ADMIN:DOCS

Wazuh: Open Source SIEM und XDR für DevOps und Security-Teams

Umfassende technische Dokumentation zur Integration der Open-Source Security-Plattform Wazuh in DevOps-Workflows. Mit praktischen Beispielen für Automation, Monitoring und Security-Integration.

Kategorie: DevOps **Veröffentlicht:** 13.11.2024 **Zuletzt geprüft:** 09.05.2025

Wichtige Information: Die Implementation einer Enterprise-Security-Plattform wie Wazuh erfordert fundierte Kenntnisse in Systemadministration, Netzwerktechnik und Security-Konzepten. Dieser technische Leitfaden richtet sich an erfahrene Administratoren und Security-Engineers.

Voraussetzungen:

- Umfangreiche Linux-Systemadministration-Erfahrung
- Fundierte Kenntnisse in Netzwerktechnik und Security
- Verständnis von Container-Technologien und Virtualisierung
- Erfahrung mit Log-Management und Monitoring-Systemen

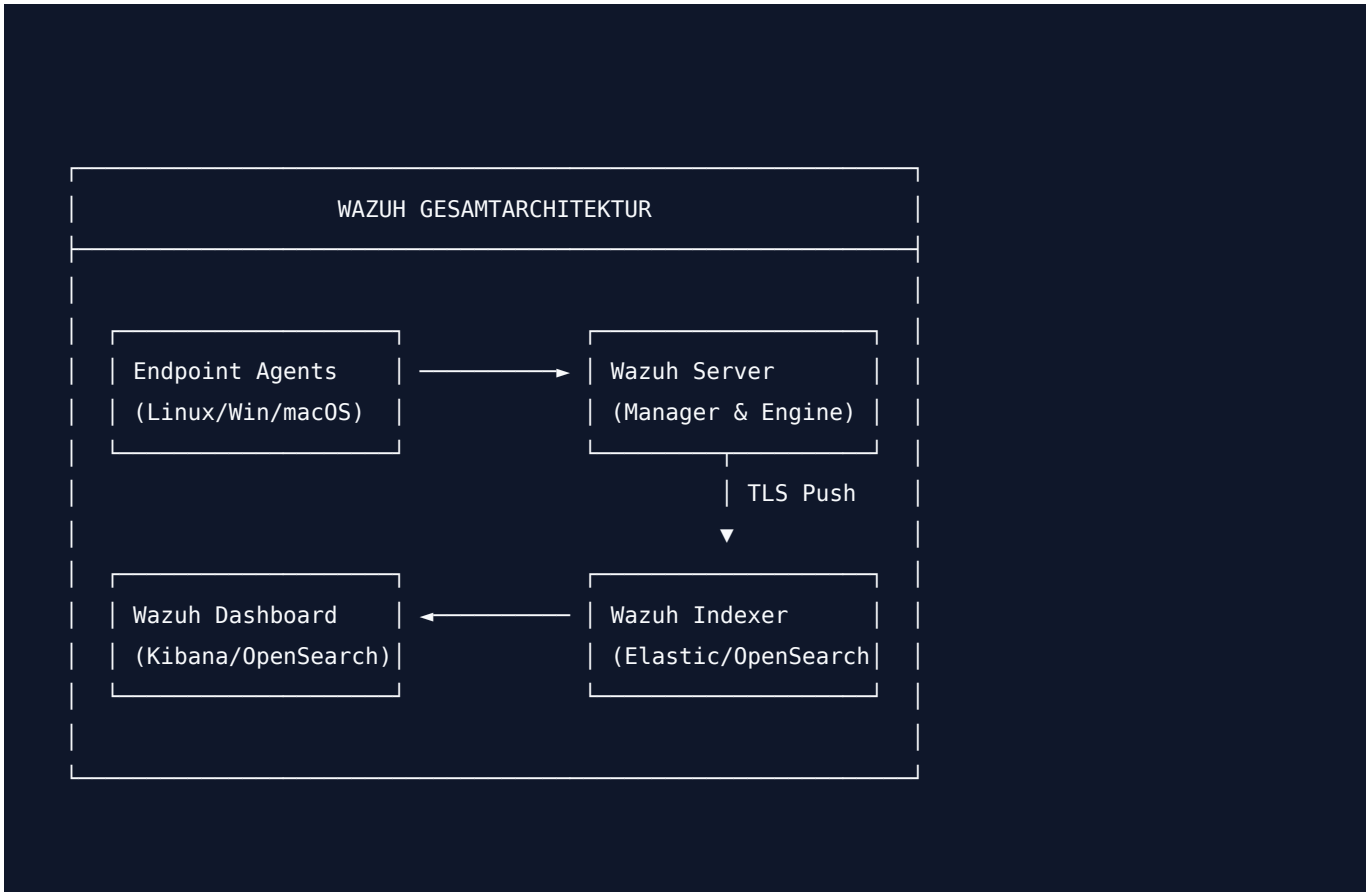
Was ist Wazuh?

In einer Zeit, in der Cybersicherheit nicht mehr optional ist, sondern überlebenswichtig für jede IT-Infrastruktur, brauchen wir mehr als nur einzelne Sicherheitswerkzeuge. Wazuh vereint als Open-Source-Plattform SIEM (Security Information and Event Management) mit XDR (Extended Detection and Response) Funktionen und bietet damit eine umfassende Sicherheitslösung.

Quelle: https://admindocs.de/de/devops/wazuh-open-source-siem-und-xdr-fuer-devops-und-security-teams

Stell dir Wazuh wie ein hochmodernes Sicherheitssystem vor: Es überwacht nicht nur alle Eingänge (Logs, Netzwerkverkehr), sondern analysiert auch das Verhalten im Inneren (Systemaktivitäten, Dateiveränderungen) und kann automatisch auf Bedrohungen reagieren. Anders als traditionelle Sicherheitslösungen arbeitet Wazuh dabei mit einem verteilten Ansatz – ein zentraler Server koordiniert zahlreiche Agenten, die auf den zu überwachenden Systemen laufen.

Die Plattform integriert sich nahtlos in moderne DevOps-Umgebungen und unterstützt Container-Überwachung, Cloud-Workloads und traditionelle On-Premise-Systeme gleichermaßen.



Komponenten und Architektur

Wazuh's Architektur ist modular aufgebaut, wobei jede Komponente eine spezifische Rolle in der Sicherheitsüberwachung übernimmt.

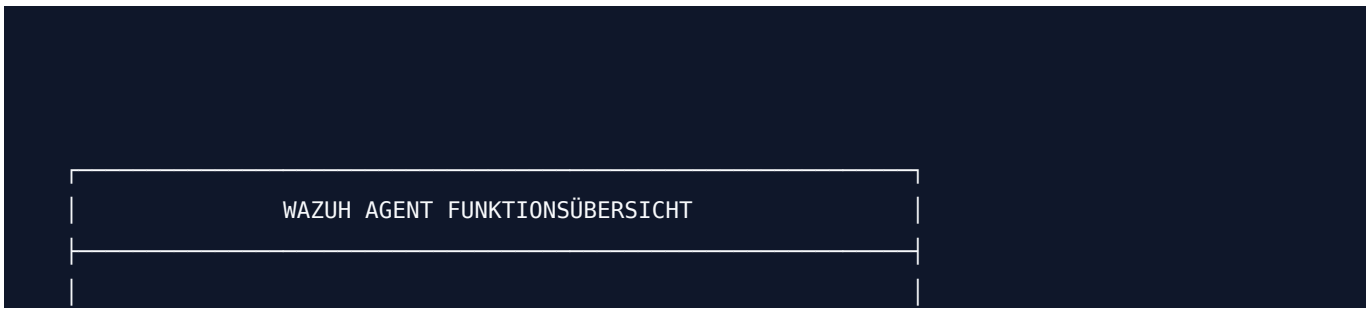
Kernkomponenten

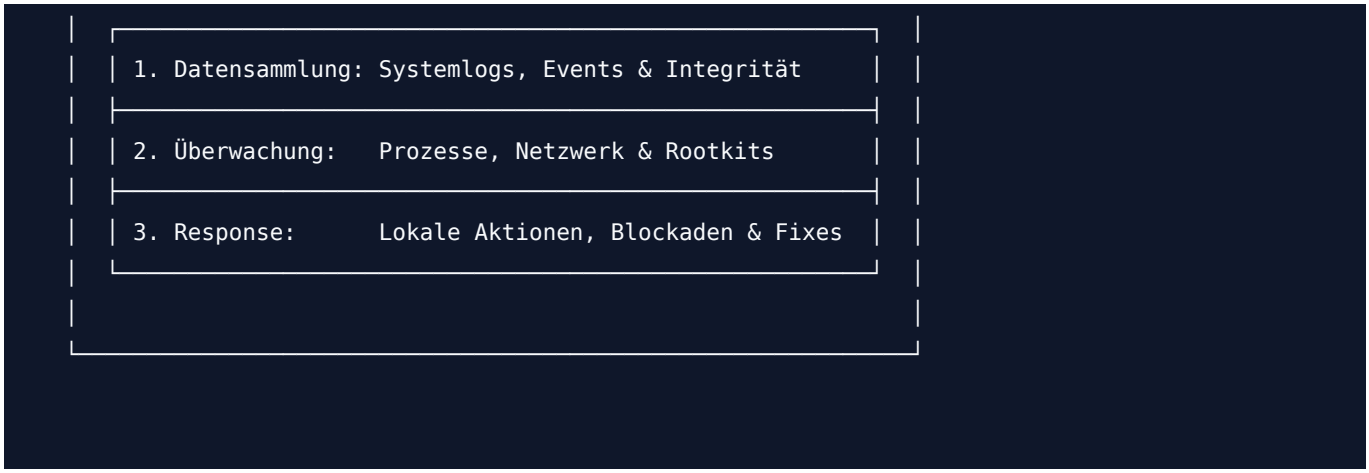


1. Wazuh Manager:

- Zentrales Nervensystem der Plattform
- Verarbeitet und analysiert eingehende Daten
- Führt Regelauswertungen durch
- Generiert Warnmeldungen
- Steuert Active Response

2. Wazuh Agents:



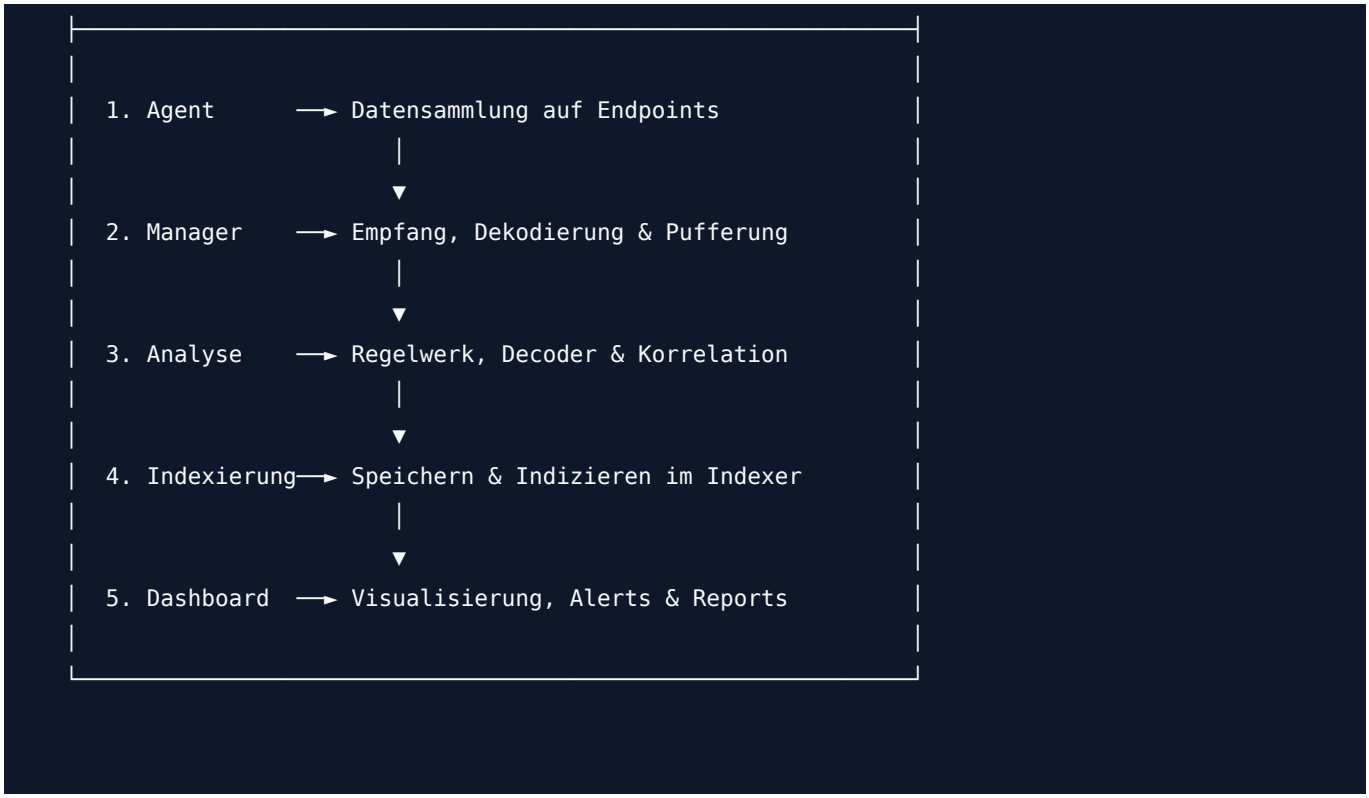


3. Elastic Stack Integration:



Funktionsweise und Datenfluss





Vergleich mit anderen SIEM-Lösungen

Feature	Wazuh	Splunk	ELK Stack	QRadar
Open Source	✅	❌	✅	❌
Active Response	✅	⚠️	✅	⚠️
FIM	✅	✅	⚠️	✅
Container Security	✅	⚠️	⚠️	⚠️
Cloud Integration	✅	✅	✅	✅
Skalierbarkeit	✅	✅	✅	✅

> **🔍 Wichtige Aspekte:**
>
> - **Architekturvorteile:** Verteilte Architektur ermöglicht Skalierung, modularer Aufbau für flexible Erweiterung, integrierte Analyse-Engine für Echtzeitverarbeitung.

- **Unterscheidungsmerkmale:** Native Integration von SIEM und XDR, vollständige Active Response Fähigkeiten, kostenlose Enterprise-Features.
- **Besondere Stärken:** Container und Cloud-native Überwachung, Regelbasierte und ML-gestützte Analyse, Compliance-Monitoring (PCI DSS, GDPR, HIPAA).

Installation und Grundkonfiguration

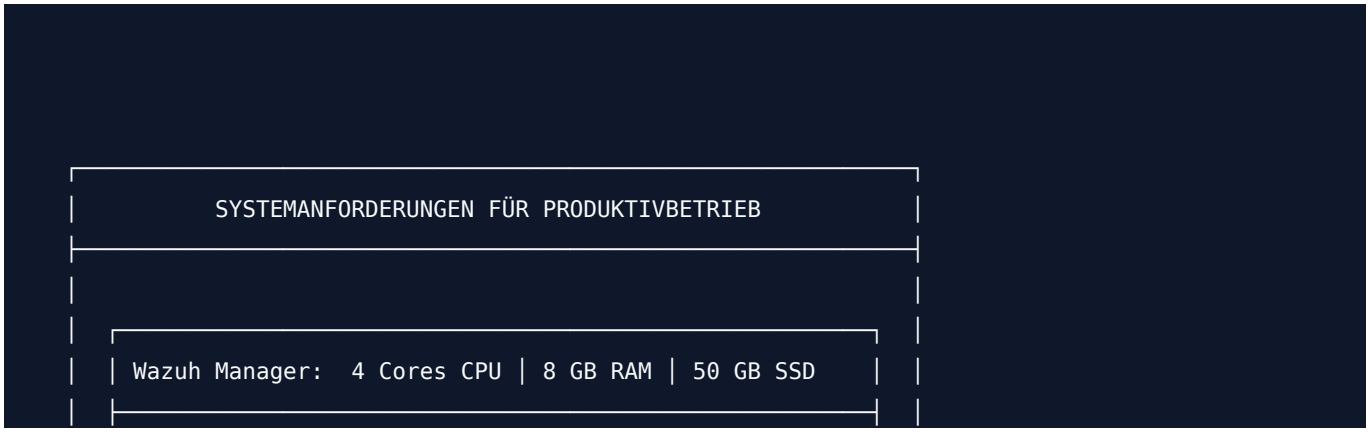
Systemanforderungen und Vorbereitung

Bevor wir mit der Installation beginnen, müssen wir sicherstellen, dass unser System die notwendigen Anforderungen erfüllt.

Information: In diesem Artikel verwenden wir Ubuntu 22.04 als Beispiel-Distribution. Die grundlegenden Konzepte sind auf allen Linux-Systemen gleich, aber die Installation von Paketen und einige Konfigurationspfade können sich je nach Distribution unterscheiden.

Systemanforderungen:

Die richtige Dimensionierung deines Wazuh-Systems ist entscheidend für dessen Leistungsfähigkeit.



```
│  │ Elastic Stack:  4 Cores CPU │ 8 GB RAM │ 100 GB SSD   │  │
│  ├──────────────────────────────────────────────────────┤  │
│  │ Kapazität:      Bis 100 Agents │ 5000 EPS │ 30d Ret.  │  │
│  └──────────────────────────────────────────────────────┘  │
│                                                            │
└────────────────────────────────────────────────────────────┘
```

> ⚠ **Wichtig:** Diese Anforderungen gelten für Produktivumgebungen. Für Tests kann man mit weniger Ressourcen arbeiten, sollte aber mindestens 4 GB RAM zur Verfügung stellen.

Systemvorbereitung:

```
# System aktualisieren
sudo apt update && sudo apt upgrade -y

# Notwendige Pakete installieren
sudo apt install curl apt-transport-https unzip wget libcap2-bin software-properties-
common lsb-release gnupg2 -y

# Systembeschränkungen anpassen
cat > /etc/sysctl.d/99-wazuh.conf << EOF
vm.max_map_count=262144
fs.file-max=65535
EOF

# Änderungen aktivieren
sudo sysctl -p /etc/sysctl.d/99-wazuh.conf
```

Wazuh-Server Installation

Die Installation des Wazuh-Servers ist der Grundstein deiner Security-Monitoring-Infrastruktur. Wir nutzen die offiziellen Paketquellen, um eine saubere und updatefähige Installation zu gewährleisten.

```
# GPG-Key hinzufügen
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | gpg --no-default-keyring --keyring
gnupg-ring:/usr/share/keyrings/wazuh.gpg --import && chmod 644
/usr/share/keyrings/wazuh.gpg

# Repository hinzufügen
echo "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.wazuh.com/4.x/apt/
stable main" | tee -a /etc/apt/sources.list.d/wazuh.list

# Wazuh-Manager installieren
sudo apt update
sudo apt install wazuh-manager -y
```

Elastic Stack Integration

Der Elastic Stack bildet das Rückgrat für Datenspeicherung und -analyse in Wazuh.

```
# Elasticsearch Installation
curl -s https://artifacts.elastic.co/GPG-KEY-elasticsearch | gpg --no-default-keyring --
keyring gnupg-ring:/usr/share/keyrings/elasticsearch.gpg --import
echo "deb [signed-by=/usr/share/keyrings/elasticsearch.gpg]
https://artifacts.elastic.co/packages/7.x/apt stable main" | tee
/etc/apt/sources.list.d/elastic-7.x.list
sudo apt update && sudo apt install elasticsearch=7.17.9

# Elasticsearch Konfiguration
cat > /etc/elasticsearch/elasticsearch.yml << EOF
node.name: node-1
network.host: localhost
http.port: 9200
cluster.initial_master_nodes: ["node-1"]
EOF
```

Dashboard-Einrichtung

Das Wazuh-Dashboard, basierend auf Kibana, ist deine zentrale Anlaufstelle für Analyse und Monitoring.

```
# Kibana Installation
sudo apt install kibana=7.17.9

# Wazuh Plugin Installation
cd /usr/share/kibana
sudo -u kibana bin/kibana-plugin install
https://packages.wazuh.com/4.x/ui/kibana/wazuh_kibana-4.4.0_7.17.9-1.zip
```

Dienste aktivieren und starten

```
# Dienste aktivieren
sudo systemctl daemon-reload
sudo systemctl enable elasticsearch wazuh-manager kibana
sudo systemctl start elasticsearch wazuh-manager kibana

# Status überprüfen
sudo systemctl status elasticsearch wazuh-manager kibana
```

> ⚠ **Wichtige Hinweise:**
>
> - **Sicherheit:** Standardpasswörter ändern, Firewall konfigurieren, SSL/TLS aktivieren.
> - **Performance:** Elasticsearch Heap-Size anpassen, Indexing-Strategien festlegen, Retention-Policies definieren.

- **Monitoring:** Service-Status überwachen, Log-Rotation einrichten, Backup-Strategie implementieren.

Agent-Deployment und Management

Die Wazuh-Agenten sind weit mehr als einfache Datensammler. Sie bilden ein komplexes Netzwerk von Sicherheitssensoren, die nicht nur Daten erfassen, sondern auch aktiv auf Bedrohungen reagieren können.

Agent-Architektur



Agent-Installation

```
# Für Ubuntu/Debian Systeme
```

```
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | apt-key add -
echo "deb https://packages.wazuh.com/4.x/apt/ stable main" | tee
/etc/apt/sources.list.d/wazuh.list
apt update
apt install wazuh-agent
```



Registrierung und Authentifizierung

Die Registrierung eines Agenten ist ein kritischer Sicherheitsprozess, der über die einfache Verbindungsherstellung hinausgeht.

```
# Auf dem Wazuh-Server
sudo /var/ossec/bin/manage_agents -a -n "WebServer01" -i "192.168.1.100"

# Agent-Schlüssel extrahieren
sudo /var/ossec/bin/manage_agents -e <agent_id>
```

```
# Auf dem Agent
echo "<extracted_key>" | sudo /var/ossec/bin/manage_agents -i
```



Gruppenverwaltung

Gruppen in Wazuh sind mehr als nur organisatorische Einheiten. Sie ermöglichen eine granulare Kontrolle über Sicherheitspolicies.

```
# Gruppe erstellen
/var/ossec/bin/agent_groups -a -g webservers

# Agent einer Gruppe zuweisen
/var/ossec/bin/agent_groups -a -i 001 -g webservers
```

```
# Gruppenkonfiguration
cat > /var/ossec/etc/shared/webservers/agent.conf << EOF
<agent_config>
  <localfile>
    <location>/var/log/apache2/access.log</location>
    <log_format>apache</log_format>
  </localfile>
</agent_config>
EOF
```

Remote-Deployment

Automatisiertes Deployment ist in großen Infrastrukturen nicht optional, sondern zwingend erforderlich.

```
# Ansible Playbook Beispiel
- name: Deploy Wazuh Agent
  hosts: all
  tasks:
    - name: Add Wazuh repo
      apt_repository:
        repo: "deb https://packages.wazuh.com/4.x/apt/ stable main"
        state: present

    - name: Install Wazuh Agent
      apt:
        name: wazuh-agent
        state: present
        update_cache: yes

    - name: Configure Agent
      template:
        src: ossec.conf.j2
        dest: /var/ossec/etc/ossec.conf
```

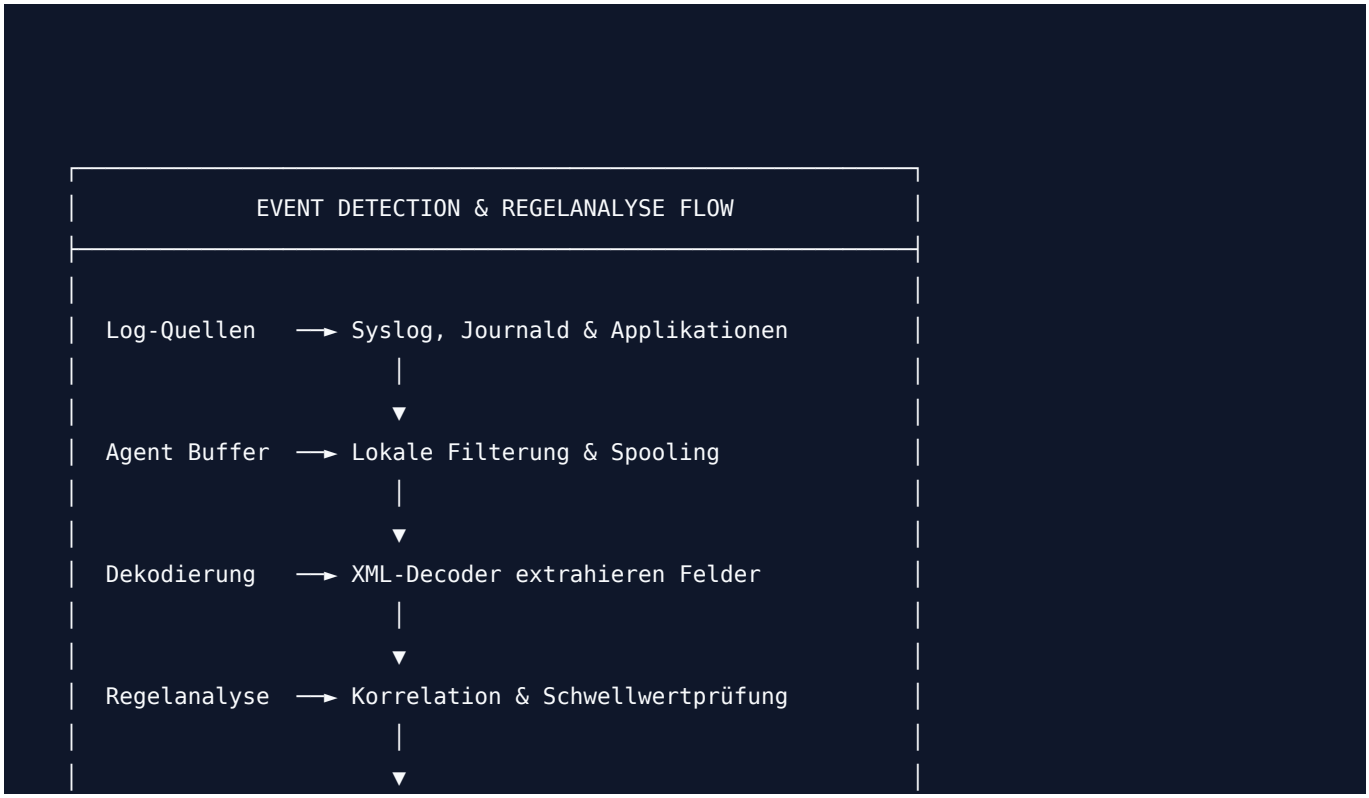
Wichtige Hinweise:

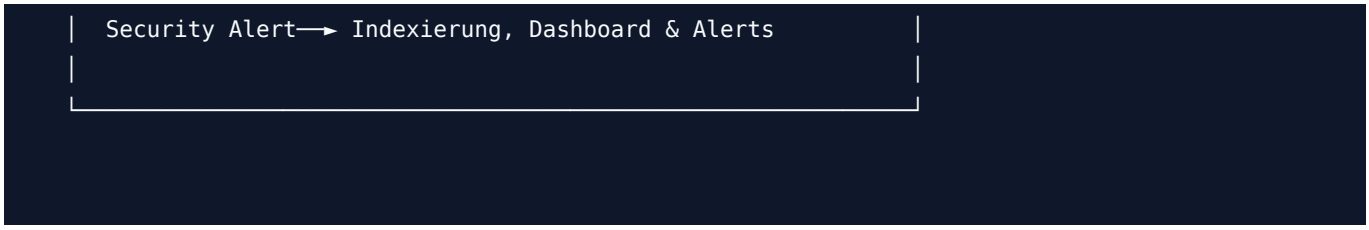
- **Sicherheitsaspekte:** Sichere Kommunikation konfigurieren, zertifikatsbasierte Authentifizierung, regelmäßige Schlüsselrotation.
- **Performance-Optimierung:** Buffer-Größen anpassen, Event-Filterung konfigurieren, Ressourcenlimits setzen.
- **Wartung:** Update-Strategie entwickeln, Monitoring der Agent-Gesundheit, Backup der Agenten-Konfiguration.

Hauptfunktionen

Security Event Detection

Die Erkennung von Sicherheitsereignissen ist weit mehr als simple Logfile-Analyse. In modernen IT-Umgebungen müssen verschiedenste Ereignisquellen in Echtzeit korreliert werden.



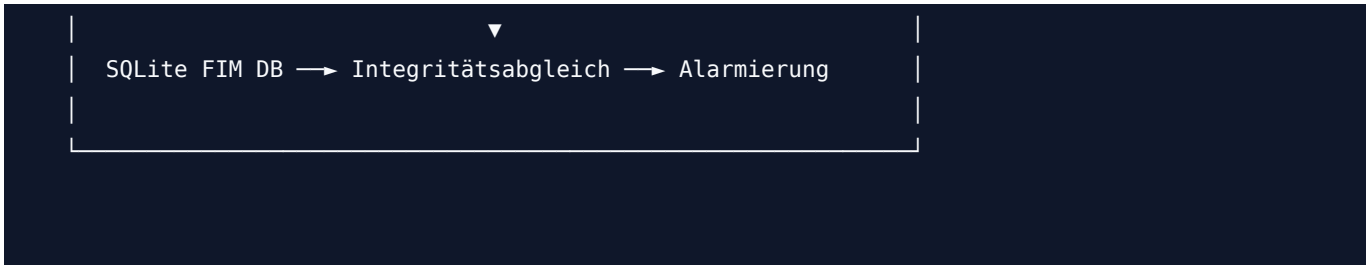



```
<!-- Beispiel einer Regel für SSH-Brute-Force-Erkennung -->
<rule id="100001" level="10">
  <if_sid>5710</if_sid>
  <match>^Failed password</match>
  <frequency>8</frequency>
  <timeframe>120</timeframe>
  <description>Multiple failed login attempts</description>
</rule>
```

File Integrity Monitoring (FIM)

File Integrity Monitoring ist ein kritischer Sicherheitsmechanismus, der weit über simple Checksummen-Vergleiche hinausgeht.

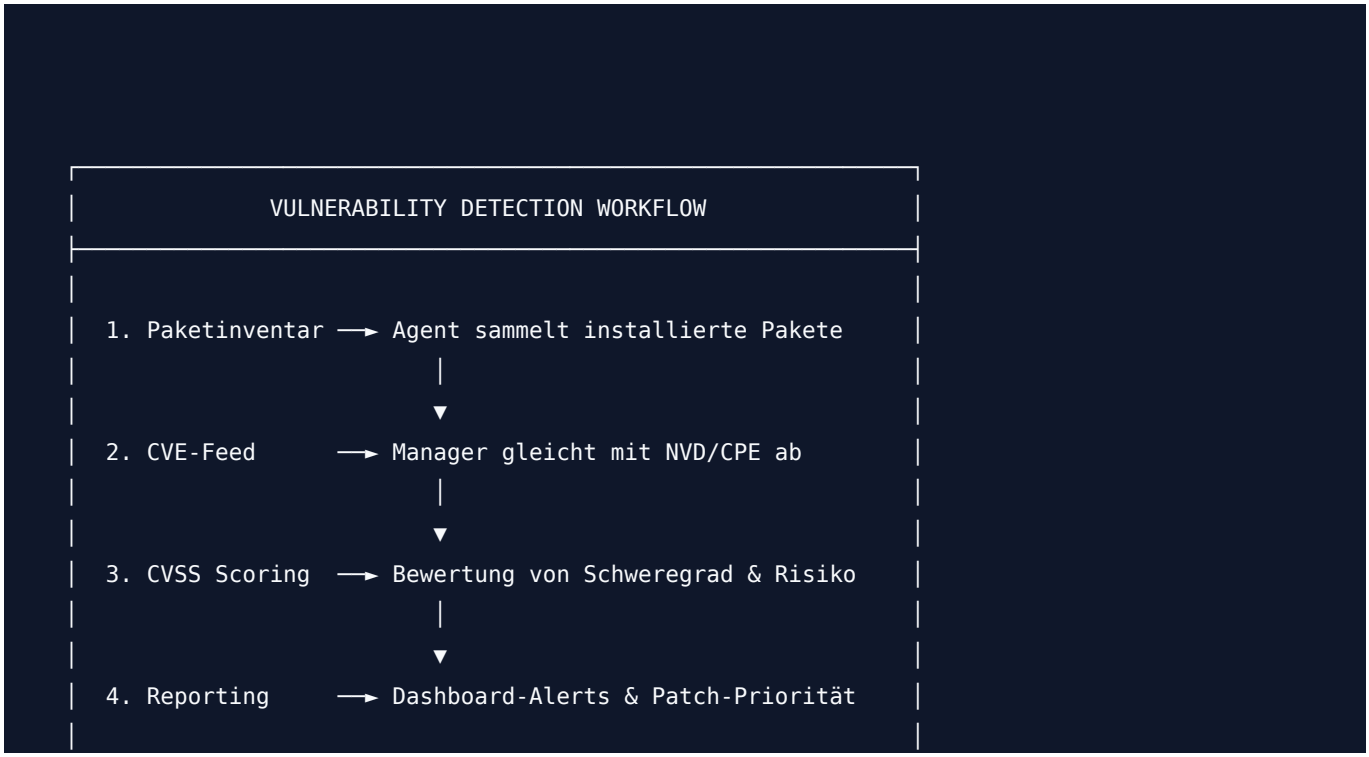





```
<!-- FIM-Konfiguration für kritische Systempfade -->
<syscheck>
  <directories check_all="yes" realtime="yes">/etc,/usr/bin,/usr/sbin</directories>
  <ignore>/etc/mtab</ignore>
  <ignore type="sregex">^/etc/.*\.swp$</ignore>
</syscheck>
```

Vulnerability Detection

Die Schwachstellenerkennung in Wazuh geht weit über einfaches CVE-Matching hinaus.

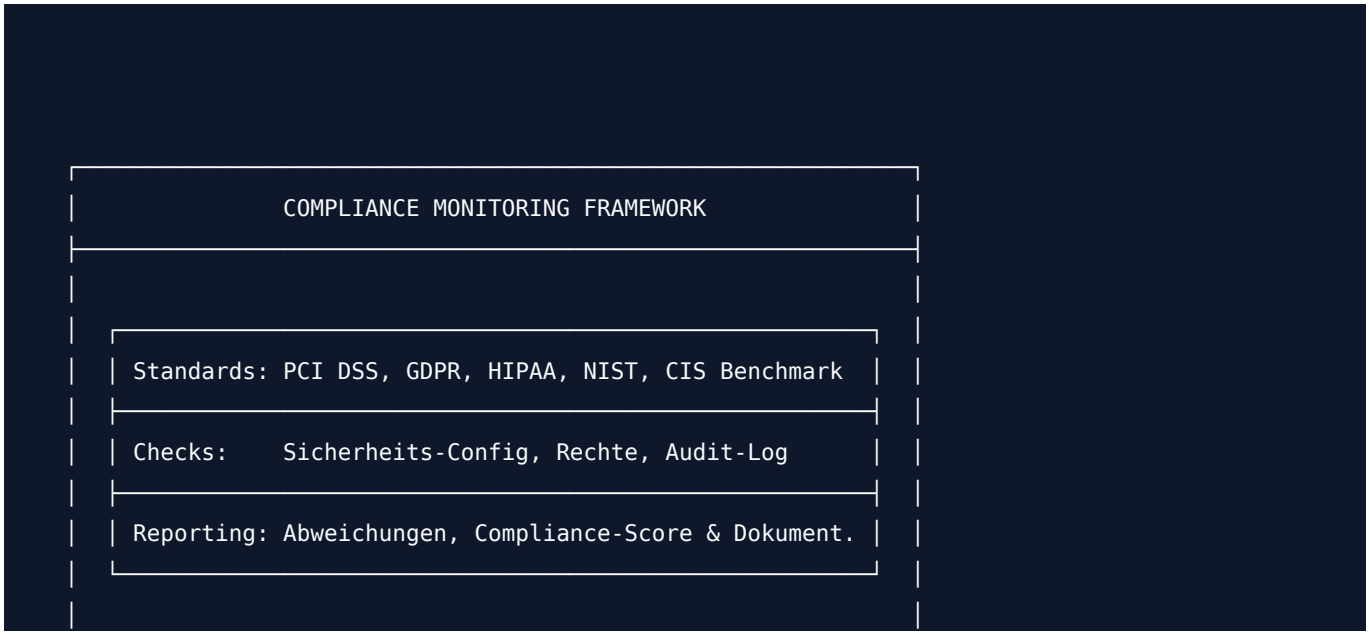


```
└──────────────────────────────────────────────────────────┘
```

```
<!-- Vulnerability Detector Konfiguration -->
<vulnerability-detector>
  <enabled>yes</enabled>
  <interval>1d</interval>
  <feed name="ubuntu-20">
    <enabled>yes</enabled>
    <update_interval>1h</update_interval>
  </feed>
</vulnerability-detector>
```

Compliance Monitoring

Compliance Monitoring in Wazuh ist mehr als nur eine Checkliste von Regelwerken. Es handelt sich um ein dynamisches System, das kontinuierlich die Einhaltung von Sicherheitsstandards überwacht und dokumentiert.



```
└──────────────────────────────────────────────────────┘
```

Active Response

Active Response ist das autonome Immunsystem deiner IT-Infrastruktur. Anders als passive Sicherheitssysteme kann Wazuh automatisch auf erkannte Bedrohungen reagieren.




```
<!-- Active Response Konfiguration -->
<command>
  <name>firewall-block</name>
  <executable>firewall-block.sh</executable>
```

```
    <expect>srcip</expect>
    <timeout_allowed>yes</timeout_allowed>
  </command>

  <active-response>
    <command>firewall-block</command>
    <location>local</location>
    <rules_id>100001</rules_id>
    <timeout>600</timeout>
  </active-response>
```

Wichtige Hinweise:

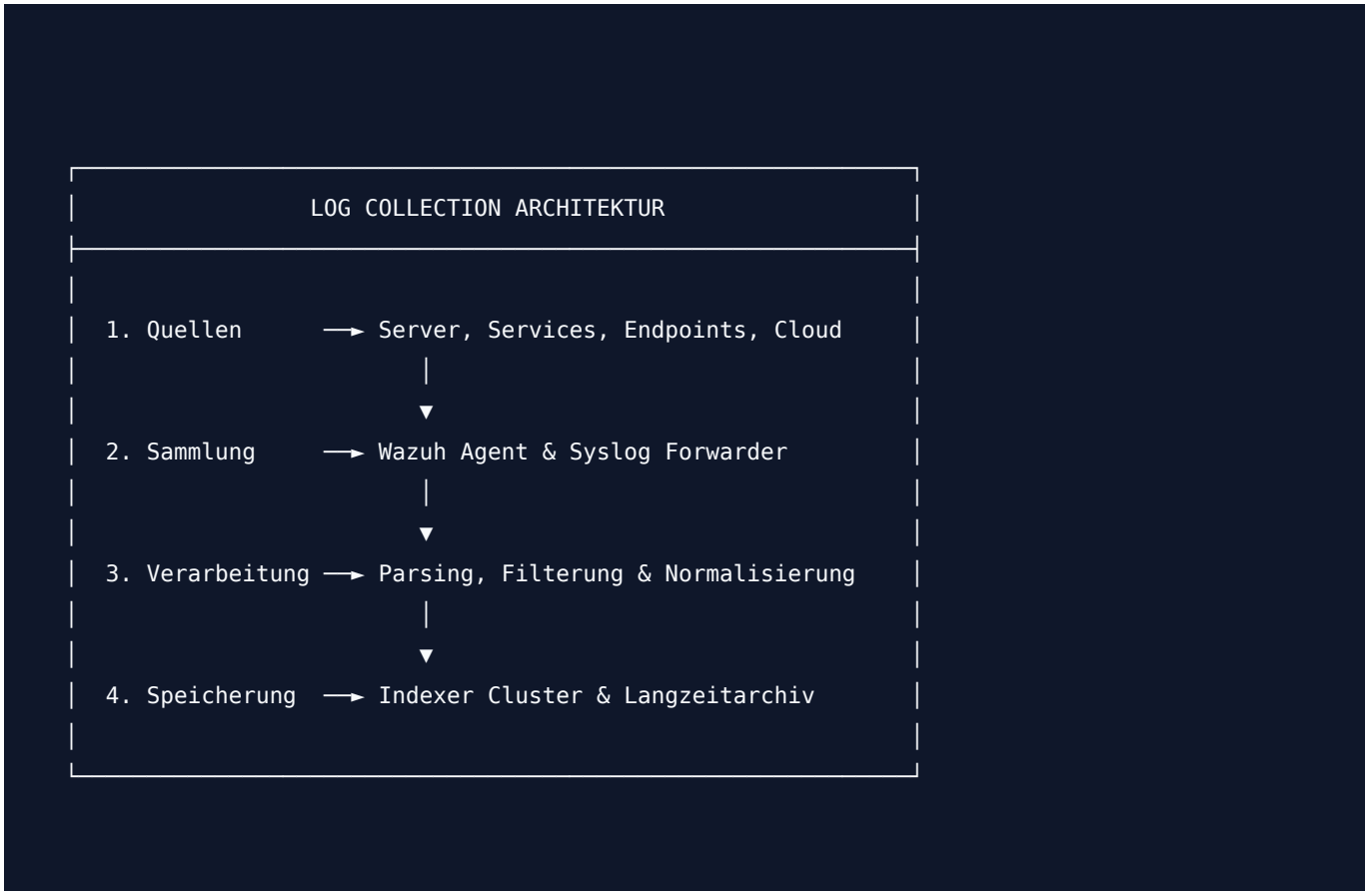
- **Performance-Optimierung:** Regeloptimierung für minimale Systemlast, effiziente FIM-Konfiguration, angepasste Scan-Intervalle.
- **Fehlalarme minimieren:** Regeltuning basierend auf Umgebung, Whitelist-Management, kontext-basierte Analyse.
- **Integration:** API-Nutzung für Automatisierung, Custom Response Scripts, External Tool Integration.

Integration und Monitoring

Ein durchdachtes Monitoring- und Integrationskonzept verbindet alle Sicherheitskomponenten zu einem kohärenten System.

Log Collection

Die Protokollsammlung in Wazuh ist ein ausgeklügeltes System zur Erfassung, Normalisierung und Korrelation von Ereignisdaten aus verschiedensten Quellen.




```
<!-- Beispiel einer erweiterten Log-Collection-Konfiguration -->
<localfile>
  <location>/var/log/auth.log</location>
  <log_format>syslog</log_format>
  <frequency>10</frequency>
  <only-future-events>yes</only-future-events>
  <query>user failed|authentication failure</query>
</localfile>
```

Alert-Konfiguration

Die Alert-Konfiguration ist das Herzstück deines Sicherheitsmonitorings.



Custom Rules erstellen

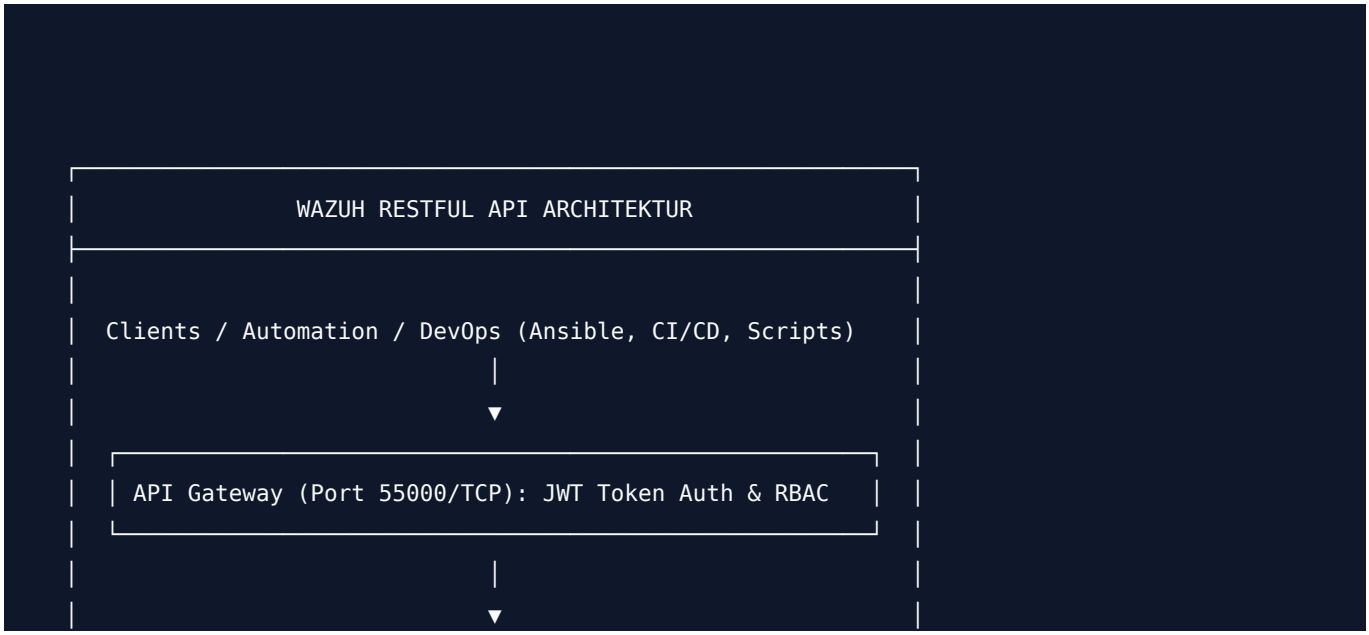
Die Erstellung benutzerdefinierter Regeln ist eine der mächtigsten Funktionen in Wazuh.



```
<!-- Beispiel einer komplexen Custom Rule -->
<rule id="100100" level="12">
  <if_sid>5710</if_sid>
  <match>^Failed password</match>
  <regex>user: (\w+)</regex>
  <frequency>8</frequency>
  <timeframe>120</timeframe>
  <if_matched_group>admin|root</if_matched_group>
  <description>Brute force attack on privileged account</description>
  <mitre>
    <id>T1110</id>
    <tactic>Credential Access</tactic>
  </mitre>
</rule>
```

API-Nutzung

Die Wazuh-API ist das Fundament für Automation und Integration. Die RESTful API ermöglicht nicht nur die Abfrage von Daten, sondern auch die programmatische Steuerung aller Wazuh-Komponenten.



```
    │  Wazuh Manager: Konfiguration, Agentensteuerung, Abfragen   │
    │                                                             │
    └─────────────────────────────────────────────────────────────┘
```

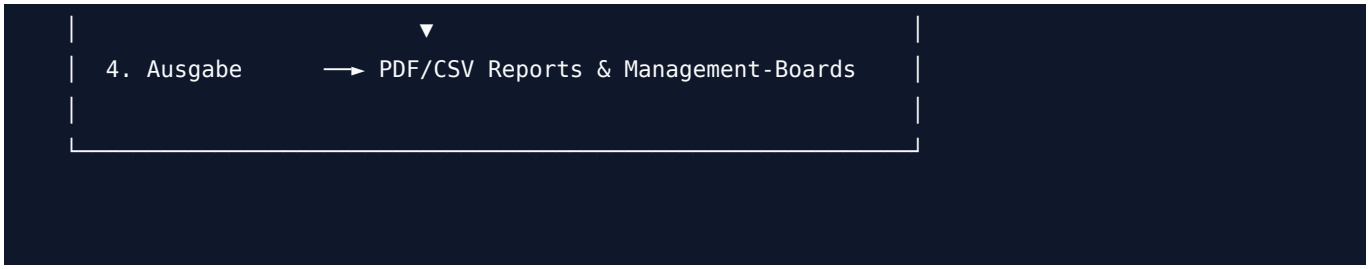
```
# Beispiel für API-Interaktion
# Token generieren
TOKEN=$(curl -u admin:admin -k -X GET "https://localhost:55000/security/user/authenticate"
| jq -r '.data.token')

# Agenten-Status abfragen
curl -k -X GET "https://localhost:55000/agents" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json"
```

Reporting

Das Reporting-System in Wazuh bietet eine umfassende Analyse- und Dokumentationsplattform.






```
<!-- Report-Konfiguration Beispiel -->
<report>
  <title>Security Compliance Report</title>
  <schedule type="weekly" day="1" time="23:59">
  <format>pdf</format>
  <sections>
    - compliance_pci
    - top_alerts
    - agent_status
  </sections>
  <mail_to>security@domain.com</mail_to>
</report>
```

Wichtige Integrationsaspekte:

- **Datenkonsistenz:** Einheitliche Datenformate, verlustfreie Übertragung, Datenvalidierung.
- **Skalierbarkeit:** Lastverteilung, Pufferung, Performance-Optimierung.
- **Sicherheit:** Verschlüsselte Kommunikation, Zugriffskontrollen, Audit-Logging.

Best Practices und Use Cases

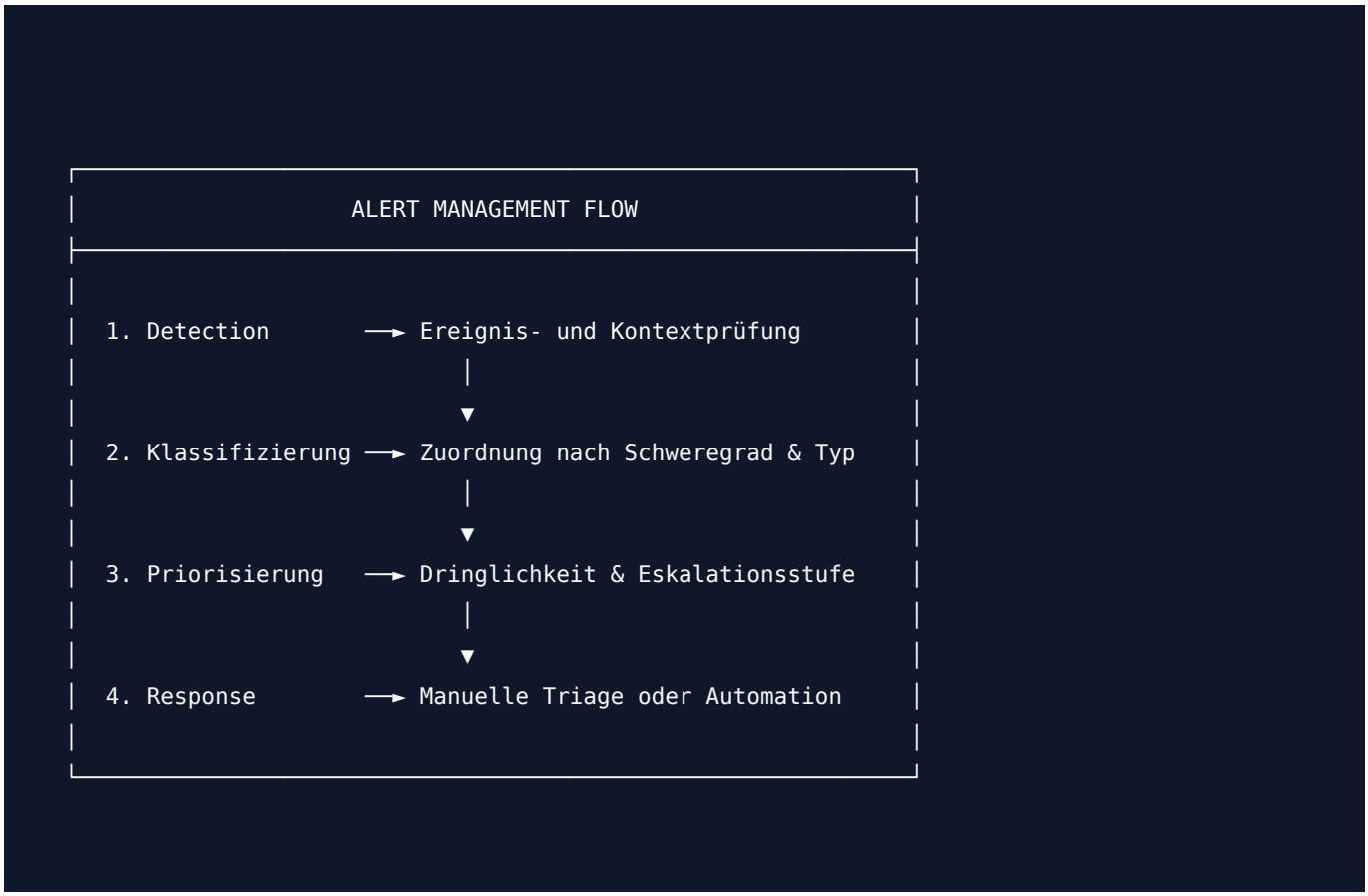
Security Policies

Sicherheitsrichtlinien in Wazuh sind das Fundament deiner Sicherheitsstrategie.

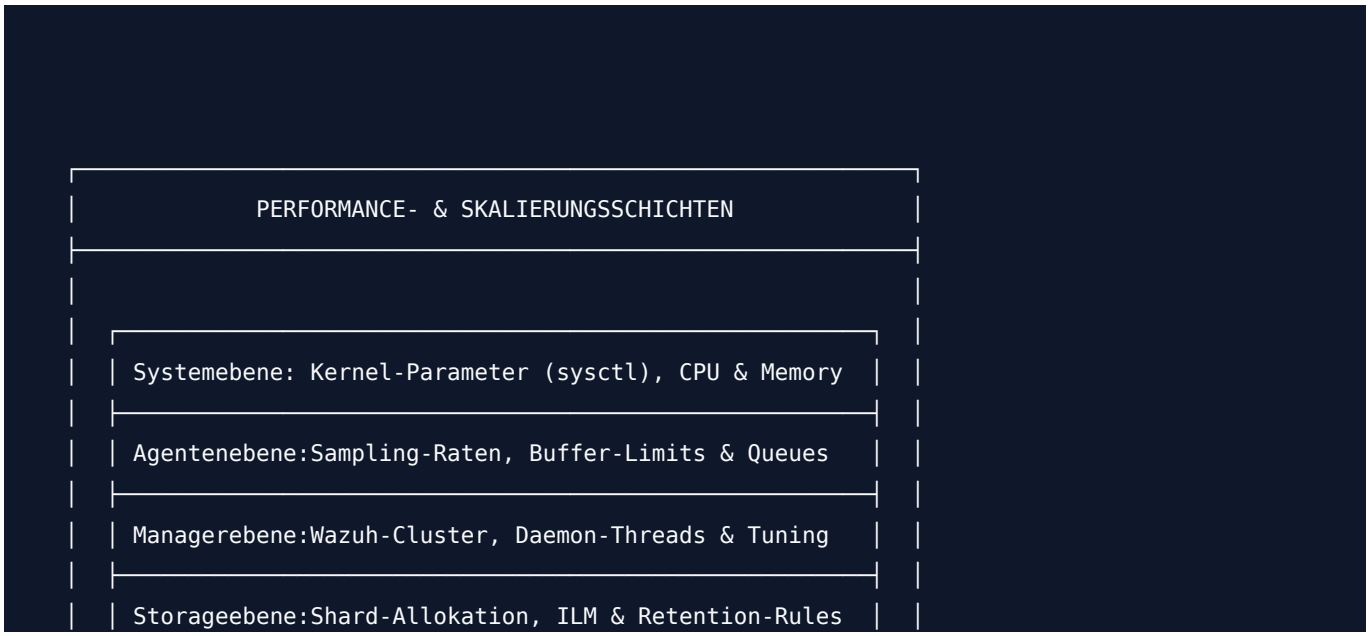



```
<!-- Beispiel einer umfassenden Security Policy -->
<policy>
  <name>Critical Server Policy</name>
  <rules>
    - system_hardening
    - access_control
    - network_protection
  </rules>
  <exceptions>
    - maintenance_window
    - authorized_changes
  </exceptions>
  <actions>
    - alert_critical
    - block_access
    - notify_admin
  </actions>
</policy>
```

Alert-Management



Performance-Optimierung



```
    │  └──────────────────────────────────────────────────────┘  │
    │                                                            │
    └────────────────────────────────────────────────────────────┘
```

```
<!-- Beispiel Performance-Optimierung -->
<ossec_config>
  <global>
    <memory_size>2048</memory_size>
    <queue_size>131072</queue_size>
    <events_per_second>1000</events_per_second>
  </global>

  <syscheck>
    <frequency>43200</frequency>
    <max_files_per_second>50</max_files_per_second>
  </syscheck>
</ossec_config>
```

Incident Response

```
┌────────────────────────────────────────────────────────────┐
│                INCIDENT RESPONSE WORKFLOW                  │
├────────────────────────────────────────────────────────────┤
│                                                            │
│  1. Erkennung   ──► Alerting, Log-Spike oder IOC-Fund      │
│                         │                                  │
│                         ▼                                  │
│  2. Analyse     ──► Forensik, Timeline & Korrelation       │
│                         │                                  │
│                         ▼                                  │
│  3. Eindämmung  ──► Isolation des Hosts, Quarantäne        │
```

```
    |                      |                                      |
    |                      ▼                                      |
    |  4. Wiederherst.──► Bereinigung, Patches & Härtung          |
    |                                                             |
    └─────────────────────────────────────────────────────────────┘
```

```
# Beispiel Incident Response Script
#!/bin/bash
case "$1" in
  'brute_force')
    iptables -A INPUT -s "$2" -j DROP
    logger "Blocked IP $2 due to brute force attack"
    notify_security_team "Brute force attack from $2"
    ;;
  'malware')
    isolate_system "$2"
    run_emergency_scan "$2"
    collect_forensics "$2"
    ;;
esac
```

Compliance-Checking

```
<!-- Compliance Check Konfiguration -->
<rootcheck>
  <check_unixaudit>yes</check_unixaudit>
  <check_files>yes</check_files>
  <check_trojans>yes</check_trojans>
  <check_dev>yes</check_dev>
  <check_sys>yes</check_sys>
  <check_pids>yes</check_pids>
```

```
    <check_ports>yes</check_ports>
    <check_if>yes</check_if>

    <system_audit>/var/ossec/etc/shared/cis_debian_linux_rcl.txt</system_audit>
    <system_audit>/var/ossec/etc/shared/pci_dss_rcl.txt</system_audit>
  </rootcheck>
```

> **Kritische Erfolgsfaktoren:**
>
> - **Prozessintegration:** Dokumentierte Verfahren, klare Verantwortlichkeiten, regelmäßige Überprüfungen.
> - **Ressourcenmanagement:** Kapazitätsplanung, Performance-Monitoring, Skalierungsstrategie.
> - **Kontinuierliche Verbesserung:** Feedback-Loops, Metriken-basierte Optimierung, regelmäßige Evaluierung.

Compliance (PCI DSS, GDPR)

Compliance in modernen IT-Umgebungen ist mehr als nur das Abhaken von Checklisten. Wazuh bietet spezifische Funktionen für verschiedene Standards.

PCI DSS Compliance:

```
<!-- PCI DSS spezifische Konfiguration -->
<pci_dss>
  <requirement_10>
    <rule_id>80700</rule_id>
    <description>Log all access to cardholder data</description>
    <monitoring>
      - file_integrity
      - access_control
      - audit_logs
```

```
    </monitoring>
  </requirement_10>
</pci_dss>
```

GDPR Compliance:

```
<!-- GDPR Monitoring Beispiel -->
<gdpr>
  <article_32>
    <security_measures>
      - encryption_at_rest
      - encryption_in_transit
      - access_logging
      - breach_detection
    </security_measures>
    <monitoring>
      - data_access
      - unauthorized_changes
      - security_events
    </monitoring>
  </article_32>
</gdpr>
```

> **□□ Compliance Best Practices:**
>
> - **Dokumentation:** Lückenlose Nachweisführung, automatisierte Berichtserstellung, Vorfallsdokumentation.
> - **Kontinuierliche Überwachung:** Real-time Compliance Monitoring, automatische Benachrichtigungen, regelmäßige Audits.
> - **Anpassungsfähigkeit:** Flexible Regelanpassung, skalierbare Kontrollen, Update-Management.

Troubleshooting und Wartung

Performance-Monitoring

Die Leistungsüberwachung einer Wazuh-Installation erfordert einen ganzheitlichen Ansatz.

```
# Umfassendes Performance-Monitoring-Script
#!/bin/bash
echo "=== System Resources ==="
ps aux | grep -E "wazuh|ossec" | awk '{print $2, $3, $4, $11}'
free -m
vmstat 1 5
iostat -x 1 5 | grep -E "Device|sda"
netstat -anp | grep -E "wazuh|ossec"
```

Debugging

```
# Erweitertes Debug-Script
#!/bin/bash
echo "=== Wazuh Debug Analysis ==="

debug_level_check() {
    echo "Checking Debug Levels..."
    grep -r "debug" /var/ossec/etc/internal_options.conf
    grep -r "debug" /var/ossec/etc/local_internal_options.conf
}

service_health_check() {
    echo "Analyzing Service Health..."
    for service in wazuh-manager filebeat elasticsearch kibana; do
        echo "=== $service Status ==="
```

```
            systemctl status $service --no-pager
            journalctl -u $service --no-pager | tail -n 50
        done
    }

    config_validation() {
        echo "Validating Configurations..."
        /var/ossec/bin/ossec-logtest -t
        /var/ossec/bin/ossec-analysisd -t
    }

    debug_level_check
    service_health_check
    config_validation
```

Backup & Recovery

```
    # Umfassendes Backup-Script
    #!/bin/bash
    BACKUP_DIR="/backup/wazuh/$(date +%Y%m%d)"
    CONFIG_DIR="/var/ossec/etc"
    DATA_DIR="/var/ossec/data"

    mkdir -p "$BACKUP_DIR"/{config,data,state}

    backup_config() {
        echo "Backing up configurations..."
        cp -r "$CONFIG_DIR"/* "$BACKUP_DIR/config/"
        tar czf "$BACKUP_DIR/config.tar.gz" -C "$BACKUP_DIR" config/
    }

    backup_data() {
        echo "Backing up data..."
        systemctl stop wazuh-manager
        cp -r "$DATA_DIR"/* "$BACKUP_DIR/data/"
        tar czf "$BACKUP_DIR/data.tar.gz" -C "$BACKUP_DIR" data/
        systemctl start wazuh-manager
```

```
}

backup_state() {
    echo "Saving system state..."
    systemctl status wazuh-* > "$BACKUP_DIR/state/services.txt"
    netstat -tulpn > "$BACKUP_DIR/state/network.txt"
    ps aux | grep -E "wazuh|ossec" > "$BACKUP_DIR/state/processes.txt"
}

{
    backup_config
    backup_data
    backup_state
} 2>"$BACKUP_DIR/backup.log"
```

Updates & Upgrades

Die Aktualisierung einer Wazuh-Installation ist ein kritischer Prozess, der sorgfältige Planung, Testing und Validierung erfordert.

```
# Update-Script mit Sicherheitsmaßnahmen
#!/bin/bash
UPDATE_LOG="/var/log/wazuh_update.log"
BACKUP_DIR="/backup/wazuh/pre_update_$(date +%Y%m%d)"
WAZUH_VERSION="4.4.0"

log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$UPDATE_LOG"
}

pre_update_check() {
    log "Starting pre-update checks..."
    if ! systemctl is-active --quiet wazuh-manager; then
        log "ERROR: Wazuh-Manager is not running"
        exit 1
    fi
    FREE_SPACE=$(df -m /var/ossec | awk 'NR==2 {print $4}')
```

```
    if [ "$FREE_SPACE" -lt 5000 ]; then
        log "ERROR: Insufficient disk space"
        exit 1
    fi
}

perform_backup() {
    log "Creating pre-update backup..."
    tar czf "$BACKUP_DIR/etc.tar.gz" /var/ossec/etc/
    /var/ossec/bin/wazuh-control info > "$BACKUP_DIR/version_info.txt"
}

perform_update() {
    log "Starting update process..."
    systemctl stop wazuh-manager
    apt-get update
    if ! apt-get install wazuh-manager="$WAZUH_VERSION"; then
        log "ERROR: Update failed"
        restore_from_backup
        exit 1
    fi
}

validate_update() {
    log "Validating update..."
    systemctl start wazuh-manager
    sleep 10
    if ! systemctl is-active --quiet wazuh-manager; then
        log "ERROR: Service failed to start after update"
        restore_from_backup
        exit 1
    fi
}

restore_from_backup() {
    log "Initiating rollback procedure..."
    systemctl stop wazuh-manager
    tar xzf "$BACKUP_DIR/etc.tar.gz" -C /
    systemctl start wazuh-manager
    log "Rollback completed"
}

{
    log "Starting Wazuh update process to version $WAZUH_VERSION"
```

```
    pre_update_check
    perform_backup
    perform_update
    validate_update
    log "Update completed successfully"
} || {
    log "Update failed - check logs for details"
    exit 1
}
```

> ⚠ **Kritische Update-Aspekte:**
>
> - **Vorbereitung:** Vollständige Systemanalyse, Backup-Validierung, Abhängigkeitscheck, Ressourcenplanung.
> - **Durchführung:** Schrittweise Migration, kontinuierliche Überwachung, Fehlerprotokollierung, Rollback-Bereitschaft.
> - **Nachbereitung:** Funktionstest aller Komponenten, Performance-Analyse, Dokumentation der Änderungen.

Systemüberwachung und Log-Management

```
# Erweitertes System-Monitoring-Script
#!/bin/bash
LOG_DIR="/var/log/wazuh-monitoring"
RETENTION_DAYS=30

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" >> "$LOG_DIR/system_health.log"
}

monitor_resources() {
```

```
    log_message "=== System Resource Check ==="
    CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2}')
    CPU_WAZUH=$(ps aux | grep ossec | awk '{sum+=$3} END {print sum}')
    log_message "CPU Usage: Total=${CPU_USAGE}%, Wazuh=${CPU_WAZUH}%"

    MEMORY_TOTAL=$(free -m | awk 'NR==2 {print $2}')
    MEMORY_USED=$(free -m | awk 'NR==2 {print $3}')
    MEMORY_WAZUH=$(ps aux | grep ossec | awk '{sum+=$4} END {print sum}')
    log_message "Memory: Total=${MEMORY_TOTAL}MB, Used=${MEMORY_USED}MB,
Wazuh=${MEMORY_WAZUH}%"
}

manage_logs() {
    log_message "=== Log Management ==="
    find "$LOG_DIR" -type f -name "*.log" -mtime +$RETENTION_DAYS -exec rm {} \;

    for LOG_FILE in /var/ossec/logs/*.log; do
        SIZE=$(du -h "$LOG_FILE" | cut -f1)
        ENTRIES=$(wc -l < "$LOG_FILE")
        ERRORS=$(grep -c "error" "$LOG_FILE")
        WARNINGS=$(grep -c "warning" "$LOG_FILE")
        log_message "Log File: $LOG_FILE - Size: $SIZE - Entries: $ENTRIES - Errors:
$ERRORS - Warnings: $WARNINGS"
    done
}

analyze_alerts() {
    log_message "=== Alert Analysis ==="
    ALERTS_TODAY=$(grep -c "$(date +%Y/%m/%d)" /var/ossec/logs/alerts/alerts.log)
    FALSE_POSITIVES=$(grep -c "level 0" /var/ossec/logs/alerts/alerts.log)
    CRITICAL_ALERTS=$(grep -c "level [789]" /var/ossec/logs/alerts/alerts.log)
    log_message "Alert Statistics: Total=$ALERTS_TODAY, False Positives=$FALSE_POSITIVES,
Critical=$CRITICAL_ALERTS"
}

{
    mkdir -p "$LOG_DIR"
    monitor_resources
    manage_logs
    analyze_alerts
}
```

Ressourcenverwaltung

Die Balance zwischen Systemressourcen und Überwachungskapazität entscheidet über die Effektivität deiner Wazuh-Installation.

```
# Umfassendes Ressourcen-Management-Script
#!/bin/bash
WAZUH_PATH="/var/ossec"
ES_PATH="/var/lib/elasticsearch"
ALERT_THRESHOLD=85

monitor_resources() {
    DISK_USAGE=$(df -h "$WAZUH_PATH" | awk 'NR==2 {print $5}' | sed 's/%//')
    ES_DISK_USAGE=$(df -h "$ES_PATH" | awk 'NR==2 {print $5}' | sed 's/%//')

    if [ "$DISK_USAGE" -gt "$ALERT_THRESHOLD" ] || [ "$ES_DISK_USAGE" -gt
"$ALERT_THRESHOLD" ]; then
        find "$WAZUH_PATH/logs/archives" -type f -mtime +30 -delete
        curl -XPOST "localhost:9200/_forcemerge?only_expunge_deletes=true"
        curl -XPOST "localhost:9200/_cache/clear"
    fi
}

optimize_shards() {
    CLUSTER_STATUS=$(curl -s "localhost:9200/_cluster/health" | jq -r '.status')
    if [ "$CLUSTER_STATUS" != "green" ]; then
        curl -XPUT "localhost:9200/_cluster/settings" -H 'Content-Type: application/json'
-d'
        {
            "transient": {
                "cluster.routing.allocation.disk.threshold_enabled": true,
                "cluster.routing.allocation.disk.watermark.low": "85%",
                "cluster.routing.allocation.disk.watermark.high": "90%",
                "cluster.routing.allocation.disk.watermark.flood_stage": "95%"
            }
        }'
    fi
}
```

```
optimize_performance() {
    cat > "$WAZUH_PATH/etc/local_internal_options.conf" << EOF
    wazuh_database.sync_agent_info_max_time=10m
    analysisd.event_threads=4
    remoted.recv_counter_flush=128
    remoted.comp_average_printout=0
EOF

    cat > "/etc/elasticsearch/jvm.options.d/heap.options" << EOF
    -Xms4g
    -Xmx4g
EOF
}

{
    echo "Starting Resource Management at $(date)"
    monitor_resources
    optimize_shards
    optimize_performance
    echo "Resource Management completed at $(date)"
} >> /var/log/wazuh-resource-management.log 2>&1
```

Netzwerkoptimierung

```
# Netzwerk-Optimierungs-Script
#!/bin/bash
WAZUH_CONF="/var/ossec/etc/ossec.conf"

optimize_network() {
    cat > /etc/sysctl.d/99-wazuh-network.conf << EOF
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_sack = 1
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 87380 16777216
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_max_tw_buckets = 1440000
```

```
net.core.somaxconn = 1024
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.rmem_default = 87380
net.core.wmem_default = 87380
EOF
    sysctl -p /etc/sysctl.d/99-wazuh-network.conf
}

optimize_communication() {
    cat > "$WAZUH_CONF.tmp" << EOF
<ossec_config>
  <remote>
    <connection>secure</connection>
    <port>1514</port>
    <protocol>tcp</protocol>
    <queue_size>131072</queue_size>
  </remote>
  <global>
    <jsonout_output>yes</jsonout_output>
    <alerts_log>yes</alerts_log>
    <logall>no</logall>
    <memory_size>8192</memory_size>
    <process_pool_size>4</process_pool_size>
  </global>
</ossec_config>
EOF

    if /var/ossec/bin/ossec-logtest -c "$WAZUH_CONF.tmp"; then
        mv "$WAZUH_CONF.tmp" "$WAZUH_CONF"
        systemctl restart wazuh-manager
    else
        rm "$WAZUH_CONF.tmp"
    fi
}

optimize_network
optimize_communication
```

> **□□ Kritische Netzwerk-Aspekte:**

- **Kommunikationsoptimierung:** TCP-Stack-Tuning, Buffer-Größen-Anpassung, Protokoll-Optimierung, Verbindungsmanagement.
- **Sicherheitsaspekte:** Verschlüsselte Kommunikation, Zertifikatsmanagement, Port-Sicherheit, Traffic-Kontrolle.
- **Performance-Monitoring:** Latenz-Überwachung, Durchsatz-Analyse, Verbindungsqualität, Ressourcennutzung.

Praktische Beispiele

Brute-Force-Erkennung

```
<!-- Kontextbasierte Brute-Force-Erkennung -->
<group name="brute_force">
  <rule id="100100" level="10">
    <if_sid>5710</if_sid>
    <match>^Failed password|Failed auth</match>
    <frequency>8</frequency>
    <timeframe>120</timeframe>
    <if_matched_ip>known_malicious_ips</if_matched_ip>
    <description>SSH authentication failures from suspicious IP.</description>
    <mitre>
      <id>T1110</id>
      <tactic>Credential Access</tactic>
    </mitre>
  </rule>

  <rule id="100101" level="12">
    <if_matched_sid>100100</if_matched_sid>
    <same_source_ip />
    <frequency>2</frequency>
    <timeframe>3600</timeframe>
    <if_matched_group>authentication_success</if_matched_group>
    <description>Pattern suggests sophisticated brute-force attempt.</description>
```

```
  </rule>
</group>
```

Malware-Detection

```
<!-- Erweiterte Malware-Erkennung -->
<group name="malware_detection">
  <rule id="100200" level="12">
    <if_group>syscheck</if_group>
    <match>WinExec|CreateRemoteThread|WriteProcessMemory</match>
    <regex>\.exe$|\.dll$|\.ps1$</regex>
    <description>Suspicious file operation detected</description>
    <mitre>
      <id>T1055</id>
      <tactic>Execution</tactic>
    </mitre>
  </rule>

  <rule id="100201" level="14">
    <if_sid>100200</if_sid>
    <if_matched_group>network_connection</if_matched_group>
    <timeframe>60</timeframe>
    <description>Process injection with network activity</description>
  </rule>
</group>
```

```
# Malware-Detection Script
#!/bin/bash
monitor_processes() {
    while true; do
        ps auxf | grep -E "cmd\.exe.*powershell|bash.*nc" >>
```

```
    /var/log/suspicious_processes.log
            top -b -n 1 | grep -E "crypto|miner" >> /var/log/cpu_anomalies.log
            netstat -tupan | grep -E ":(4444|8080|6666)" >> /var/log/network_anomalies.log
            sleep 60
        done
    }

    yara_scan() {
        for dir in /bin /usr/bin /tmp; do
            yara /var/ossec/etc/rules/malware_rules.yar "$dir" >> /var/log/yara_detections.log
        done
    }

    {
        monitor_processes &
        while true; do
            yara_scan
            sleep 3600
        done
    } 2>&1 | /var/ossec/bin/ossec-csyslogd
```

Custom Rules

```
    <!-- Fortgeschrittene Custom Rules -->
    <group name="custom_detection">
      <rule id="100400" level="3">
        <decoded_as>sshd</decoded_as>
        <match>authentication success</match>
        <description>SSH Login erfolgreich</description>
      </rule>

      <rule id="100401" level="8">
        <if_sid>100400</if_sid>
        <if_matched_group>software_change</if_matched_group>
        <timeframe>300</timeframe>
        <description>Verdächtige Aktivität nach SSH-Login</description>
        <mitre>
```

```
      <id>T1078</id>
      <tactic>Initial Access</tactic>
    </mitre>
  </rule>

  <rule id="100402" level="12">
    <if_group>web</if_group>
    <match>SELECT.*UNION|INSERT.*INTO|UPDATE.*SET</match>
    <regex>\.php$|\.asp$</regex>
    <description>SQL Injection Versuch erkannt</description>
    <options>no_log</options>
    <group>attack,sql_injection</group>
  </rule>
</group>
```

```
#!/bin/bash
RULES_DIR="/var/ossec/etc/rules"
BACKUP_DIR="/var/ossec/backup/rules"

create_rule_template() {
    local RULE_ID="$1"
    local DESCRIPTION="$2"
    cat << EOF > "$RULES_DIR/custom_${RULE_ID}.xml"
<group name="custom_${RULE_ID}">
    <rule id="${RULE_ID}" level="0">
        <if_sid>530</if_sid>
        <description>${DESCRIPTION}</description>
        <mitre>
            <id>TXXXX</id>
            <tactic>Detection</tactic>
        </mitre>
    </rule>
</group>
EOF
}

test_rule() {
    local RULE_FILE="$1"
```

```
    if ! /var/ossec/bin/ossec-analysisd -t -c "$RULE_FILE"; then
        echo "Syntax error in rule file"
        return 1
    fi
    logger "TEST: Custom rule test event for $RULE_FILE"
}

case "$1" in
    "create") create_rule_template "$2" "$3" ;;
    "test") test_rule "$2" ;;
    *) echo "Usage: $0 {create|test} [rule_id] [description]"; exit 1 ;;
esac
```

Regel-Entwicklung:

- **Entwurfsprinzipien:** Präzise Bedingungen, minimale False Positives, effiziente Verarbeitung, skalierbare Struktur.
- **Implementierung:** Regelgruppen-Organisation, Abhängigkeitsmanagement, Performance-Optimierung, Fehlerbehandlung.
- **Wartung:** Regelmäßige Überprüfung, Performance-Monitoring, Anpassung an neue Bedrohungen, Dokumentation.

Befehlsreferenz (Cheatsheet)

Für den schnellen Zugriff im Administrations- und Incident-Response-Alltag fasst die folgende Referenztabelle die wichtigsten Wazuh-Kommandos zur Dienststeuerung, Agentenverwaltung und Regelerkennung zusammen:

Befehl / Syntax	Beschreibung & Praxiseinsatz
`/var/ossec/bin/wazuh-control status`	Zeigt den aktuellen Laufzeitstatus aller Wazuh-Manager-Daemons an.

Befehl / Syntax	Beschreibung & Praxiseinsatz
`/var/ossec/bin/wazuh-control restart`	Führt einen sauberen Neustart aller Manager-Dienste durch.
`/var/ossec/bin/manage_agents -l`	Listet alle registrierten Agents inklusive ID, Name, IP und Status auf.
`/var/ossec/bin/manage_agents -a -n <name> -i <ip>`	Fügt einen neuen Agenten manuell zur Server-Datenbank hinzu.
`/var/ossec/bin/manage_agents -e <agent_id>`	Extrahiert den symmetrischen Authentifizierungsschlüssel für einen Agenten.
`/var/ossec/bin/agent_control -l`	Zeigt aktive und mit dem Manager verbundene Endgeräte an.
`/var/ossec/bin/agent_control -i <agent_id>`	Gibt detaillierte System- und Verbindungsinformationen eines Agenten aus.
`/var/ossec/bin/agent_control -r -u <agent_id>`	Startet den FIM- (Syscheck) oder Rootkit-Scan auf dem Zielagenten neu.
`/var/ossec/bin/wazuh-logtest`	Interaktives Testwerkzeug zur Überprüfung von Logs gegen Decoder und Regeln.
`/var/ossec/bin/wazuh-analysisd -t`	Validiert die Syntax der XML-Regelwerke und Dekoder vor dem Reload.
`/var/ossec/bin/syscheck_control -i <agent_id>`	Zeigt die File-Integrity-Monitoring (FIM) Datenbank des Agenten an.
`/var/ossec/bin/rootcheck_control -i <agent_id>`	Listet gefundene Systemanomalien und Rootkit-Verdachtsfälle auf.
`systemctl status wazuh-manager`	Prüft den Systemd-Dienststatus des Wazuh-Managers.
`systemctl status wazuh-agent`	Prüft den Systemd-Dienststatus des lokalen Wazuh-Agenten.
`systemctl status wazuh-indexer`	Prüft den Cluster- und Indizierungsdienst (Wazuh Indexer).

Befehl / Syntax	Beschreibung & Praxiseinsatz
`systemctl status wazuh-dashboard`	Kontrolliert die Web-Benutzeroberfläche (Wazuh Dashboard).

Weiterführende Ressourcen

Die folgenden offiziellen Dokumentationen, Frameworks und Community-Projekte vertiefen die im Artikel behandelten Sicherheits- und Automatisierungskonzepte:

Ressource	Beschreibung
Wazuh Dokumentation	Offizielle technische Dokumentation zu Installation, Konfiguration und Modulen.
Elastic Stack Docs	Handbücher und API-Referenzen für Elasticsearch, Filebeat und Logstash.
MITRE ATT&CK Framework	Weltweite Wissensdatenbank für Taktiken und Techniken von Cyberangreifern.
OSSEC	Open Source HIDS-Basis, auf der die Wazuh-Architektur historisch aufbaut.
Suricata IDS	Hochperformantes Network Threat Detection und Deep Packet Inspection System.
ELK Stack	Zentrale Log-Management- und Analyseplattform von Elastic.
Wazuh Google Group	Offizielles Diskussionsforum für Administratoren und Entwickler.
Wazuh GitHub Repository	Quellcode, Release-Branches, Regelwerke und Issue-Tracker.
PCI DSS	Offizieller Sicherheitsstandard für den Schutz von Zahlungskartendaten.
GDPR / DSGVO	Datenschutz-Grundverordnung und Anforderungen an Incident Logging.
HIPAA	US-Gesundheitsdatenschutz-Standard für Audit-Trails und Zugriffskontrolle.
CIS Benchmarks	Standardisierte Härtungsleitfäden für Betriebssysteme und Serverdienste.
NIST Cybersecurity Framework	Leitfaden für Risikomanagement, Bedrohungserkennung und Vorfallsreaktion.
ISO 27001	Internationale Norm für Informationssicherheits-Managementsysteme (ISMS).

Quelle: https://admindocs.de/de/devops/wazuh-open-source-siem-und-xdr-fuer-devops-und-security-teams

Fazit

Die Implementation von Wazuh als umfassende Sicherheitsplattform erfordert tiefgreifendes technisches Verständnis und sorgfältige Planung. Wazuh bietet leistungsfähige Tools für Threat Detection, Compliance Monitoring und Incident Response.

Mit der zunehmenden Komplexität von IT-Infrastrukturen und der steigenden Zahl von Cyber-Bedrohungen wird die Bedeutung integrierter Sicherheitslösungen wie Wazuh weiter zunehmen. Zukünftige Entwicklungen könnten verstärkt auf KI-basierte Analysen, Cloud-native Sicherheit und automatisierte Response-Mechanismen setzen. Administratoren und Security-Teams sollten die kontinuierliche Weiterbildung und Anpassung ihrer Sicherheitsstrategien als essentiellen Bestandteil ihres Aufgabenbereichs betrachten.

Zur Übersicht: Alle DevOps Artikel & Guides

Quelle: https://admindocs.de/de/devops/wazuh-open-source-siem-und-xdr-fuer-devops-und-security-teams