

Secure Internet-Facing Application Deployment Guide

High-Assurance Security Implementation
for Public Production Workloads

*Companion to Comprehensive Cloud-Native
Architecture Implementation Guide*

Defense-in-Depth Security Controls
Edge Security • API Protection • DDoS Mitigation
WAF • Rate Limiting • Zero-Trust Architecture
Threat Detection • Incident Response

Technical Reference Guide

Production-Ready Internet-Facing Deployments

January 19, 2026

Contents

1 Executive Summary	2
1.1 Document Purpose	2
1.2 Key Insight: Public Internet Changes Everything	2
1.3 Relationship to Companion Guides	3
1.4 Scope and Target Audience	3
1.5 Security Maturity Phases	3
2 Official Documentation Resources	4
2.1 Edge and Ingress Security	4
2.1.1 Kubernetes Ingress Controllers	4
2.1.2 Web Application Firewalls	5
2.1.3 Cloud Provider Edge Services	5
2.2 TLS and Certificate Management	6
2.3 API Security	6
2.4 DDoS Protection and Rate Limiting	7
2.5 Service Mesh Security	7
2.6 Secrets Management	7
2.7 Security Scanning and Vulnerability Management	8
2.8 Monitoring, Logging, and Observability	8
2.9 Policy and Compliance	9
2.10 Incident Response	9
3 Phase 1: Protected Pilot (Weeks 1-4)	9
3.1 Overview	9
3.2 Core Security Controls	10
3.2.1 Network Perimeter Protection	10
3.2.2 TLS/SSL Configuration	10
3.2.3 Basic Web Application Firewall	12
3.2.4 Kubernetes Security Baseline	12
3.2.5 Secrets Management	13
3.2.6 Authentication and Authorization	14
3.2.7 Logging and Monitoring	15
3.3 Phase 1 Go/No-Go Checklist	17
4 Phase 2: Controlled Public Access (Weeks 5-8)	18
4.1 Overview	18
4.2 Enhanced Security Controls	18
4.2.1 Cloud-Native DDoS Protection	18
4.2.2 Production-Grade WAF Configuration	19
4.2.3 Comprehensive Rate Limiting	20
4.2.4 Advanced Authentication and Authorization	21
4.2.5 Container Image Security	22
4.2.6 Service Mesh Security	24
4.3 Phase 2 Go/No-Go Checklist	26

5 Phase 3: Production Internet Deployment (Weeks 9-12)	27
5.1 Overview	27
5.2 Advanced Security Controls	27
5.2.1 Runtime Security and Threat Detection	27
5.2.2 Advanced Secrets Management	28
5.2.3 Comprehensive Security Monitoring	29
5.2.4 Security Policy Enforcement	31
5.2.5 Compliance and Audit Controls	33
5.2.6 Incident Response Capabilities	35
5.3 Disaster Recovery and Business Continuity	37
5.4 Phase 3 Go/No-Go Checklist	38
6 Defense-in-Depth Architecture	39
6.1 Security Layer Model	39
6.2 Zero-Trust Principles	39
6.3 Attack Surface Reduction	40
7 API Security Best Practices	40
7.1 API Authentication Methods	40
7.2 API Rate Limiting Strategies	41
7.3 API Security Headers	41
7.4 Input Validation and Sanitization	42
8 Security Testing and Validation	42
8.1 Automated Security Testing in CI/CD	42
8.2 Penetration Testing	43
8.3 Security Chaos Engineering	44
9 Compliance and Regulatory Requirements	44
9.1 Compliance Framework Mapping	44
9.2 Audit Trail Requirements	46
10 Continuous Security Improvement	46
10.1 Security Metrics and KPIs	46
10.2 Security Review Cadence	47
10.3 Threat Intelligence Integration	48
11 Conclusion	48
11.1 Critical Success Factors	48
11.2 Common Pitfalls to Avoid	49
11.3 Roadmap Beyond Phase 3	49
11.4 Final Checklist	50
11.5 Resources and Community	50

1 Executive Summary

This guide provides a comprehensive security implementation framework for deploying internet-facing applications with high assurance requirements. It serves as a companion to the *Comprehensive Cloud-Native Architecture Implementation Guide* and the *Secure Internal Application Hosting Guide*, focusing specifically on the additional security controls, threat mitigations, and operational practices required for public-facing production workloads.

1.1 Document Purpose

This implementation guide serves as:

1. **Security Hardening Roadmap:** Progressive maturity from protected pilot to production-ready internet deployment
2. **Defense-in-Depth Framework:** Layered security controls from edge to application to data
3. **Threat Mitigation Playbook:** Specific countermeasures for internet-facing attack vectors
4. **Compliance Accelerator:** Controls aligned with SOC 2, PCI-DSS, HIPAA, and GDPR requirements
5. **Operational Security Guide:** Monitoring, incident response, and continuous validation procedures

1.2 Key Insight: Public Internet Changes Everything

Critical Security Requirement

Critical Security Principle:

Internet-facing applications face fundamentally different threat models than internal deployments:

- **Constant Attack Surface:** Automated scanning, credential stuffing, exploit attempts 24/7
- **Anonymized Attackers:** No authentication required for reconnaissance and initial attacks
- **DDoS Vulnerability:** Application and infrastructure overwhelm attacks
- **Zero Trust Required:** Every request must be validated; assume breach at every layer
- **Regulatory Scrutiny:** Compliance frameworks mandate specific controls for public systems

Internet-facing deployments require defense-in-depth at edge, application, and data layers simultaneously.

1.3 Relationship to Companion Guides

This guide builds upon:

- **Cloud-Native Architecture Guide:** Assumes Kubernetes, service mesh, and cloud platform knowledge
- **Internal Application Hosting Guide:** Inherits all zero-trust controls and adds internet-specific protections

Critical Distinction: This guide focuses on *additional* security controls required when applications face the public internet. All internal security controls remain mandatory; internet-facing requires layered defenses beyond those baselines.

1.4 Scope and Target Audience

In Scope:

- Internet-facing web applications and APIs
- Public-facing microservices architectures
- SaaS and customer-facing platforms
- High-assurance production deployments
- Compliance-regulated internet services
- Multi-tenant internet applications

Out of Scope:

- Internal-only applications (see Internal Application Hosting Guide)
- Development or staging environments without production data
- Mobile app backends (covered partially; requires additional mobile-specific controls)
- IoT device management (requires specialized IoT security controls)

Target Audience:

- Security engineers implementing internet-facing defenses
- Platform engineers deploying public production workloads
- DevSecOps teams integrating security into CI/CD pipelines
- Compliance officers validating regulatory requirements
- SRE teams responsible for internet service availability and resilience

1.5 Security Maturity Phases

This guide defines three security maturity phases for internet-facing deployments:

Critical Note: Each phase builds cumulatively. Phase 3 includes all Phase 1 and Phase 2 controls plus additional hardening. Never skip phases for internet-facing deployments.

Phase	Timeframe	Readiness Level
Phase 1	Weeks 1-4	Protected Pilot IP whitelisting, basic WAF, limited user base Controlled internet exposure for testing
Phase 2	Weeks 5-8	Controlled Public Access Full WAF, DDoS protection, rate limiting Beta/limited production with monitoring
Phase 3	Weeks 9-12	Production Internet Deployment Complete defense-in-depth, threat detection Full compliance, incident response capabilities

Table 1: Internet-Facing Security Maturity Phases

2 Official Documentation Resources

The following official documentation provides authoritative references for implementing internet-facing security controls. These resources should be used alongside this guide for detailed implementation guidance.

2.1 Edge and Ingress Security

2.1.1 Kubernetes Ingress Controllers

NGINX Ingress Controller:

- Main Documentation: <https://kubernetes.github.io/ingress-nginx/>
- User Guide: <https://kubernetes.github.io/ingress-nginx/user-guide/>
- Rate Limiting: <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#rate-limiting>
- ModSecurity WAF: [https://kubernetes.github.io/ingress-nginx/user-guide/third-party-addons/modsecurity/](https://kubernetes.github.io/ingress-nginx/user-guide/third-party-addons/modsecurity)
- TLS Configuration: <https://kubernetes.github.io/ingress-nginx/user-guide/tls/>
- Security Best Practices: <https://kubernetes.github.io/ingress-nginx/deploy/#security-considerations>

Traefik:

- Main Documentation: <https://doc.traefik.io/traefik/>
- Middlewares: <https://doc.traefik.io/traefik/middlewares/overview/>
- Rate Limiting: <https://doc.traefik.io/traefik/middlewares/http/ratelimit/>
- IP Whitelisting: <https://doc.traefik.io/traefik/middlewares/http/ipwhitelist/>
- HTTPS Configuration: <https://doc.traefik.io/traefik/https/overview/>

Contour:

- Main Documentation: <https://projectcontour.io/docs/>
- TLS Configuration: <https://projectcontour.io/docs/main/config/tls-termination/>
- Rate Limiting: <https://projectcontour.io/docs/main/config/rate-limiting/>

2.1.2 Web Application Firewalls

ModSecurity:

- Main Documentation: <https://github.com/SpiderLabs/ModSecurity>
- OWASP Core Rule Set: <https://owasp.org/www-project-modsecurity-core-rule-set/>
- CRS Documentation: <https://coreruleset.org/docs/>
- Rule Customization: <https://coreruleset.org/docs/configuring/>

Coraza WAF:

- Main Documentation: <https://coraza.io/docs/>
- Kubernetes Integration: <https://github.com/corazawaf/coraza-proxy-wasm>

2.1.3 Cloud Provider Edge Services

AWS:

- CloudFront: <https://docs.aws.amazon.com/cloudfront/>
- AWS WAF: <https://docs.aws.amazon.com/waf/>
- WAF Rules: <https://docs.aws.amazon.com/waf/latest/developerguide/waf-rules.html>
- Shield (DDoS): <https://docs.aws.amazon.com/shield/>
- Shield Advanced: <https://docs.aws.amazon.com/waf/latest/developerguide/shield-chapter.html>
- Application Load Balancer: <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/>

Google Cloud:

- Cloud CDN: <https://cloud.google.com/cdn/docs>
- Cloud Armor: <https://cloud.google.com/armor/docs>
- Cloud Load Balancing: <https://cloud.google.com/load-balancing/docs>
- Security Policies: <https://cloud.google.com/armor/docs/security-policy-overview>

Azure:

- Front Door: <https://docs.microsoft.com/azure/frontdoor/>
- Application Gateway: <https://docs.microsoft.com/azure/application-gateway/>
- WAF Overview: <https://docs.microsoft.com/azure/web-application-firewall/>
- DDoS Protection: <https://docs.microsoft.com/azure/ddos-protection/>

2.2 TLS and Certificate Management

cert-manager:

- Main Documentation: <https://cert-manager.io/docs/>
- Installation: <https://cert-manager.io/docs/installation/>
- ACME (Let's Encrypt): <https://cert-manager.io/docs/configuration/acme/>
- Certificate Resources: <https://cert-manager.io/docs/usage/certificate/>
- Securing Ingress: <https://cert-manager.io/docs/usage/ingress/>

Let's Encrypt:

- Main Documentation: <https://letsencrypt.org/docs/>
- Rate Limits: <https://letsencrypt.org/docs/rate-limits/>
- Best Practices: <https://letsencrypt.org/docs/best-practice/>

2.3 API Security

API Gateway Solutions:

- Kong Gateway: <https://docs.konghq.com/gateway/>
- Kong Rate Limiting: <https://docs.konghq.com/hub/kong-inc/rate-limiting/>
- Kong Security Plugins: <https://docs.konghq.com/hub/?category=security>
- Tyk API Gateway: <https://tyk.io/docs/>
- KrakenD: <https://www.krakend.io/docs/>

OAuth 2.0 and OIDC:

- OAuth 2.0 Specification: <https://oauth.net/2/>
- OpenID Connect: <https://openid.net/connect/>
- OAuth 2.0 Security Best Practices: <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>
- Keycloak Documentation: <https://www.keycloak.org/documentation>
- Dex (OIDC Provider): <https://dexidp.io/docs/>

2.4 DDoS Protection and Rate Limiting

Rate Limiting Tools:

- Envoy Rate Limiting: https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/rate_limit_filter
- Redis Rate Limiter: <https://redis.io/docs/manual/patterns/rate-limiter/>
- Nginx Rate Limiting: <https://www.nginx.com/blog/rate-limiting-nginx/>

2.5 Service Mesh Security

Istio:

- Security Concepts: <https://istio.io/latest/docs/concepts/security/>
- Authorization Policies: <https://istio.io/latest/docs/tasks/security/authorization/>
- Request Authentication: <https://istio.io/latest/docs/tasks/security/authentication/authn-policy/>
- Mutual TLS: <https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/>
- Ingress Gateway: <https://istio.io/latest/docs/tasks/traffic-management/ingress/>
- Security Best Practices: <https://istio.io/latest/docs/ops/best-practices/security/>

Linkerd:

- Automatic mTLS: <https://linkerd.io/2/features/automatic-mtls/>
- Server Policies: <https://linkerd.io/2/features/server-policy/>
- Authorization Policy: <https://linkerd.io/2/reference/authorization-policy/>

2.6 Secrets Management

HashiCorp Vault:

- Main Documentation: <https://www.vaultproject.io/docs>
- Kubernetes Auth: <https://www.vaultproject.io/docs/auth/kubernetes>
- Dynamic Secrets: <https://www.vaultproject.io/docs/secrets>
- Transit Secrets Engine: <https://www.vaultproject.io/docs/secrets/transit>
- PKI Secrets Engine: <https://www.vaultproject.io/docs/secrets/pki>

External Secrets Operator:

- Main Documentation: <https://external-secrets.io/>
- Cloud Provider Backends: <https://external-secrets.io/latest/provider/aws-secrets-manager/>

2.7 Security Scanning and Vulnerability Management

Container Scanning:

- Trivy: <https://aquasecurity.github.io/trivy/>
- Grype: <https://github.com/anchore/grype>
- Clair: <https://quay.github.io/clair/>
- Snyk Container: <https://snyk.io/product/container-vulnerability-management/>

Image Signing and Verification:

- Sigstore Cosign: <https://docs.sigstore.dev/cosign/overview/>
- Notary Project: <https://notaryproject.dev>
- Kyverno Image Verification: <https://kyverno.io/docs/writing-policies/verify-image-s/>

Runtime Security:

- Falco: <https://falco.org/docs/>
- Falco Rules: <https://github.com/falcosecurity/rules>
- Tetragon (eBPF): <https://tetragon.io/docs/>

2.8 Monitoring, Logging, and Observability

Prometheus and Alerting:

- Prometheus Documentation: <https://prometheus.io/docs/>
- Security Best Practices: <https://prometheus.io/docs/operating/security/>
- Alertmanager: <https://prometheus.io/docs/alerting/latest/alertmanager/>

Log Aggregation:

- Loki: <https://grafana.com/docs/loki/latest/>
- Fluentd: <https://docs.fluentd.org/>
- Fluent Bit: <https://docs.fluentbit.io/manual/>

Distributed Tracing:

- Jaeger: <https://www.jaegertracing.io/docs/>
- Tempo: <https://grafana.com/docs/tempo/latest/>

2.9 Policy and Compliance

Policy Enforcement:

- OPA/Gatekeeper: <https://open-policy-agent.github.io/gatekeeper/>
- Kyverno: <https://kyverno.io/docs/>
- Kyverno Policy Library: <https://kyverno.io/policies/>

Compliance Frameworks:

- CIS Kubernetes Benchmark: <https://www.cisecurity.org/benchmark/kubernetes>
- NSA Kubernetes Hardening Guide: https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF
- NIST SP 800-190: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>

2.10 Incident Response

Security Incident Management:

- NIST Incident Response Guide: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf>
- OWASP Incident Response: https://owasp.org/www-community/Incident_Response

3 Phase 1: Protected Pilot (Weeks 1-4)

3.1 Overview

Security Posture: Controlled internet exposure with IP whitelisting and basic protections for pilot testing.

Acceptable Risk Profile:

- Limited known IP addresses (corporate network, test users)
- Non-production or pilot data only
- Ability to immediately block all external traffic if needed
- Extensive monitoring and alerting on all access

Security Warning

Phase 1 Limitation:

This phase is NOT suitable for unrestricted public access. IP whitelisting is the primary control. Any expansion beyond known IPs requires advancing to Phase 2.

3.2 Core Security Controls

3.2.1 Network Perimeter Protection

IP Whitelisting:

- Implement strict IP allowlists at ingress controller level
- Document all whitelisted IP ranges and their business justification
- Establish change control process for IP list modifications
- Monitor for connection attempts from non-whitelisted IPs

Example: NGINX Ingress IP Whitelist:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: protected-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: |
      203.0.113.0/24,
      198.51.100.0/24
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - pilot.example.com
    secretName: pilot-tls
  rules:
  - host: pilot.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-service
            port:
              number: 80
```

Official documentation: <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#whitelist-source-range>

3.2.2 TLS/SSL Configuration

Certificate Management with cert-manager:

- Deploy cert-manager with ACME (Let's Encrypt) or internal CA
- Automate certificate issuance and renewal
- Enforce TLS 1.2 minimum, prefer TLS 1.3
- Configure strong cipher suites only

Example: cert-manager Let's Encrypt Configuration:

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: security@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: nginx
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pilot-tls
  namespace: pilot-app
spec:
  secretName: pilot-tls
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
  dnsNames:
    - pilot.example.com

```

Official documentation: <https://cert-manager.io/docs/configuration/acme/TLS Configuration Best Practices>:

- Minimum TLS 1.2, prefer TLS 1.3
- Disable weak ciphers (RC4, 3DES, MD5)
- Enable HSTS (HTTP Strict Transport Security)
- Configure secure cipher suites

Example: NGINX Ingress TLS Hardening:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
data:
  ssl-protocols: "TLSv1.2 TLSv1.3"
  ssl-ciphers: "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:
                 ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384"
  ssl-prefer-server-ciphers: "true"
  hsts: "true"
  hsts-max-age: "31536000"
  hsts-include-subdomains: "true"

```

```
hsts-preload: "true"
```

Official documentation: <https://kubernetes.github.io/ingress-nginx/user-guide/tls/>

3.2.3 Basic Web Application Firewall

ModSecurity with OWASP Core Rule Set:

- Deploy ModSecurity with NGINX Ingress Controller
- Enable OWASP Core Rule Set (CRS) in detection mode initially
- Monitor for false positives before enforcing blocking mode
- Tune rules based on application behavior

Example: ModSecurity Configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
data:
  enable-modsecurity: "true"
  enable-owasp-modsecurity-crs: "true"
  modsecurity-snippet: |
    SecRuleEngine DetectionOnly
    SecAuditLog /var/log/modsec_audit.log
    SecAuditLogFormat JSON
```

Official documentation: <https://kubernetes.github.io/ingress-nginx/user-guide/third-party-addons/modsecurity/>

3.2.4 Kubernetes Security Baseline

Pod Security Standards:

- Enforce **restricted** Pod Security Standard for application namespaces
- Document any exemptions with security review
- Use security contexts to drop capabilities and run as non-root

Example: Pod Security Standard Enforcement:

```
apiVersion: v1
kind: Namespace
metadata:
  name: pilot-app
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

Official documentation: <https://kubernetes.io/docs/concepts/security/pod-security-standards/>

Network Policies:

- Implement default-deny network policies
- Allow only required ingress from ingress controller
- Allow only required egress (DNS, external APIs)
- Deny all other traffic by default

Example: Default Deny Network Policy:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: pilot-app
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-controller
  namespace: pilot-app
spec:
  podSelector:
    matchLabels:
      app: web-app
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              name: ingress-nginx
  ports:
    - protocol: TCP
      port: 8080
```

Official documentation: <https://kubernetes.io/docs/concepts/services-networking/network-policies/>

3.2.5 Secrets Management

External Secrets Operator with Cloud Provider:

- Never store secrets in Git or Kubernetes Secrets directly
- Use External Secrets Operator with AWS Secrets Manager, GCP Secret Manager, or Azure Key Vault

- Implement secret rotation policies (30-90 days)
- Audit all secret access

Example: External Secrets Operator Configuration:

```
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: aws-secretsmanager
  namespace: pilot-app
spec:
  provider:
    aws:
      service: SecretsManager
      region: us-east-1
      auth:
        jwt:
          serviceAccountRef:
            name: external-secrets-sa
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: database-credentials
  namespace: pilot-app
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: aws-secretsmanager
    kind: SecretStore
  target:
    name: db-credentials
    creationPolicy: Owner
  data:
    - secretKey: password
      remoteRef:
        key: pilot-app/database
        property: password
```

Official documentation: <https://external-secrets.io/latest/introduction/getting-started/>

3.2.6 Authentication and Authorization

OAuth 2.0 / OIDC Integration:

- Implement OAuth 2.0 for API authentication
- Use OIDC for web application authentication
- Integrate with corporate identity provider (Okta, Auth0, Keycloak)
- Enforce MFA for all user access

Example: OAuth2 Proxy with OIDC:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oauth2-proxy
  namespace: pilot-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: oauth2-proxy
  template:
    metadata:
      labels:
        app: oauth2-proxy
    spec:
      containers:
        - name: oauth2-proxy
          image: quay.io/oauth2-proxy/oauth2-proxy:v7.5.1
          args:
            - --provider=oidc
            - --oidc-issuer-url=https://auth.example.com
            - --upstream=http://app-service:80
            - --http-address=0.0.0.0:4180
            - --email-domain=example.com
            - --cookie-secure=true
            - --cookie-samesite=lax
          env:
            - name: OAUTH2_PROXY_CLIENT_ID
              valueFrom:
                secretKeyRef:
                  name: oauth2-proxy-secrets
                  key: client-id
            - name: OAUTH2_PROXY_CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  name: oauth2-proxy-secrets
                  key: client-secret
            - name: OAUTH2_PROXY_COOKIE_SECRET
              valueFrom:
                secretKeyRef:
                  name: oauth2-proxy-secrets
                  key: cookie-secret
```

Official documentation: <https://oauth2-proxy.github.io/oauth2-proxy/>

3.2.7 Logging and Monitoring

Security Event Logging:

- Enable Kubernetes audit logging for all API requests
- Centralize logs with Loki or cloud provider logging service
- Retain logs for minimum 90 days

- Configure alerts for security events

Monitoring and Alerting:

- Deploy Prometheus and Grafana
- Configure alerts for:
 - Non-whitelisted IP connection attempts
 - Failed authentication attempts (>5 per minute)
 - WAF rule violations
 - Certificate expiration warnings (30 days)
 - Anomalous traffic patterns
- Establish on-call rotation for security alerts

Example: Prometheus Alert Rules:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-alerts
  namespace: monitoring
data:
  security-alerts.yaml: |
    groups:
    - name: security
      interval: 30s
      rules:
      - alert: UnauthorizedAccess
        expr: rate(nginx_ingress_controller_requests{status="403"}[5m]) >
          10
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High rate of 403 responses"
          description: "Potential unauthorized access attempts detected"

      - alert: CertificateExpiringSoon
        expr: (certmanager_certificate_expiration_timestamp_seconds - time
          ()) / 86400 < 30
        for: 1h
        labels:
          severity: warning
        annotations:
          summary: "Certificate expiring in less than 30 days"
```

Official documentation: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

3.3 Phase 1 Go/No-Go Checklist

Go/No-Go Checkpoint

Before enabling internet access in Phase 1, verify:
Network Security:

- IP whitelist configured and tested
- TLS certificates issued and auto-renewing
- TLS 1.2+ enforced, weak ciphers disabled
- HSTS headers configured

Application Security:

- ModSecurity with OWASP CRS enabled (detection mode)
- Pod Security Standards enforced (restricted level)
- Network policies: default-deny implemented
- OAuth2/OIDC authentication configured

Secrets and Credentials:

- External Secrets Operator configured
- No secrets in Git repositories
- Secret rotation policy documented

Monitoring and Response:

- Kubernetes audit logging enabled
- Centralized logging configured (90-day retention)
- Security alerts configured and tested
- On-call rotation established
- Incident response runbook created

Documentation:

- Whitelisted IPs documented with justification
- Architecture diagram created
- Runbook for adding/removing IPs from whitelist
- Emergency shutdown procedure documented

4 Phase 2: Controlled Public Access (Weeks 5-8)

4.1 Overview

Security Posture: Full WAF protection, DDoS mitigation, and rate limiting for controlled public access (beta/limited production).

Acceptable Risk Profile:

- Limited public user base (beta users, limited launch)
- Production-like data with comprehensive monitoring
- Ability to implement emergency rate limiting or blocking
- 24/7 security monitoring and incident response

Critical Security Requirement

Phase 2 Transition:

Removing IP whitelisting exposes application to internet-wide attacks. All Phase 2 controls must be implemented BEFORE removing IP restrictions. Never partially deploy Phase 2 controls.

4.2 Enhanced Security Controls

4.2.1 Cloud-Native DDoS Protection

Cloud Provider DDoS Services:

- Enable AWS Shield Standard (automatic for all AWS customers)
- Consider AWS Shield Advanced for Layer 7 protection
- Alternatively: Google Cloud Armor or Azure DDoS Protection
- Configure automatic scaling for DDoS absorption

AWS Shield Standard Configuration:

- Automatically enabled for CloudFront and Route 53
- Protects against common Layer 3/4 DDoS attacks
- No additional configuration required
- Monitor via CloudWatch metrics

Official documentation: <https://docs.aws.amazon.com/shield/latest/developerguide/what-is-aws-shield.html>

Google Cloud Armor:

```
# Create security policy
gcloud compute security-policies create internet-app-policy \
--description "DDoS and WAF protection for internet-facing app"

# Add rate limiting rule
gcloud compute security-policies rules create 1000 \
--security-policy internet-app-policy \
--expression "true" \
--action "rate-based-ban" \
--rate-limit-threshold-count 100 \
--rate-limit-threshold-interval-sec 60 \
--ban-duration-sec 600 \
--conform-action allow \
--exceed-action deny -403
```

Official documentation: <https://cloud.google.com/armor/docs/security-policy-overview>

4.2.2 Production-Grade WAF Configuration

ModSecurity Transition to Blocking Mode:

- Review detection mode logs for false positives
- Create exception rules for legitimate traffic patterns
- Enable blocking mode for OWASP CRS rules
- Implement graduated response (log, rate-limit, block)

Example: ModSecurity Blocking Mode:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
data:
  enable-modsecurity: "true"
  enable-owasp-modsecurity-crs: "true"
  modsecurity-snippet: |
    SecRuleEngine On
    SecAuditEngine RelevantOnly
    SecAuditLog /var/log/modsec_audit.log
    SecAuditLogFormat JSON

    # Custom rule to allow legitimate traffic
    SecRule REQUEST_URI "@streq /healthz" \
      "id:1000,phase:1,pass,nolog,ctl:ruleEngine=Off"

    # Set paranoia level (1-4, higher = stricter)
    SecAction "id:900000,phase:1,nolog,pass,\\
      t:none,setvar:tx.paranoia_level=2"
```

OWASP CRS Protection Coverage:

- SQL Injection (SQLi) protection
- Cross-Site Scripting (XSS) prevention
- Local/Remote File Inclusion detection
- Command Injection blocking
- Protocol attack prevention
- Malicious automation detection

Official documentation: <https://coreruleset.org/docs/>

4.2.3 Comprehensive Rate Limiting

Multi-Layer Rate Limiting Strategy:

1. **Edge/CDN Layer:** Coarse-grained rate limiting (1000s req/min)
2. **Ingress Layer:** Per-path rate limiting (100s req/min)
3. **Application Layer:** Per-user/API key rate limiting (10s req/min)

Example: NGINX Ingress Rate Limiting:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: rate-limited-ingress
  annotations:
    # Global rate limit: 100 requests per second from same IP
    nginx.ingress.kubernetes.io/limit-rps: "100"

    # Burst allowance
    nginx.ingress.kubernetes.io/limit-burst-multiplier: "5"

    # Rate limit by client IP
    nginx.ingress.kubernetes.io/limit-rate-after: "100"
    nginx.ingress.kubernetes.io/limit-rate: "500"

    # Connection limits
    nginx.ingress.kubernetes.io/limit-connections: "10"
spec:
  ingressClassName: nginx
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 80
```

Official documentation: <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#rate-limiting>

API-Specific Rate Limiting with Kong:

```
apiVersion: configuration.konghq.com/v1
kind: KongPlugin
metadata:
  name: api-rate-limit
  namespace: production
plugin: rate-limiting
config:
  minute: 100
  hour: 5000
  policy: redis
  redis_host: redis.default.svc.cluster.local
  redis_port: 6379
  fault_tolerant: true
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    konghq.com/plugins: api-rate-limit
spec:
  ingressClassName: kong
  rules:
  - host: api.example.com
    http:
      paths:
      - path: /v1
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 80
```

Official documentation: <https://docs.konghq.com/hub/kong-inc/rate-limiting/>

4.2.4 Advanced Authentication and Authorization

API Key Management:

- Implement API key rotation (90-day lifecycle)
- Hash API keys in storage (never store plaintext)
- Rate limit per API key
- Monitor for compromised keys (unusual geographic access, rate spikes)

JWT Token Validation:

- Validate JWT signatures at ingress layer

- Enforce short token expiration (15 minutes access, 7 days refresh)
- Implement token revocation list for compromised tokens
- Use RS256 or ES256 (not HS256 for production)

Example: Istio JWT Validation:

```

apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-validation
  namespace: production
spec:
  selector:
    matchLabels:
      app: api-service
  jwtRules:
    - issuer: "https://auth.example.com"
      jwksUri: "https://auth.example.com/.well-known/jwks.json"
      audiences:
        - "api.example.com"
      forwardOriginalToken: true
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: production
spec:
  selector:
    matchLabels:
      app: api-service
  action: DENY
  rules:
    - from:
        - source:
            notRequestPrincipals: ["*"]

```

Official documentation: <https://istio.io/latest/docs/tasks/security/authentication/authn-policy/>

4.2.5 Container Image Security

Image Scanning in CI/CD:

- Scan all images for vulnerabilities before deployment
- Fail builds with HIGH or CRITICAL vulnerabilities
- Generate SBOM (Software Bill of Materials)
- Block images without valid signatures

Example: Trivy Scanning in CI/CD:

```
# Scan image for vulnerabilities
trivy image --severity HIGH,CRITICAL \
--exit-code 1 \
myapp:latest

# Generate SBOM
trivy image --format cyclonedx \
--output sbom.json \
myapp:latest
```

Official documentation: <https://aquasecurity.github.io/trivy/>

Image Signing and Verification:

- Sign all production images with Cosign
- Verify signatures at admission time with Kyverno
- Store signatures in OCI registry
- Reject unsigned images in production namespaces

Example: Cosign Image Signing:

```
# Generate keypair (one-time)
cosign generate-key-pair

# Sign image
cosign sign --key cosign.key \
myregistry.io/myapp:v1.2.3

# Verify signature
cosign verify --key cosign.pub \
myregistry.io/myapp:v1.2.3
```

Example: Kyverno Image Verification Policy:

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-image-signatures
spec:
  validationFailureAction: enforce
  background: false
  rules:
    - name: verify-signature
      match:
        any:
          - resources:
              kinds:
                - Pod
              namespaces:
                - production
      verifyImages:
        - imageReferences:
          - "myregistry.io/*"
```

```

attestors:
- count: 1
  entries:
  - keys:
    publicKeys: |-
      -----BEGIN PUBLIC KEY-----
      ...
      -----END PUBLIC KEY-----

```

Official documentation: <https://kyverno.io/docs/writing-policies/verify-images/>

4.2.6 Service Mesh Security

Mutual TLS Between Services:

- Deploy service mesh (Istio or Linkerd)
- Enable automatic mTLS for all inter-service communication
- Enforce STRICT mTLS mode (reject plaintext)
- Monitor certificate rotation

Example: Istio Strict mTLS:

```

apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: production
spec:
  mtls:
    mode: STRICT
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: mtls-for-all
  namespace: production
spec:
  host: "*..svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL

```

Official documentation: <https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/>

Fine-Grained Authorization Policies:

- Implement least-privilege service-to-service authorization
- Deny all traffic by default, explicitly allow required paths
- Use namespaces and service accounts for authorization

Example: Istio Authorization Policy:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: frontend-to-backend
  namespace: production
spec:
  selector:
    matchLabels:
      app: backend-api
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/production/sa/frontend"]
    to:
    - operation:
        methods: ["GET", "POST"]
        paths: ["/api/v1/*"]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all-default
  namespace: production
spec:
  {} # Empty spec = deny all
```

Official documentation: <https://istio.io/latest/docs/tasks/security/authorization/>

4.3 Phase 2 Go/No-Go Checklist

Go/No-Go Checkpoint

Before removing IP whitelist and enabling public access: DDoS and Rate Limiting:

- Cloud DDoS protection enabled (Shield/Armor/DDoS Protection)
- CDN/edge caching configured
- Ingress rate limiting implemented and tested
- Application-layer rate limiting configured
- Auto-scaling policies configured

WAF and Attack Prevention:

- ModSecurity in blocking mode with tuned rules
- OWASP CRS false positives resolved
- Custom WAF rules for application-specific attacks
- WAF alerts configured and tested

Authentication and Authorization:

- OAuth2/OIDC fully implemented and tested
- API key management system deployed
- JWT validation at ingress layer
- MFA enforced for administrative access
- Service mesh mTLS enabled (STRICT mode)

Image and Container Security:

- Image scanning in CI/CD pipeline
- Image signing implemented
- Signature verification enforced
- No HIGH/CRITICAL vulnerabilities in production images

Monitoring and Response:

- Security monitoring dashboard deployed
- DDoS attack alerts configured
- WAF violation alerts configured
- Failed authentication alerts configured
- 24/7 on-call coverage established
- Incident response playbooks created

5 Phase 3: Production Internet Deployment (Weeks 9-12)

5.1 Overview

Security Posture: Complete defense-in-depth with advanced threat detection, compliance controls, and mature incident response for unrestricted public production deployment.

Acceptable Risk Profile:

- Full production deployment with unrestricted public access
- Compliance requirements (SOC 2, PCI-DSS, HIPAA, GDPR)
- High-assurance security with continuous monitoring
- Mature incident response and disaster recovery capabilities

5.2 Advanced Security Controls

5.2.1 Runtime Security and Threat Detection

Falco for Runtime Anomaly Detection:

- Deploy Falco to detect runtime anomalies
- Configure rules for:
 - Unauthorized file access
 - Unexpected network connections
 - Container escape attempts
 - Privilege escalation
 - Crypto-mining activity
- Integrate with SIEM or incident response platform

Example: Falco Custom Rules:

```
# Detect reverse shells
- rule: Reverse Shell
  desc: Detect reverse shell connection attempts
  condition: >
    spawned_process and
    (proc.name in (netcat, nc, ncat) or
     (proc.name = bash and proc.args contains "-i"))
  output: >
    Reverse shell detected (user=%user.name command=%proc.cmdline
                           container=%container.id image=%container.image.repository)
  priority: CRITICAL
  tags: [network, shell]

# Detect unauthorized file access
- rule: Read Sensitive File
  desc: Detect reads to sensitive files
  condition: >
    open_read and
```

```

fd.name in (/etc/shadow, /etc/sudoers, ~/.ssh/id_rsa)
and not proc.name in (sshd, sudo)
output: >
  Sensitive file accessed (user=%user.name file=%fd.name
  command=%proc.cmdline container=%container.id)
priority: WARNING
tags: [filesystem, security]

```

Official documentation: <https://falco.org/docs/rules/>

Tetragon for eBPF-based Security Observability:

- Deploy Tetragon for deep kernel-level observability
- Monitor system calls, network activity, file access
- Create security policies based on observed behavior
- Generate real-time security events

Official documentation: <https://tetragon.io/docs/>

5.2.2 Advanced Secrets Management

HashiCorp Vault for Dynamic Secrets:

- Deploy Vault in HA mode with auto-unsealing
- Use dynamic database credentials (short-lived)
- Implement PKI as a Service for certificate management
- Enable secret versioning and rollback
- Configure audit logging for all secret access

Example: Vault Dynamic Database Secrets:

```

# Configure database secrets engine
vault write database/config/production-db \
  plugin_name=postgresql-database-plugin \
  allowed_roles="application-role" \
  connection_url="postgresql://{{username}}:{{password}}@postgres:5432/
    app" \
  username="vault" \
  password="vault-password"

# Create role with dynamic credentials
vault write database/roles/application-role \
  db_name=production-db \
  creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expiration}}';{{GRANT SELECT , INSERT , UPDATE , DELETE ON ALL TABLES IN SCHEMA public TO \"{{name}}\";}}" \
  default_ttl="1h" \
  max_ttl="24h"

```

```
# Application retrieves dynamic credentials
vault read database/creds/application-role
# Returns:
# lease_id: database/creds/application-role/abc123
# username: v-token-applicati-abc123def456
# password: A1a-randompassword123
```

Official documentation: <https://www.vaultproject.io/docs/secrets/databases>

Vault Kubernetes Integration:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-service-account
  namespace: production
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: secure-app
  namespace: production
spec:
  template:
    metadata:
      annotations:
        vault.hashicorp.com/agent-inject: "true"
        vault.hashicorp.com/role: "application-role"
        vault.hashicorp.com/agent-inject-secret-db: "database/creds/
          application-role"
        vault.hashicorp.com/agent-inject-template-db: |
          {{- with secret "database/creds/application-role" -}}
          export DB_USER="{{ .Data.username }}"
          export DB_PASS="{{ .Data.password }}"
          {{- end -}}
    spec:
      serviceAccountName: app-service-account
      containers:
        - name: app
          image: myapp:v1.2.3
          command: ["/bin/sh", "-c"]
          args: ["source /vault/secrets/db && ./app"]
```

Official documentation: <https://www.vaultproject.io/docs/platform/k8s>

5.2.3 Comprehensive Security Monitoring

Security Information and Event Management (SIEM):

- Aggregate logs from all security controls (WAF, Falco, Kubernetes audit, application)
- Correlate events across multiple sources
- Create detection rules for attack patterns
- Implement automated response workflows

Key Security Metrics to Monitor:**1. Authentication Metrics:**

- Failed login attempts per IP/user
- Successful logins from new geographic locations
- API key usage patterns
- Token refresh rates

2. Traffic Anomalies:

- Request rate deviations from baseline
- Unusual user agent patterns
- Geographic access anomalies
- HTTP method distribution changes

3. WAF and Attack Detection:

- WAF rule violations by type
- Blocked request sources (IP, ASN)
- SQL injection/XSS attempt patterns
- File upload anomalies

4. Runtime Security:

- Falco alerts by severity
- Container escape attempts
- Privilege escalation events
- Unexpected network connections

5. Infrastructure:

- Certificate expiration status
- Security policy violations
- Image vulnerability trends
- Secret access patterns

Example: Comprehensive Security Dashboard (Grafana):

```
{  
  "dashboard": {  
    "title": "Security Monitoring Dashboard",  
    "panels": [  
      {  
        "title": "Failed Authentication Attempts",  
        "targets": [  
          {  
            "expr": "rate(authentication_failures_total[5m])"  
          }  
        ]  
      }  
    ]  
  }  
}
```

```

        ],
    },
    {
        "title": "WAF Blocks by Rule",
        "targets": [
            {
                "expr": "sum by (rule_id) (waf_blocks_total)"
            }
        ]
    },
    {
        "title": "Falco Critical Alerts",
        "targets": [
            {
                "expr": "falco_alerts{priority=\"Critical\"}"
            }
        ]
    }
]
}

```

5.2.4 Security Policy Enforcement

OPA/Gatekeeper for Admission Control:

- Enforce security policies at cluster admission
- Validate container images are signed
- Require security contexts on all pods
- Block privileged containers
- Enforce resource limits
- Validate network policy definitions

Example: Gatekeeper Constraint Templates:

```

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredsecuritycontrols
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredSecurityControls
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package k8srequiredsecuritycontrols

      violation[{"msg": msg}] {

```

```

        container := input.review.object.spec.containers[_]
        not container.securityContext.runAsNonRoot
        msg := sprintf("Container %v must run as non-root", [container.
            name])
    }

    violation[{"msg": msg}] {
        container := input.review.object.spec.containers[_]
        not container.securityContext.readOnlyRootFilesystem
        msg := sprintf("Container %v must have read-only root filesystem",
            [container.name])
    }

    violation[{"msg": msg}] {
        container := input.review.object.spec.containers[_]
        container.securityContext.allowPrivilegeEscalation
        msg := sprintf("Container %v must not allow privilege escalation",
            [container.name])
    }

---  

apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredSecurityControls
metadata:
  name: security-controls-required
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: []
        kinds: ["Pod"]
    namespaces:
      - production

```

Official documentation: <https://open-policy-agent.github.io/gatekeeper/website/docs/>

Kyverno Policy Examples:

```

apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: production-security-standards
spec:
  validationFailureAction: enforce
  background: true
  rules:
    - name: require-image-signature
      match:
        any:
          - resources:
              kinds:
                - Pod
              namespaces:
                - production
  verifyImages:

```

```

- imageReferences:
  - "*"
  attestors:
  - count: 1
    entries:
    - keys:
      publicKeys: |-
        -----BEGIN PUBLIC KEY-----
        ...
        -----END PUBLIC KEY-----

- name: require-resource-limits
  match:
  any:
  - resources:
    kinds:
    - Pod
    namespaces:
    - production
  validate:
  message: "All containers must have CPU and memory limits"
  pattern:
  spec:
    containers:
    - resources:
      limits:
        memory: "?*"
        cpu: "?*"

- name: block-latest-tag
  match:
  any:
  - resources:
    kinds:
    - Pod
    namespaces:
    - production
  validate:
  message: "Using 'latest' tag is not allowed in production"
  pattern:
  spec:
    containers:
    - image: "!*:latest"

```

Official documentation: <https://kyverno.io/policies/>

5.2.5 Compliance and Audit Controls

SOC 2 Type II Controls:

- Implement continuous audit logging (all API access, data access, admin actions)
- Enable log immutability (write-once storage)
- Maintain audit trail retention (minimum 1 year)

- Implement change management process with approval workflows
- Document and test disaster recovery procedures quarterly
- Conduct penetration testing annually

PCI-DSS Compliance (if applicable):

- Network segmentation for cardholder data environment (CDE)
- Encrypt cardholder data at rest and in transit
- Implement strong access control measures (MFA, least privilege)
- Regularly monitor and test networks
- Maintain vulnerability management program
- Implement and maintain firewall configurations

GDPR Compliance (if applicable):

- Implement data encryption at rest and in transit
- Enable data portability mechanisms
- Support right to erasure (data deletion)
- Maintain data processing records
- Implement breach notification procedures (72-hour requirement)
- Conduct Data Protection Impact Assessments (DPIAs)

Example: Audit Log Policy:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# Log all requests at metadata level
- level: Metadata
  omitStages:
    - RequestReceived

# Log request and response bodies for secret access
- level: RequestResponse
  resources:
    - group: ""
      resources: ["secrets"]

# Log request and response for authentication
- level: RequestResponse
  verbs: ["create", "update", "patch", "delete"]
  resources:
    - group: ""
      resources: ["serviceaccounts", "tokenreviews"]
```

```
# Log all API access in production namespaces
- level: Request
  namespaces: ["production", "production-data"]
  verbs: ["create", "update", "patch", "delete"]

# Don't log certain high-volume, low-value resources
- level: None
  resources:
    - group: ""
      resources: ["events"]
```

Official documentation: <https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/>

5.2.6 Incident Response Capabilities

Automated Incident Response:

- Automated blocking of malicious IPs (based on WAF violations, Falco alerts)
- Automatic scaling during DDoS attacks
- Auto-rotation of compromised secrets
- Automated pod quarantine for suspicious behavior

Example: Automated IP Blocking:

```
#!/bin/bash
# Prometheus Alertmanager webhook receiver
# Automatically blocks IPs with excessive WAF violations

while read -r alert; do
    ip=$(echo "$alert" | jq -r '.labels.client_ip')
    violation_count=$(echo "$alert" | jq -r '.annotations.violation_count')
    )

    if [ "$violation_count" -gt 100 ]; then
        # Add IP to blocklist
        kubectl annotate ingress production-ingress \
            nginx.ingress.kubernetes.io/blacklist-source-range+="ip/32"

        # Log to SIEM
        logger -t incident-response "Blocked IP $ip due to \
            violation_count WAF violations"

        # Create incident ticket
        create_incident_ticket "IP $ip auto-blocked" "$ip" \
            $violation_count"
    fi
done
```

Incident Response Playbooks:

1. DDoS Attack Response:

- Activate emergency auto-scaling

- Enable aggressive rate limiting
- Contact cloud provider for additional DDoS mitigation
- Notify stakeholders and establish incident bridge

2. Data Breach Response:

- Isolate affected systems (network policies)
- Preserve evidence (pod logs, network captures)
- Rotate all credentials
- Activate breach notification procedures
- Conduct post-incident review

3. Compromised Credentials:

- Immediately revoke compromised credentials
- Audit access logs for unauthorized activity
- Rotate all related secrets
- Force re-authentication for affected users
- Review and strengthen authentication controls

4. Zero-Day Vulnerability:

- Assess impact and exposure
- Implement temporary mitigations (WAF rules, network isolation)
- Fast-track patching process
- Monitor for exploitation attempts
- Communicate with stakeholders

Incident Documentation Requirements:

- Timeline of events
- Root cause analysis
- Impact assessment
- Remediation actions taken
- Lessons learned and improvements
- Compliance notifications (if required)

5.3 Disaster Recovery and Business Continuity

Backup Strategy:

- Automated daily backups with Velero
- Off-site backup storage (different region/provider)
- Backup encryption at rest
- Regular restore testing (quarterly)
- Document RTO (Recovery Time Objective) and RPO (Recovery Point Objective)

Example: Velero Backup Configuration:

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  name: default
  namespace: velero
spec:
  provider: aws
  objectStorage:
    bucket: production-backups-encrypted
    prefix: kubernetes
  config:
    region: us-west-2
    s3ForcePathStyle: "true"
    serverSideEncryption: AES256
---
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: production-daily-backup
  namespace: velero
spec:
  schedule: "0 2 * * *" # Daily at 2 AM
  template:
    includedNamespaces:
    - production
    - production-data
    ttl: 720h # 30 days retention
    snapshotVolumes: true
    volumeSnapshotLocations:
    - default
```

Official documentation: <https://velero.io/docs/>

Multi-Region Deployment:

- Deploy to multiple geographic regions
- Implement global load balancing with health checks
- Synchronize data across regions (with appropriate consistency model)
- Test regional failover procedures

5.4 Phase 3 Go/No-Go Checklist

Go/No-Go Checkpoint

Before full production deployment with unrestricted access:

Advanced Threat Detection:

- Falco deployed with custom rules
- Runtime security alerts integrated with incident response
- SIEM correlation rules configured
- Automated threat response workflows tested

Secrets and Credential Management:

- Vault deployed in HA mode
- Dynamic secrets implemented for databases
- Secret rotation automated (30-90 day lifecycle)
- Vault audit logging enabled

Policy and Compliance:

- OPA/Gatekeeper policies enforced
- Image signature verification required
- Security contexts enforced on all pods
- CIS Kubernetes Benchmark compliance validated
- Compliance audit logging enabled (SOC 2/PCI/HIPAA/GDPR)

Monitoring and Observability:

- Security dashboard deployed
- All critical security metrics monitored
- Alert fatigue minimized (tuned thresholds)
- Distributed tracing implemented
- Log retention meets compliance requirements (1+ years)

Incident Response:

- Incident response playbooks documented
- Automated response workflows tested
- 24/7 security operations coverage
- Incident communication plan established
- Post-incident review process defined
- Breach notification procedures documented

6 Defense-in-Depth Architecture

This section provides the conceptual framework for layered security controls across the entire application stack.

6.1 Security Layer Model

Layer	Controls	Technologies
Edge/Perimeter	DDoS protection, WAF, rate limiting, geo-blocking	CloudFront, Shield, Cloud Armor, ModSecurity
Ingress	TLS termination, authentication, path-based routing	NGINX, Traefik, Istio Ingress Gateway
Service Mesh	mTLS, service-to-service authz, observability	Istio, Linkerd
Application	Input validation, output encoding, session management	Application code, API gateway
Platform	RBAC, network policies, pod security, admission control	Kubernetes, OPA/Gatekeeper
Runtime	Anomaly detection, syscall monitoring, container isolation	Falco, Tetragon, seccomp
Data	Encryption at rest, encryption in transit, access logging	Vault, database TLS, audit logs
Infrastructure	Network segmentation, host hardening, patch management	VPC, security groups, CIS benchmarks

Table 2: Defense-in-Depth Security Layers

6.2 Zero-Trust Principles

Core Tenets:

1. **Verify Explicitly:** Always authenticate and authorize based on all available data points
2. **Least Privilege Access:** Limit user and service access with just-in-time and just-enough-access
3. **Assume Breach:** Minimize blast radius and segment access; verify end-to-end encryption

Implementation in Kubernetes:

- No implicit trust based on network location
- Every service-to-service call authenticated (mTLS)
- Every service-to-service call authorized (Istio AuthorizationPolicy)

- Minimal pod permissions (restrictive security contexts)
- Network segmentation (NetworkPolicies with default deny)
- Continuous verification (Falco runtime monitoring)

6.3 Attack Surface Reduction

Minimize Exposure:

- Expose only required ports and services
- Disable unnecessary Kubernetes APIs
- Use distroless or minimal base images
- Remove debugging tools from production images
- Implement read-only root filesystems
- Drop all unnecessary Linux capabilities

Example: Minimal Security Context:

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-app
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 10000
    fsGroup: 10000
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: app
      image: myapp:v1.2.3
      securityContext:
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        capabilities:
          drop:
            - ALL
      volumeMounts:
        - name: tmp
          mountPath: /tmp
  volumes:
    - name: tmp
      emptyDir: {}
```

7 API Security Best Practices

7.1 API Authentication Methods

Comparison of Authentication Approaches:

Method	Use Case	Pros	Cons
API Keys	Server-to-server, public APIs	Simple, fast	Not suitable for user auth, difficult rotation
OAuth 2.0	Third-party integrations	Standard, delegated access	Complex, requires token management
JWT	Microservices, SPAs	Stateless, self-contained	Token size, revocation challenges
Mutual TLS	Service mesh, high security	Strong authentication	Certificate management complexity

Table 3: API Authentication Methods

7.2 API Rate Limiting Strategies

Rate Limit Tiers:

1. **Anonymous/Unauthenticated:** 10 requests/minute (strict)
2. **Authenticated User:** 100 requests/minute
3. **Premium/Paid Tier:** 1000 requests/minute
4. **Trusted Partners:** 10000 requests/minute

Rate Limiting Algorithms:

- **Token Bucket:** Allows bursts, smooths traffic over time
- **Leaky Bucket:** Enforces constant rate, no bursts
- **Fixed Window:** Simple but allows double traffic at window boundaries
- **Sliding Window:** More accurate but more complex

Recommended: Token bucket for most use cases, sliding window for strict enforcement.

7.3 API Security Headers

Essential Security Headers:

```
# NGINX Ingress security headers
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
data:
  # Prevent clickjacking
  add-headers: "X-Frame-Options: DENY"

  # Prevent MIME sniffing
  add-headers: "X-Content-Type-Options: nosniff"
```

```
# Enable XSS protection
add-headers: "X-XSS-Protection: 1; mode=block"

# Content Security Policy
add-headers: "Content-Security-Policy: default-src 'self'"

# Referrer Policy
add-headers: "Referrer-Policy: strict-origin-when-cross-origin"

# HSTS
hsts: "true"
hsts-max-age: "31536000"
hsts-include-subdomains: "true"
```

7.4 Input Validation and Sanitization

Validation Principles:

- Validate on server side (never trust client)
- Whitelist allowed characters/patterns (reject by default)
- Enforce maximum lengths
- Validate data types and formats
- Reject unexpected input immediately

Common Injection Prevention:

- **SQL Injection:** Use parameterized queries, never string concatenation
- **XSS:** Encode output, use Content Security Policy
- **Command Injection:** Avoid shell execution, validate inputs strictly
- **Path Traversal:** Validate file paths, use chroot/jails

8 Security Testing and Validation

8.1 Automated Security Testing in CI/CD

Pipeline Security Gates:

1. **SAST (Static Application Security Testing):** Source code analysis for vulnerabilities
2. **Dependency Scanning:** Check for known vulnerabilities in dependencies
3. **Container Scanning:** Scan images for CVEs, malware, misconfigurations
4. **Infrastructure as Code Scanning:** Validate Terraform/Kubernetes manifests
5. **Secret Scanning:** Detect accidentally committed secrets

Example: GitHub Actions Security Pipeline:

```
name: Security Scanning
on: [push, pull_request]

jobs:
  security-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      # SAST with Semgrep
      - name: Semgrep Security Scan
        uses: returntocorp/semgrep-action@v1
        with:
          config: p/security-audit

      # Dependency scanning
      - name: Dependency Check
        run: |
          npm audit --audit-level=high
          pip-audit

      # Container image scanning
      - name: Build and scan image
        run: |
          docker build -t myapp:${{ github.sha }} .
          trivy image --severity HIGH,CRITICAL \
          --exit-code 1 myapp:${{ github.sha }}

      # IaC scanning
      - name: Terraform security scan
        run: |
          tfsec ./terraform/
          checkov -d ./terraform/ --framework terraform

      # Secret scanning
      - name: Secret scanning
        uses: trufflesecurity/trufflehog@main
        with:
          path: './'
          base: ${{ github.event.repository.default_branch }}
          head: HEAD
```

8.2 Penetration Testing

Annual Penetration Test Scope:

- External network penetration testing
- Web application security assessment (OWASP Top 10)
- API security testing
- Authentication and authorization bypass attempts

- Business logic testing
- Social engineering (phishing simulations)

Bug Bounty Program:

- Establish responsible disclosure policy
- Define scope (in-scope vs out-of-scope targets)
- Set bounty rewards based on severity
- Provide clear communication channels
- Respond to submissions within 48 hours

8.3 Security Chaos Engineering

Attack Simulation Scenarios:

- Simulate credential compromise and test auto-rotation
- Inject malicious traffic and validate WAF blocking
- Test DDoS response with synthetic traffic spike
- Simulate certificate expiration and validate auto-renewal
- Test incident response procedures with tabletop exercises

9 Compliance and Regulatory Requirements

9.1 Compliance Framework Mapping

Framework	Key Requirements	Implementation
SOC 2 Type II	<ul style="list-style-type: none">• Access controls• Change management• Risk assessment• Vendor management• Incident response	<ul style="list-style-type: none">• RBAC + MFA• GitOps with approval workflows• Annual risk assessments• Vendor security reviews• Documented IR playbooks

Framework	Key Requirements	Implementation
PCI-DSS	<ul style="list-style-type: none"> • Network segmentation • Encryption in transit/at rest • Access controls (MFA) • Logging and monitoring • Vulnerability management 	<ul style="list-style-type: none"> • Dedicated CDE namespace • TLS 1.2+ and AES-256 • OAuth2 + MFA enforced • 1-year audit log retention • Monthly vulnerability scans
HIPAA	<ul style="list-style-type: none"> • PHI encryption • Access controls • Audit controls • Transmission security • Integrity controls 	<ul style="list-style-type: none"> • Encryption at rest/transit • Least-privilege RBAC • Comprehensive audit logging • mTLS for all services • Digital signatures (Cosign)
GDPR	<ul style="list-style-type: none"> • Data protection by design • Right to erasure • Data portability • Breach notification (72hr) • DPIAs for high-risk processing 	<ul style="list-style-type: none"> • Encryption, pseudonymization • Data deletion APIs • Export functionality • Automated alerting system • DPIA templates and process

Framework	Key Requirements	Implementation
ISO 27001	<ul style="list-style-type: none"> • ISMS framework • Risk assessment • Asset management • Incident management • Business continuity 	<ul style="list-style-type: none"> • Documented ISMS policies • Annual risk assessments • Asset inventory (CMDB) • IR procedures and drills • DR testing (quarterly)

Table 4: Compliance Framework Implementation

9.2 Audit Trail Requirements

Minimum Audit Logging:

- User authentication events (success and failure)
- Authorization decisions (access granted/denied)
- Resource creation, modification, deletion
- Administrative actions
- Security policy changes
- Secret access
- Data access (especially PII/PHI)

Log Retention:

- SOC 2: Minimum 1 year
- PCI-DSS: Minimum 1 year, online for 3 months
- HIPAA: Minimum 6 years
- GDPR: Varies by data type and legal basis

10 Continuous Security Improvement

10.1 Security Metrics and KPIs

Track and Report Monthly:

1. Vulnerability Metrics:

- Mean Time to Detect (MTTD) vulnerabilities
- Mean Time to Remediate (MTTR) vulnerabilities
- Number of HIGH/CRITICAL vulnerabilities in production
- Percentage of images with known vulnerabilities

2. Attack Metrics:

- WAF block rate (blocked requests / total requests)
- DDoS attack frequency and duration
- Failed authentication attempts
- API rate limit violations

3. Compliance Metrics:

- Audit findings (open vs closed)
- Policy violations detected
- Time to compliance for new services
- Percentage of resources meeting security standards

4. Incident Response Metrics:

- Mean Time to Detect (MTTD) incidents
- Mean Time to Respond (MTTR) incidents
- Number of security incidents by severity
- Percentage of incidents with complete post-mortems

10.2 Security Review Cadence

Quarterly Activities:

- Review and update security policies
- Conduct tabletop incident response exercises
- Review access control lists (user and service accounts)
- Test disaster recovery procedures
- Update threat model

Annual Activities:

- Third-party penetration testing
- Security awareness training for all staff
- Comprehensive security architecture review
- Compliance audit preparation and execution
- Risk assessment update

10.3 Threat Intelligence Integration

Integrate Threat Feeds:

- Subscribe to CVE databases (NVD, vendor advisories)
- Monitor CISA Known Exploited Vulnerabilities catalog
- Track Kubernetes security advisories
- Join cloud provider security bulletins
- Participate in industry-specific ISACs (Information Sharing and Analysis Centers)

Actionable Threat Intelligence:

- Automatically update WAF rules based on threat intelligence
- Prioritize patching based on active exploitation
- Update IP blocklists with known malicious sources
- Adjust monitoring rules for emerging attack patterns

11 Conclusion

11.1 Critical Success Factors

Successful internet-facing application security requires:

1. **Defense-in-Depth:** Multiple layers of security controls, no single point of failure
2. **Zero-Trust Architecture:** Continuous verification at every layer
3. **Automation:** Security controls integrated into CI/CD, not bolt-on
4. **Continuous Monitoring:** Real-time visibility and alerting
5. **Incident Response Readiness:** Documented procedures, tested regularly
6. **Security Culture:** Security is everyone's responsibility, not just security team

11.2 Common Pitfalls to Avoid

Security Warning

Avoid These Common Mistakes:

- Skipping security phases to accelerate deployment
- Deploying internet-facing before Phase 2 controls complete
- Relying solely on perimeter security (no defense-in-depth)
- Neglecting log monitoring and alerting
- Using default credentials or weak passwords
- Failing to patch vulnerabilities promptly
- Ignoring security alerts due to alert fatigue
- Not testing incident response procedures

11.3 Roadmap Beyond Phase 3

Continuous Improvement:

- Implement machine learning for anomaly detection
- Adopt eBPF-based security tools (Tetragon, Cilium)
- Expand to multi-region active-active deployment
- Implement automated threat hunting
- Achieve additional compliance certifications
- Establish bug bounty program
- Develop security chaos engineering practice

11.4 Final Checklist

Go/No-Go Checkpoint

Production Readiness Final Verification: Security Controls (All Phases):

- All Phase 1, 2, and 3 controls implemented
- Defense-in-depth at all layers verified
- Zero-trust principles enforced

Testing and Validation:

- Penetration testing completed with findings remediated
- Load testing at 3x expected capacity
- Disaster recovery procedures tested successfully
- Incident response drills conducted

Operational Readiness:

- 24/7 security monitoring operational
- On-call rotation established
- Runbooks documented and accessible
- Escalation procedures defined

Compliance:

- All required compliance controls implemented
- Audit logging meets retention requirements
- Compliance documentation complete
- Privacy policies updated

Documentation:

- Architecture diagrams current
- Security policies documented
- Incident response playbooks complete
- Disaster recovery procedures documented

11.5 Resources and Community

Kubernetes Security Resources:

- Kubernetes Security SIG: <https://github.com/kubernetes/community/tree/master/sig-security>
- CNCF Security Technical Advisory Group: <https://github.com/cncf/tag-security>
- Kubernetes Security Checklist: <https://kubernetes.io/docs/concepts/security/security-checklist/>

Security Training and Certifications:

- Certified Kubernetes Security Specialist (CKS)
- Certified Information Systems Security Professional (CISSP)
- GIAC Cloud Security Automation (GCSA)
- Cloud Security Alliance (CSA) Certifications

Security Communities:

- OWASP Foundation: <https://owasp.org/>
- Cloud Native Security Con: <https://events.linuxfoundation.org/cloudnativesecuritycon-north-america/>
- r/kubernetes (Reddit)
- Kubernetes Slack #sig-security channel