Production Security Hardening Configuration Guide

This guide provides detailed configurations for securing your ThreatCompass deployment in production.

1. RDS Secret Rotation Setup

Enable Automatic Secret Rotation

The Terraform templates already create the necessary secrets. To enable automatic rotation:

```bash
# Create rotation Lambda function (AWS provides this)
aws secretsmanager update-secret \
  --secret-id threatcompass-production/database/credentials \
  --secret-string '{"username":"threatcompass_admin","password":"NEW_PASSW
  --region us-east-1

# Enable rotation (30-day cycle)
aws secretsmanager rotate-secret \
  --secret-id threatcompass-production/database/credentials \
  --rotation-rules AutomaticallyAfterDays=30 \
  --region us-east-1
```

Application Configuration for Secret Rotation

Update your Flask application configuration:

python

```python
# config.py - Add to your existing config
import boto3
import json
from botocore.exceptions import ClientError

class ProductionConfig(Config):
    @staticmethod
    def get_secret(secret_name, region_name="us-east-1"):
        """Retrieve secrets from AWS Secrets Manager"""
        session = boto3.session.Session()
        client = session.client(
            service_name='secretsmanager',
            region_name=region_name
        )

        try:
            get_secret_value_response = client.get_secret_value(
                SecretId=secret_name
            )
            secret = json.loads(get_secret_value_response['SecretString'])
            return secret
        except ClientError as e:
            raise e

    def __init__(self):
        # Get database credentials from Secrets Manager
        db_secret = self.get_secret("threatcompass-production/database/credentials"
        redis_secret = self.get_secret("threatcompass-production/redis/credentials")
```

```
        app_secret = self.get_secret("threatcompass-production/app/secrets")

        # Set configuration from secrets
        self.SQLALCHEMY_DATABASE_URI = db_secret['url']
        self.CELERY_BROKER_URL = redis_secret['url']
        self.CELERY_RESULT_BACKEND = redis_secret['url']
        self.SECRET_KEY = app_secret['flask_secret_key']
        self.VT_API_KEY = app_secret.get('virustotal_api_key', '')
        self.ABUSEIPDB_API_KEY = app_secret.get('abuseipdb_api_key', '')
```

2. SSL Certificate Management with AWS ACM

Option A: Using AWS Certificate Manager (Recommended)

If you have a domain, AWS ACM provides free SSL certificates with auto-renewal:

```
bash

# Request certificate via CLI (or use Terraform as shown above)
aws acm request-certificate \
    --domain-name threatcompass.yourdomain.com \
    --subject-alternative-names "*.threatcompass.yourdomain.com" \
    --validation-method DNS \
    --region us-east-1
```

Option B: Import Existing Certificate

```bash
# Import your own certificate
aws acm import-certificate \
    --certificate fileb://Certificate.pem \
    --certificate-chain fileb://CertificateChain.pem \
    --private-key fileb://PrivateKey.pem \
    --region us-east-1
```

Domain Validation

For DNS validation, add the CNAME records provided by ACM to your DNS provider:

```bash
# Get validation records
aws acm describe-certificate \
    --certificate-arn arn:aws:acm:us-east-1:123456789012:certificate/12345678-12
    --region us-east-1
```

3. Security Headers and CORS Configuration

Option A: Application-Level Security Headers (Flask)

Add to your Flask application:

python

```python
# security_headers.py
from flask import Flask
from functools import wraps

def add_security_headers(app: Flask):
    """Add security headers to all responses"""

    @app.after_request
    def set_security_headers(response):
        # Prevent clickjacking
        response.headers['X-Frame-Options'] = 'DENY'

        # Prevent MIME type sniffing
        response.headers['X-Content-Type-Options'] = 'nosniff'

        # Enable XSS protection
        response.headers['X-XSS-Protection'] = '1; mode=block'

        # Enforce HTTPS (only in production)
        if app.config.get('ENV') == 'production':
            response.headers['Strict-Transport-Security'] = 'max-age=31536000; inclu

        # Content Security Policy
        csp = (
            "default-src 'self'; "
            "script-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://cdnjs.cloudflare
            "style-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://cdnjs.cloudflare.
            "img-src 'self' data: https:; "
```

```
        "font-src 'self' https://cdn.jsdelivr.net https://cdnjs.cloudflare.com; "
        "connect-src 'self'; "
        "frame-ancestors 'none';"
    )
    response.headers['Content-Security-Policy'] = csp

    # Referrer Policy
    response.headers['Referrer-Policy'] = 'strict-origin-when-cross-origin'

    # Permissions Policy (formerly Feature Policy)
    response.headers['Permissions-Policy'] = (
        "accelerometer=(), "
        "camera=(), "
        "geolocation=(), "
        "gyroscope=(), "
        "magnetometer=(), "
        "microphone=(), "
        "payment=(), "
        "usb=()"
    )

    return response

# In your app.py
from security_headers import add_security_headers

app = create_app()
add_security_headers(app)
```

Option B: ALB-Level Security Headers

Add custom response headers via ALB listener rules:

bash

```
# Example AWS CLI command to add security headers at ALB level
aws elbv2 modify-listener \
  --listener-arn arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/ap
  --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:u
```

CORS Configuration

```python
# cors_config.py
from flask_cors import CORS

def configure_cors(app):
    """Configure CORS for production"""

    if app.config.get('ENV') == 'production':
        # Strict CORS for production
        CORS(app,
             origins=['https://threatcompass.yourdomain.com'],
             methods=['GET', 'POST', 'PUT', 'DELETE', 'PATCH'],
             allow_headers=['Content-Type', 'Authorization', 'X-API-Key'],
             expose_headers=['X-Total-Count'],
             supports_credentials=True,
             max_age=3600)
    else:
        # More permissive for development
        CORS(app, origins=['http://localhost:3000', 'http://localhost:5000'])

# In your app.py
from cors_config import configure_cors
configure_cors(app)
```

4. WAF Rate Limiting Configuration

The Terraform templates include basic WAF rules. Here's how to customize them:

Custom WAF Rules

hcl

```
# Add to your Terraform WAF configuration
resource "aws_wafv2_rule_group" "threatcompass_custom" {
  name     = "${local.name_prefix}-custom-rules"
  scope    = "REGIONAL"
  capacity = 100

  rule {
    name     = "BlockSuspiciousUserAgents"
    priority = 1

    action {
      block {}
    }

    statement {
      byte_match_statement {
        search_string = "sqlmap"
        field_to_match {
          single_header {
            name = "user-agent"
          }
        }
        text_transformation {
          priority = 0
          type     = "LOWERCASE"
        }
        positional_constraint = "CONTAINS"
      }
```

```
  }

  visibility_config {
    cloudwatch_metrics_enabled = true
    metric_name                = "BlockSuspiciousUserAgents"
    sampled_requests_enabled   = true
  }
}

rule {
  name     = "RateLimitAPI"
  priority = 2

  action {
    block {}
  }

  statement {
    rate_based_statement {
      limit              = 100  # requests per 5-minute window
      aggregate_key_type = "IP"

      scope_down_statement {
        byte_match_statement {
          search_string = "/api/"
          field_to_match {
            uri_path {}
          }
          text_transformation {
```

```
            priority = 0
            type     = "LOWERCASE"
          }
          positional_constraint = "STARTS_WITH"
        }
      }
    }
  }

  visibility_config {
    cloudwatch_metrics_enabled = true
    metric_name                = "RateLimitAPI"
    sampled_requests_enabled   = true
  }
 }
}
```

WAF Logging Configuration

```hcl
# WAF Logging
resource "aws_wafv2_web_acl_logging_configuration" "main" {
  resource_arn            = aws_wafv2_web_acl.main.arn
  log_destination_configs = [aws_cloudwatch_log_group.waf.arn]

  redacted_fields {
    single_header {
      name = "authorization"
    }
  }

  redacted_fields {
    single_header {
      name = "x-api-key"
    }
  }
}

resource "aws_cloudwatch_log_group" "waf" {
  name              = "/aws/wafv2/${local.name_prefix}"
  retention_in_days = 30
}
```

5. Backup and Disaster Recovery

RDS Backup Configuration

hcl

```hcl
# Enhanced RDS backup configuration (add to existing RDS resource)
resource "aws_db_instance" "main" {
  # ... existing configuration ...

  # Extended backup settings
  backup_retention_period   = 30  # Keep backups for 30 days
  backup_window             = "03:00-04:00"  # UTC
  maintenance_window        = "Sun:04:00-Sun:05:00"  # UTC

  # Enable automated backups to S3
  copy_tags_to_snapshot = true

  # Enable point-in-time recovery
  delete_automated_backups = false

  # Performance Insights for monitoring
  performance_insights_enabled          = true
  performance_insights_retention_period = 31  # days
}
```

Cross-Region Backup Strategy

hcl

```
# Cross-region RDS snapshot copying
resource "aws_db_snapshot_copy" "main" {
  count = var.enable_cross_region_backup ? 1 : 0

  source_db_snapshot_identifier = aws_db_instance.main.final_snapshot_identifier
  target_custom_availability_zone = "us-west-2a"
  target_db_snapshot_identifier = "${local.name_prefix}-snapshot-replica"

  tags = local.common_tags
}

# S3 Cross-Region Replication
resource "aws_s3_bucket_replication_configuration" "logs" {
  role   = aws_iam_role.s3_replication.arn
  bucket = aws_s3_bucket.logs.id

  rule {
    id     = "replicate-logs"
    status = "Enabled"

    destination {
      bucket        = aws_s3_bucket.logs_replica.arn
      storage_class = "STANDARD_IA"

      encryption_configuration {
        replica_kms_key_id = aws_kms_key.replica.arn
      }
    }
```

```
  }

  depends_on = [aws_s3_bucket_versioning.logs]
}

# Backup S3 bucket in different region
resource "aws_s3_bucket" "logs_replica" {
  provider = aws.replica
  bucket   = "${local.name_prefix}-logs-replica-${random_string.bucket_suffix.resu
}
```

ECS Service Auto-Recovery

hcl

```
# CloudWatch Alarms for ECS Service Health
resource "aws_cloudwatch_metric_alarm" "ecs_service_unhealthy" {
  alarm_name          = "${local.name_prefix}-ecs-unhealthy-tasks"
  comparison_operator = "LessThanThreshold"
  evaluation_periods  = "2"
  metric_name         = "HealthyHostCount"
  namespace           = "AWS/ApplicationELB"
  period              = "300"
  statistic           = "Average"
  threshold           = "1"
  alarm_description   = "This metric monitors healthy ECS tasks"
  alarm_actions       = [aws_sns_topic.alerts.arn]

  dimensions = {
    TargetGroup  = aws_lb_target_group.flask_app.arn_suffix
    LoadBalancer = aws_lb.main.arn_suffix
  }
}

# SNS Topic for Alerts
resource "aws_sns_topic" "alerts" {
  name = "${local.name_prefix}-alerts"
}

# Auto-restart unhealthy ECS services
resource "aws_cloudwatch_event_rule" "ecs_task_stopped" {
  name        = "${local.name_prefix}-ecs-task-stopped"
  description = "Capture ECS task stopped events"
```

```
  event_pattern = jsonencode({
    source      = ["aws.ecs"]
    detail-type = ["ECS Task State Change"]
    detail = {
      lastStatus = ["STOPPED"]
      clusterArn = [aws_ecs_cluster.main.arn]
    }
  })
}

resource "aws_cloudwatch_event_target" "ecs_restart" {
  rule      = aws_cloudwatch_event_rule.ecs_task_stopped.name
  target_id = "RestartECSTask"
  arn       = aws_lambda_function.ecs_restart.arn
}
```

Disaster Recovery Runbook

bash

```bash
#!/bin/bash
# disaster_recovery.sh - Automated disaster recovery script

set -e

ENVIRONMENT="production"
REGION="us-east-1"
BACKUP_REGION="us-west-2"

echo "=== ThreatCompass Disaster Recovery ==="
echo "Environment: $ENVIRONMENT"
echo "Primary Region: $REGION"
echo "Backup Region: $BACKUP_REGION"

# Step 1: Assess damage
echo "1. Assessing current infrastructure status..."
aws ecs describe-clusters --cluster threatcompass-$ENVIRONMENT-cluster --reg

# Step 2: Restore database
echo "2. Restoring database from latest snapshot..."
LATEST_SNAPSHOT=$(aws rds describe-db-snapshots \
  --db-instance-identifier threatcompass-$ENVIRONMENT-postgresql \
  --snapshot-type automated \
  --query 'DBSnapshots | sort_by(@, &SnapshotCreateTime) | [-1].DBSnapshotIder
  --output text \
  --region $REGION)

echo "Latest snapshot: $LATEST_SNAPSHOT"
```

```
# Restore database
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier threatcompass-$ENVIRONMENT-postgresql-restored \
  --db-snapshot-identifier $LATEST_SNAPSHOT \
  --region $REGION

# Step 3: Update ECS services
echo "3. Updating ECS service configurations..."
aws ecs update-service \
  --cluster threatcompass-$ENVIRONMENT-cluster \
  --service threatcompass-$ENVIRONMENT-flask-app \
  --force-new-deployment \
  --region $REGION

# Step 4: Verify health
echo "4. Verifying application health..."
sleep 300  # Wait for services to start

HEALTH_URL="https://$(aws elbv2 describe-load-balancers \
  --names threatcompass-$ENVIRONMENT-alb \
  --query 'LoadBalancers[0].DNSName' \
  --output text \
  --region $REGION)/health"

curl -f $HEALTH_URL || {
  echo "Health check failed!"
  exit 1
}
```

```
echo "=== Disaster Recovery Complete ==="
```

6. S3 Lifecycle Policies and Data Retention

Enhanced S3 Lifecycle Rules

hcl

```
resource "aws_s3_bucket_lifecycle_configuration" "enhanced_logs" {
  bucket = aws_s3_bucket.logs.id

  rule {
    id     = "application_logs"
    status = "Enabled"

    filter {
      prefix = "application-logs/"
    }

    transition {
      days          = 30
      storage_class = "STANDARD_IA"
    }

    transition {
      days          = 90
      storage_class = "GLACIER"
    }

    transition {
      days          = 365
      storage_class = "DEEP_ARCHIVE"
    }

    expiration {
      days = 2555  # 7 years for compliance
```

```
  }
}

rule {
  id     = "access_logs"
  status = "Enabled"

  filter {
    prefix = "access-logs/"
  }

  transition {
    days          = 7
    storage_class = "STANDARD_IA"
  }

  transition {
    days          = 30
    storage_class = "GLACIER"
  }

  expiration {
    days = 90  # Shorter retention for access logs
  }
}

rule {
  id     = "audit_logs"
  status = "Enabled"
```

```
    filter {
      prefix = "audit-logs/"
    }

    transition {
      days          = 90
      storage_class = "GLACIER"
    }

    transition {
      days          = 365
      storage_class = "DEEP_ARCHIVE"
    }

    expiration {
      days = 3653  # 10 years for audit logs
    }
  }
}
```

7. Monitoring and Alerting Configuration

CloudWatch Custom Metrics

python

```python
# monitoring.py - Add to your Flask application
import boto3
import time
from datetime import datetime

class CloudWatchMetrics:
    def __init__(self, namespace="ThreatCompass"):
        self.cloudwatch = boto3.client('cloudwatch')
        self.namespace = namespace

    def put_metric(self, metric_name, value, unit='Count', dimensions=None):
        """Send custom metric to CloudWatch"""
        try:
            self.cloudwatch.put_metric_data(
                Namespace=self.namespace,
                MetricData=[
                    {
                        'MetricName': metric_name,
                        'Value': value,
                        'Unit': unit,
                        'Timestamp': datetime.utcnow(),
                        'Dimensions': dimensions or []
                    }
                ]
            )
        except Exception as e:
            print(f"Failed to send metric {metric_name}: {e}")
```

```
    def record_ioc_processed(self, ioc_type):
        """Record IOC processing metrics"""
        self.put_metric(
            'IOCsProcessed',
            1,
            dimensions=[{'Name': 'IOCType', 'Value': ioc_type}]
        )

    def record_playbook_generated(self, tenant_id):
        """Record playbook generation metrics"""
        self.put_metric(
            'PlaybooksGenerated',
            1,
            dimensions=[{'Name': 'TenantId', 'Value': str(tenant_id)}]
        )

# Usage in your application
metrics = CloudWatchMetrics()

@app.route('/api/v1/iocs', methods=['POST'])
def create_ioc():
    # ... existing IOC creation logic ...
    metrics.record_ioc_processed(ioc.type)
    return jsonify({"status": "success"})
```

Comprehensive Monitoring Stack

hcl

```
# CloudWatch Dashboard
resource "aws_cloudwatch_dashboard" "main" {
  dashboard_name = "${local.name_prefix}-dashboard"

  dashboard_body = jsonencode({
    widgets = [
      {
        type   = "metric"
        x      = 0
        y      = 0
        width  = 12
        height = 6

        properties = {
          metrics = [
            ["AWS/ApplicationELB", "RequestCount", "LoadBalancer", aws_lb.main.arn_s
            [".", "TargetResponseTime", ".", "."],
            [".", "HTTPCode_Target_2XX_Count", ".", "."],
            [".", "HTTPCode_Target_4XX_Count", ".", "."],
            [".", "HTTPCode_Target_5XX_Count", ".", "."]
          ]
          view    = "timeSeries"
          stacked = false
          region  = var.aws_region
          title   = "Application Load Balancer Metrics"
          period  = 300
        }
      },
```

```
      {
        type   = "metric"
        x      = 0
        y      = 6
        width  = 12
        height = 6

        properties = {
          metrics = [
            ["AWS/ECS", "CPUUtilization", "ServiceName", aws_ecs_service.flask_app.n
            [".", "MemoryUtilization", ".", ".", ".", "."]
          ]
          view   = "timeSeries"
          region = var.aws_region
          title  = "ECS Service Metrics"
          period = 300
        }
      }
    ]
  })
}
```

This comprehensive hardening guide provides production-ready security configurations for your ThreatCompass deployment. Each section can be implemented incrementally, and the monitoring components will help you maintain visibility into your application's security posture.