shadow file contains hashed passwords
- Simulated /var/www/html directory suggests web server

From the attacker's perspective, they've successfully compromised a production system. In reality, they're in a sandboxed environment with all actions logged.

4.2.2 Behavioral Intelligence

Cowrie logs reveal attacker methodology and tactics:

- Initial reconnaissance (whoami, uname -a, ifconfig)
- Privilege escalation attempts (sudo, su)

- Persistence mechanisms (crontab -e, rc.local modification)
- Malware download (wget malicious.sh, curl cryptominer)

Foundational Concept: Intelligence Value

Pattern analysis across multiple attacks reveals shared playbooks or automated tools. If multiple attackers use identical command sequences, it suggests a common tool or training. This intelligence can inform signature updates and help predict attacker next steps.

4.2.3 Attacker Time Consumption

The honeypot serves as a "tar pit" delaying attackers. While exploring the fake system, valuable time is wasted. Real SSH on port 2222 remains undiscovered, and security teams gain time to analyze behavior and update defenses.

4.3 Enterprise PKI Implementation

4.3.1 Certificate Hierarchy

Root CA:

- Common Name: DITISS Root CA
- Key Size: 2048-bit RSA
- Purpose: Signs Intermediate CA only (kept offline in production)
- Storage: Private key secured on Internal Vault

Intermediate CA:

- Common Name: DITISS Intermediate CA
- Signed By: Root CA
- Purpose: Signs server certificates (operational use)
- Extension: CA:TRUE (required to sign other certificates)

Server Certificate:

- Common Name: *.corp.local (wildcard for all subdomains)
- Subject Alternative Names: jenkins.corp.local, app.corp.local, 192.168.192.168
- Signed By: Intermediate CA
- Purpose: TLS termination in Nginx

Why This Matters: Wildcard Certificate Benefits

Single certificate covers all subdomains (*.corp.local), simplifying management versus maintaining individual certificates per service. Trade-off: If private key is compromised, all subdomains are affected. For a lab environment, the management simplicity outweighs the security risk.

4.3.2 Trust Establishment

Browser trust chain validation process:

1. Browser receives server certificate during TLS handshake
2. Checks issuer: DITISS Intermediate CA (not in trust store)
3. Follows chain to Root CA
4. Finds Root CA in Windows Trusted Root Certification Authorities
5. Verifies cryptographic signatures throughout chain
6. Validates certificate not expired and hostname matches SAN
7. Displays padlock icon (connection trusted)

Without Root CA installation, browser shows "NET::ERR_CERT_AUTHORITY_INVALID". Installing Root CA makes the entire chain trusted, enabling browser to validate certificates signed by our Intermediate CA.

4.4 Threat Intelligence Engine

4.4.1 Modular Signature System

Custom Python daemon queries Loki for threat patterns defined in external JSON database:

```
{
  "signature_id": "TI-001",
  "name": "Cryptocurrency Miner Download",
  "loki_query": "{job=\"honeypot\"} |= \"wget\" |= \"xmrig\"",
  "severity": "critical",
  "description": "Attacker downloading XMRig monero miner"
}
```

When query returns results, daemon extracts context and sends alerts via email through the DMZ NAT gateway.

Why This Matters: External Signature Files

Separating detection logic (Python code) from detection rules (JSON signatures) allows security teams to update threat patterns without modifying code. New signatures can be added by editing JSON, enabling rapid response to emerging threats without developer involvement.

4.4.2 Email Alerting Through Air-Gap

Critical: Controlled Air-Gap Puncture

Internal Vault has no direct internet access but needs to send email alerts. Solution: DMZ configured as NAT router for SMTP (port 587) and DNS (port 53) traffic only. All other internet access from Vault remains blocked.

This "puncture" in the air-gap is highly controlled—only essential protocols for alerting are permitted, while general internet access remains prohibited. This was achieved using **IP Masquerading** via `iptables`, allowing the internal traffic to appear as if originating from the DMZ's public-facing interface. Email notification functionality is standardized through a custom Python library (`alerter.py`) using Gmail App Passwords and SMTP authentication, enabling both the threat intelligence engine and service watchdog to send alerts via a common interface.

4.4.3 Service Availability Monitoring

A complementary Python daemon (`service_watchdog.py`) monitors critical service endpoints defined in `services.json`. This uptime monitor checks HTTP endpoints (Jenkins, K3s application, Nginx) and TCP ports (Cowrie honeypot), tracking state transitions from UP to DOWN. Email alerts are triggered when services become unavailable, providing operational awareness distinct from security event monitoring.

4.5 Automated System Maintenance

To ensure long-term operational stability without manual intervention, an Ansible-based maintenance engine was integrated into the Internal Vault. This system performs daily health checks on the DMZ Bastion, including disk usage analysis and security patch auditing.

4.5.1 Cross-Zone Reporting Logic

A specific challenge arose in reporting health status via email. While the Ansible playbook executes commands on the DMZ Bastion (target), the DMZ host itself is not configured to send emails. The Internal Vault, however, has the established SMTP tunnel (via Phase 3).

To resolve this, the Ansible playbook utilizes a **delegation pattern**:

1. **Gather Facts:** Ansible connects to the DMZ via SSH (port 2222) to run `df -h` and `apt list -upgradable`.
2. **Register Variables:** The output of these commands is captured in the playbook's memory.
3. **Delegate Transmission:** The email task is configured with `delegate_to: localhost`.

This configuration forces the email generation to occur on the **Internal Vault (Controller)**, which has the valid SMTP routing table and credentials, using the data collected from the **DMZ (Target)**. This ensures maintenance alerts reach administrators even if the DMZ's outbound capabilities are restricted.

Chapter 5

Testing and Validation

◎ Key Objective: Testing Approach

Testing focused on demonstrating the key capabilities of the integrated system: air-gap isolation, active defense response, honeypot deception, PKI trust, and automated deployment. All tests were performed on the actual 2-VM setup with the Windows host acting as the attack source.

5.1 Air-Gap Integrity Verification

5.1.1 External Isolation Test

Objective: Verify Internal Vault cannot be reached from Windows host.

Procedure:

1. From Windows Command Prompt: `ping 192.168.192.168`
2. Result: Success (DMZ reachable)
3. From Windows Command Prompt: `ping 10.10.10.2`
4. Result: "Request timed out" (Vault unreachable)

Validation: Internal Vault is completely isolated from external access. Only DMZ is accessible from Windows host.

5.1.2 Controlled Proxy Access Test

Objective: Verify Vault can access whitelisted domains via Squid proxy.

Procedure:

1. From Internal Vault: `curl -x http://10.10.10.1:3128 docker.io`
2. Result: HTTP 200 (whitelisted domain accessible)

3. From Internal Vault: `curl -x http://10.10.10.1:3128 facebook.com`
4. Result: HTTP 403 Forbidden (non-whitelisted blocked)

Validation: Squid proxy enforces whitelist, providing controlled access while maintaining isolation.

5.2 Active Defense Testing

5.2.1 Port Scan Detection and Blocking

Objective: Demonstrate automated detection and blocking of reconnaissance attacks.

Procedure:

1. From Windows: `nmap -sS 192.168.192.168` (stealth scan)
2. Observation: Scan hangs after a few seconds
3. On DMZ: `sudo fail2ban-client status suricata`
4. Result: Windows IP (192.168.192.1) appears in banned list
5. On DMZ: `sudo iptables -L -n | grep 192.168.192.1`
6. Result: DROP rule exists for Windows IP
7. Wait 10 minutes for automatic unban
8. Verify: Ban removed, access restored

Validation: System automatically detected scan pattern, blocked source IP, and auto-removed ban after timeout. Full active defense loop functional.

5.3 Honeypot Effectiveness Testing

5.3.1 SSH Deception Test

Objective: Verify honeypot captures attacker sessions while hiding real SSH.

Procedure:

1. From Windows: `ssh root@192.168.192.168` (port 22, honeypot)
2. Credentials: root/admin
3. Result: Login accepted, shell prompt displayed
4. Execute: `ls -la /etc`, `cat /etc/shadow`, `wget http://malware.example.com`
5. Result: Commands execute, fake output shown
6. Access Grafana, query Loki: `{job="honeypot"}`
7. Result: All commands logged with timestamps and source IP

Validation: Honeypot successfully deceives attackers and captures complete forensic trail.

5.3.2 Real SSH Access Test

Procedure:

1. From Windows: `ssh -p 2222 dmz-bastion-admin@192.168.192.168`
2. Credentials: dmz-bastion-admin/admin
3. Result: Successful login to actual DMZ system

Validation: Real SSH remains accessible on non-standard port, isolated from honeypot.

5.4 PKI and Certificate Validation

5.4.1 Certificate Trust Test

Objective: Verify browser trusts locally-issued certificates.

Procedure:

1. Install Root CA certificate (rootCA.crt) in Windows Trusted Root store
2. Add to Windows hosts file: `192.168.192.168 app.corp.local jenkins.corp.local`
3. Access in browser: `https://app.corp.local`
4. Result: Padlock icon displayed (no certificate warnings)
5. Inspect certificate: Verify chain traces to DITISS Root CA

Validation: Complete PKI chain validated, browser trusts locally-issued certificates.

5.5 CI/CD Pipeline Validation

5.5.1 Automated Deployment Test

Objective: Demonstrate zero-touch deployment from code commit to production.

Procedure:

1. On Internal Vault: Edit application code in `templates/index.html`
2. Commit changes: `git commit -am "Updated app v2.0"`
3. Wait 1 minute (Jenkins polling interval)
4. Observe: Jenkins detects commit, triggers build pipeline
5. Pipeline executes: Checkout → Build → Deploy to K3s
6. Access `https://app.corp.local`
7. Result: Updated application content visible

Validation: Complete CI/CD pipeline functional with zero manual intervention.

5.6 DNS Resolution Testing

5.6.1 Split-Horizon DNS Test

Objective: Verify local zone resolution and external forwarding.

Procedure:

1. From Internal nodes: `nslookup app.corp.local 10.10.10.2`
2. Result: Resolves to 10.10.10.2 (local zone)
3. From Internal Vault: `nslookup google.com`
4. Result: Resolves to public IP (forwarded via NAT tunnel)

Validation: BIND9 correctly handles both local zone authority and external resolution.

5.7 Test Summary

All core functionalities validated through practical demonstrations:

Table 5.1: Testing Summary

Component	Test	Result
Air-Gap	Ping test from Windows to Vault	✔ Blocked
Active Defense	Nmap scan → auto-block	✔ Functional
Honeypot	SSH to port 22, session logged	✔ Functional
PKI	Browser trusts certificates	✔ Validated
CI/CD	Git commit → auto-deploy	✔ Functional
DNS	Local and external resolution	✔ Functional

Foundational Concept: Testing Philosophy

Tests focused on demonstrating integrated functionality rather than exhaustive unit testing. The goal was to prove the architecture works as designed in realistic scenarios, not to achieve comprehensive test coverage. This approach is appropriate for a proof-of-concept implementation.

Chapter 6

Results and Conclusion

6.1 Project Achievements

This project successfully demonstrated that enterprise-grade security and rapid deployment automation can coexist in resource-constrained, isolated environments.

6.1.1 Key Accomplishments

Strategic Vision: Integration Over Individual Components

The project's value lies not in the individual technologies (Suricata, Jenkins, K3s are all mature tools), but in their integration into a cohesive security architecture. Proving these components can work together in an air-gapped, resource-constrained environment addresses a genuine industry need.

Security without Sacrificing Velocity:
Implemented automated CI/CD pipeline (git commit to production in 1-2 minutes) while simultaneously maintaining defense-in-depth security with air-gap isolation, active threat defense, and comprehensive logging.

Automation without Internet Dependency:
Built fully functional CI/CD pipeline operating in air-gapped environment through strategic use of controlled proxy access. Demonstrates that DevOps practices don't require always-connected cloud infrastructure.

Active Defense Implementation:
Automated threat detection and response system successfully blocks reconnaissance attacks without human intervention. While response time wasn't precisely measured, the system demonstrably blocks attacks faster than manual processes.

Resource Efficiency:
Entire stack (security monitoring, container orchestration, CI/CD, observability) operates

within 8GB RAM across two VMs, proving viability for edge deployments, branch offices, and development environments.

Practical Reference Architecture:
Provided working implementation integrating 15+ open-source technologies. Serves as blueprint for organizations implementing DevSecOps in regulated industries.

6.2 Technical Insights and Lessons Learned

6.2.1 Design Challenges

⚠ Critical: Air-Gap Bootstrap Paradox

Cannot configure proxy before proxy exists, creating circular dependency. Solution: staged approach (install with internet, then disable). Production environments would use pre-staged packages or internal mirrors to avoid this temporary compromise.

Systemd vs Shell Environments:
Shell exports don't propagate to systemd services. Learned to create dedicated service environment files for persistent proxy configuration across reboots.

Certificate Trust Complexity:
Browser trust depends on entire chain validation. Installing only Intermediate CA is insufficient; must install Root CA in system trust store. PKI hierarchy isn't just organizational structure; it's a cryptographic requirement.

Honeypot Placement Strategy:
Port 22 carries attacker expectations. Moving real SSH to non-standard port while running honeypot on 22 exploits attacker assumptions; security sometimes benefits from predictability.

Client-Side Caching:
Browser persistence of HTTP redirects often interfered with TLS validation testing, requiring strict use of Incognito/Private sessions to verify the "Green Lock" status after PKI deployment.

6.2.2 Design Trade-offs

Polling vs Webhooks:
Chose polling (1-minute delay) over webhooks (immediate) to maintain air-gap. Trade-off: deployment latency versus network isolation. Acceptable for project scope; production might require different balance.

Signature vs Anomaly Detection:
Chose signature-based IDS for deterministic behavior and resource efficiency. Trade-off: miss novel attacks versus avoid false positive chaos. Mitigated with honeypot for behavioral analysis.

Temporary vs Permanent Bans:
10-minute Fail2Ban timeout balances security (blocks attacks) versus availability (doesn't per-

manently lock out legitimate users behind NAT). Automated security must consider operational impact.

6.3 Limitations and Constraints

6.3.1 Architecture Limitations

◎ Key Objective: Honest Assessment

This is a proof-of-concept demonstrating integrated DevSecOps concepts, not a production-ready system. Understanding limitations is as important as celebrating achievements.

Single Point of Failure:
DMZ Bastion failure renders entire system inaccessible. No redundancy or failover. Acceptable for lab/development; unacceptable for production. Would require clustered bastions with load balancing.

Manual Certificate Management:
Certificate generation requires manual OpenSSL commands. No automated renewal (Let's Encrypt ACME unavailable in air-gap). Could implement internal ACME server but adds complexity.

No Multi-Tenancy:
Single application deployment. Production requires namespace isolation, RBAC, and network policies for multiple applications.

6.3.2 Performance Limitations

Resource Ceiling:
Single-node K3s limits vertical scaling to VM allocation. Can't handle massive traffic spikes. Would require multi-node cluster for true horizontal scaling.

Limited Testing Scope:
Testing demonstrated functionality with single-source attacks from Windows host. Real-world validation would require diverse attack vectors and load testing, which was beyond project scope.

6.4 Applicability to Real-World Scenarios

This architecture addresses genuine industry needs despite its limitations:

Financial Services:
Banks require air-gapped core banking systems. This architecture demonstrates automated deployments to isolated environments while maintaining regulatory compliance.

Healthcare:
HIPAA mandates strict access controls and audit trails. Defense-in-depth model with comprehensive logging satisfies compliance requirements.

Government:
Classified networks demand physical isolation. Controlled proxy approach allows essential updates while preventing data exfiltration.

Industrial Control Systems (ICS/SCADA):
Critical infrastructure operates in isolated networks. Architecture demonstrates secure remote management without exposing control systems to internet.

Edge Computing:
Branch offices and remote sites often operate on limited resources. Stack running in 8GB proves viability for distributed deployments.

6.5 Future Enhancements

Potential improvements for production deployment:

High Availability: Clustered DMZ bastions with load balancer and shared state. Multi-node K3s cluster with distributed etcd for resilience.

Advanced Security: Web Application Firewall (ModSecurity), threat intelligence integration (VirusTotal, AbuseIPDB), GeoIP blocking for risk reduction.

Enhanced Observability: Distributed tracing (Jaeger), SIEM integration (Wazuh), compliance reporting for regulatory requirements.

Operational Improvements: GitOps workflow (ArgoCD), automated backups (Velero), certificate automation (internal ACME server).

6.6 Closing Remarks

The DevSecOps philosophy; "security as code, built into the pipeline"; often seems incompatible with isolation requirements. This project demonstrates the opposite: proper architecture enables both.

Air-gaps need not mean manual processes. Active defense doesn't require enterprise budgets. Enterprise PKI isn't exclusive to large organizations. With thoughtful integration of open-source tools, even resource-constrained environments can achieve robust security postures.

Strategic Vision: Final Insight

The future of infrastructure security lies not in adding security after development, but in architecting systems where security enables velocity. This project demonstrates that future is achievable today with open-source tools and sound architectural principles.

This project contributes a reference implementation proving these concepts work in practice, not just theory. The architecture scales from development laptops to production data centers, from single applications to enterprise platforms. Most importantly, it demonstrates that security and automation are complementary forces when designed holistically from the start.

Appendix A

Screenshots

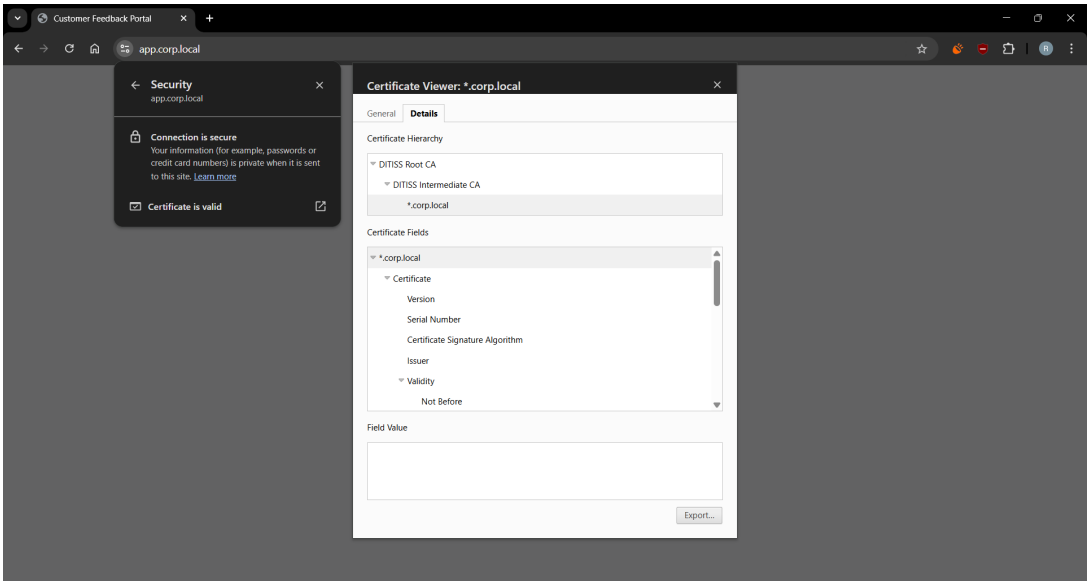


Figure A.1: Verification of the 3-Tier Certificate Authority (CA) hierarchy. The browser establishes a secure HTTPS connection to app.corp.local by validating the chain of trust from the DITISS Root CA through the Intermediate CA to the wildcard server certificate.

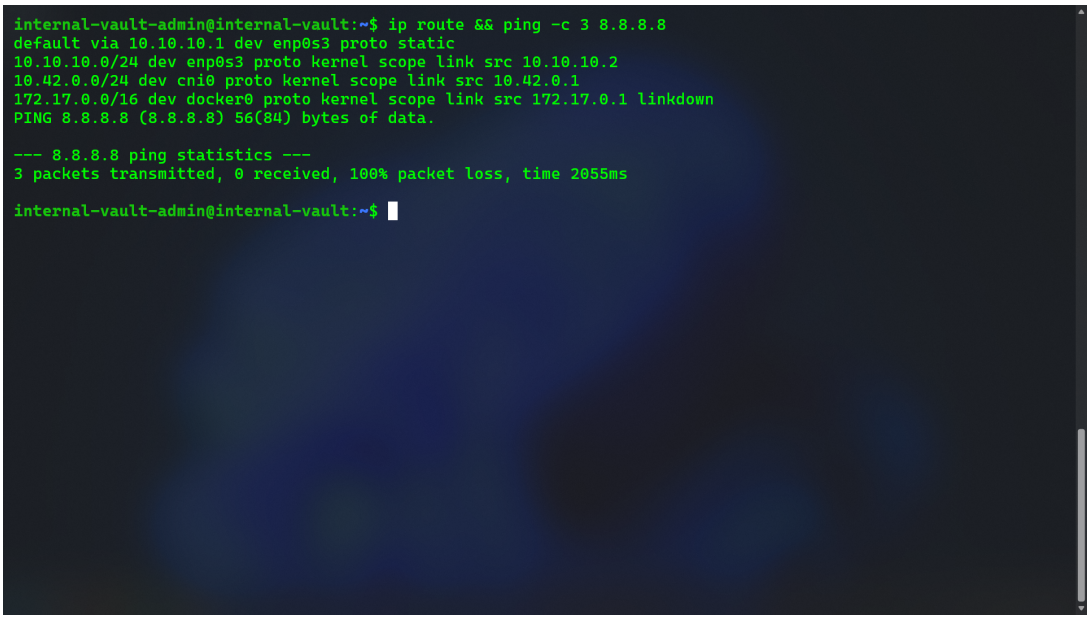


Figure A.2: Verification of network isolation on the Internal Vault. The routing table shows traffic is directed through the DMZ Bastion (10.10.10.1), while the failed ping to 8.8.8.8 (100% packet loss) confirms the absence of direct internet connectivity.

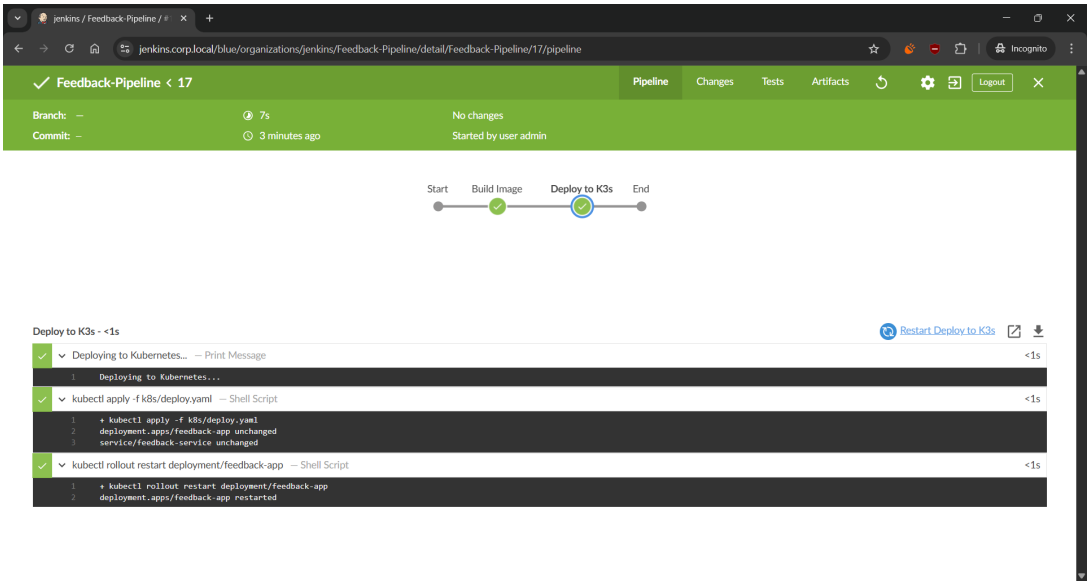


Figure A.3: Visual representation of the automated Jenkins pipeline. This demonstrates the "Zero-Touch" deployment of the Feedback Application, transitioning seamlessly from the build stage to K3s container orchestration within the isolated Internal Vault.

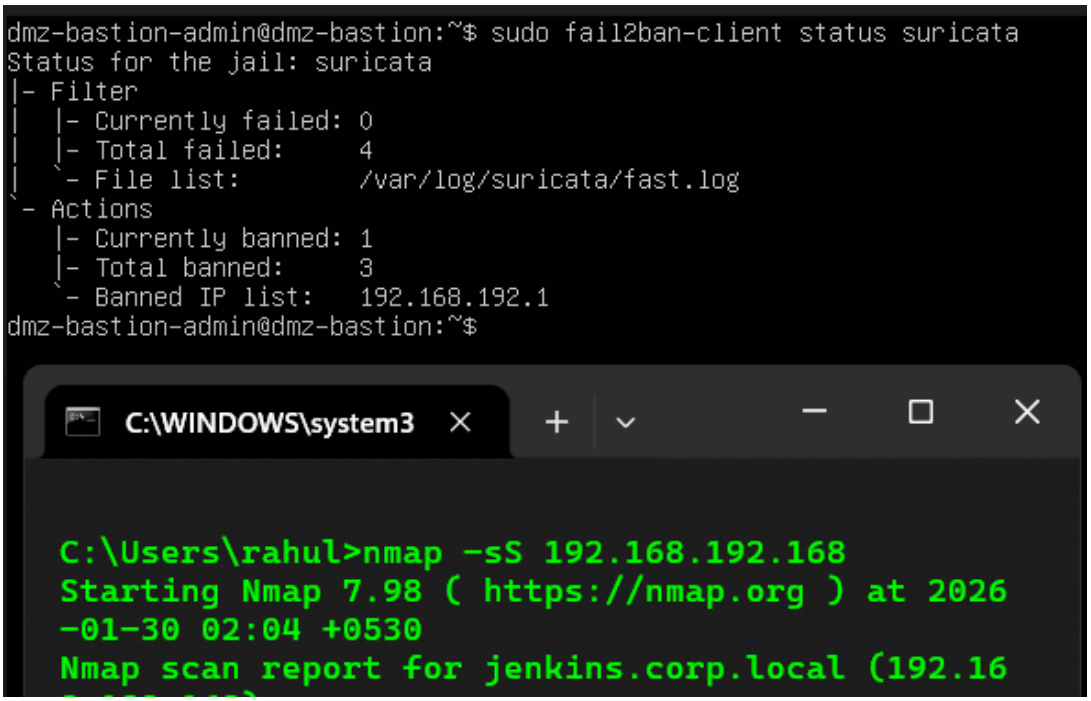


Figure A.4: Active defense mechanism in operation. The bottom window shows an incoming Nmap reconnaissance scan from the Windows host, while the top window demonstrates the autonomous response by Fail2Ban, which has identified and jailed the attacker's IP (192.168.192.1) in real-time.

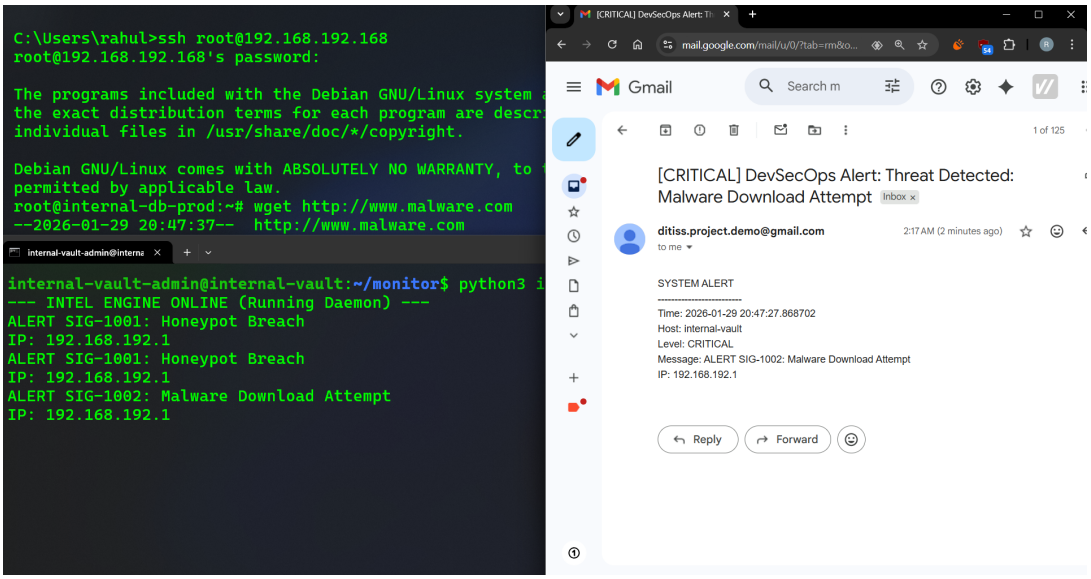


Figure A.5: Integrated detection and notification flow. An attacker's malware download attempt in the honeypot (top-left) is correlated by the custom Python Intel Engine (bottom-left), which triggers a high-severity [CRITICAL] email alert (right) sent via the controlled SMTP gateway.

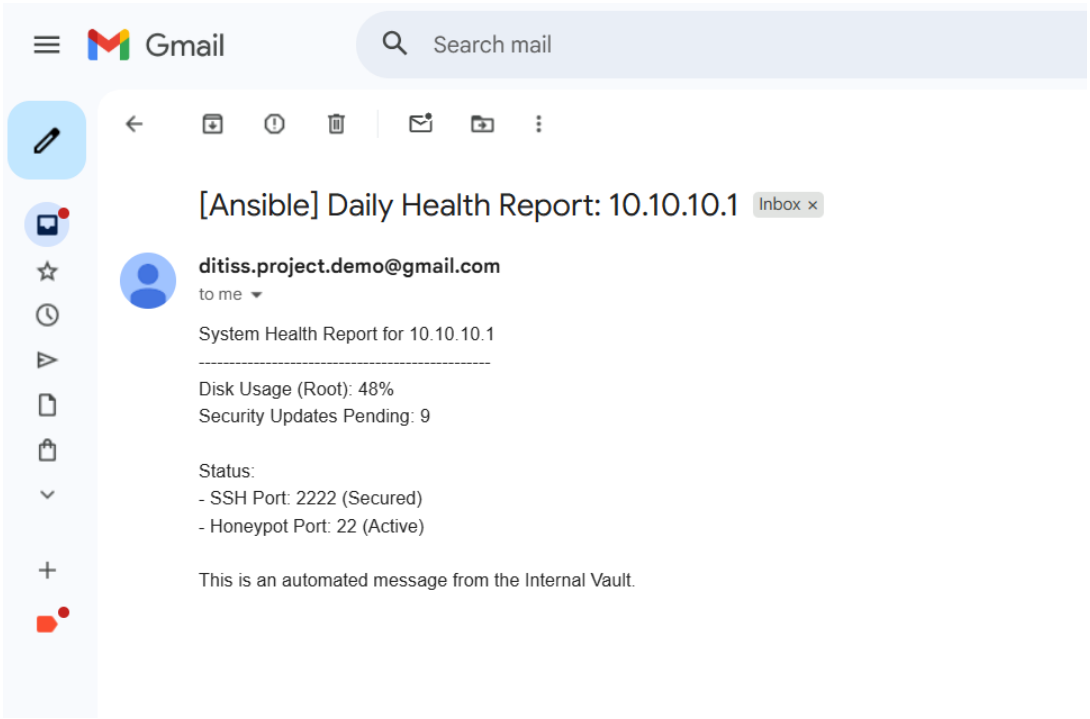


Figure A.6: Demonstration of automated system administration. An Ansible-driven daily health report is dispatched from the Internal Vault to the administrator, providing status updates on disk usage, pending security patches, and service availability for the DMZ Bastion.

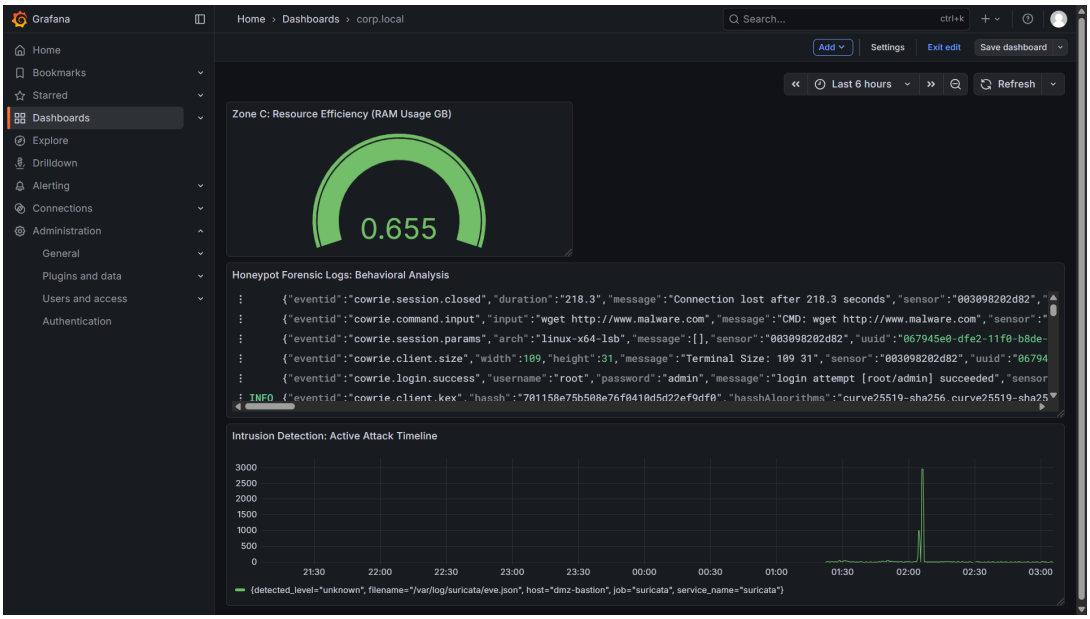


Figure A.7: Comprehensive observability stack in Grafana. This single-pane-of-glass view aggregates system resource metrics (0.655GB RAM usage), forensic honeypot logs, and intrusion detection timelines, proving the system's efficiency and visibility under resource-constrained conditions.