Sentrilite: Hybrid-Cloud Observability, Security and Intelligence in One Platform

Sentrilite — Cloud-Native, AI-Powered, eBPF-based Lightweight, Real-Time System Observability & Runtime Security

Overview

Sentrilite is a Hybrid-Cloud Programmable **Observability & Runtime Security** Platform and streams structured, real-time events to a web UI where custom rules drive risk scoring, alerting, and reporting. Hybrid & multi-cloud ready: Works the same across public clouds and on-prem—EKS, GKE, AKS, vanilla Kubernetes, bare-metal Linux Servers, and edge—so you get a consistent, low-overhead security and observability layer for entire infrastructure all managed from a single dashboard. In Kubernetes, Sentrilite runs as a **privileged DaemonSet** on every node (no app changes required). Each agent observes container processes and **enriches events with Kubernetes metadata** (namespace, pod, container, UID/PID) by correlating cgroups with the API server.

Problem: Runtime blind spots in Kubernetes & hybrid cloud

Modern attacks increasingly unfold at runtime—after code is deployed—via process abuse, file access, container breakout attempts, and east-west network activity. Industry reports highlight rapid weaponization in cloud-native environments, where adversaries pivot quickly and live off the land inside containers and nodes. Sysdig+1

Beyond container misconfigurations, defenders need visibility into what actually executes: processes (`execve`), file reads/writes of sensitive material, and socket connects/binds. This is reflected in both community guidance and adversary models (MITRE ATT&CK for Containers), which enumerate runtime techniques like credential dumping, cryptomining, and command-and-control over common ports. MITRE ATT&CK+1

- CNCF Cloud Native Survey (Linux Foundation Research) — broad adoption of containers/Kubernetes continues; orgs still report gaps around security/observability and skills—driving interest in standardized telemetry. MITRE ATT&CK

- Red Hat “State of Kubernetes Security” (2023/2024 series) — misconfigurations and human error are frequent root causes of K8s incidents; many orgs experienced security events tied to configuration and supply-chain issues. MITRE ATT&CK

- Sysdig 2024 Global Cloud Threat Report — runtime threats (cryptomining, misuse of credentials), excessive permissions, and noisy images are common; emphasizes need for real-time detection and least-privilege in Kubernetes. MITRE ATT&CK

- OWASP Kubernetes Top Ten — catalogs the most impactful K8s risks (e.g., insecure defaults, supply chain, inadequate logging/monitoring), reinforcing that visibility and policy are core to defense. ARMO

- CNCF Cloud Native Security Whitepaper — end-to-end guidance across build/deploy/run; calls for defense-in-depth with runtime detection, least privilege, and strong observability of workload behavior. MITRE ATT&CK

- AWS EKS Best Practices Guide — recommends enabling audit/metrics/trace pipelines, using IRSA/least-privilege IAM, and enforcing NetworkPolicies—i.e., observability + policy are required for secure EKS. MITRE ATT&CK

- Google (GKE) security best practices — emphasizes hardening nodes, enforcing identity/authorization, and collecting/acting on audit logs and workload telemetry at scale. MITRE ATT&CK

- Grafana OpenTelemetry report — sustained growth in OpenTelemetry interest/adoption; signals industry momentum toward standard, vendor-neutral telemetry for better visibility and cost control. Grafana Labs

Why runtime telemetry (eBPF-class) matters

Authoritative guidance emphasizes **workload-level monitoring** alongside hardening. NIST’s container security guide and the NSA/CISA Kubernetes Hardening Guide both call out the need to monitor process and network activity to detect abuse that slips past pre-deploy controls. NIST Computer Security Resource Center+1

In practice, open-source Falco popularized rules that trigger on syscall patterns (e.g., reading /etc/shadow, spawning shells in containers, or unexpected network opens). This model—observe syscalls and emit policy-driven alerts—is now a de-facto pattern for runtime defense. Falco

What to capture at runtime (signals that matter)

Processes: command + args (execve), user/UID, PID/PPID, executable/comm—and the Kubernetes context (namespace/pod/container/UID) for attribution. These align with ATT&CK (e.g., discovery, credential access, defense evasion) and Falco-style detections. MITRE ATT&CK+1

Files: reads of secrets/keys/tokens (e.g., service account tokens, SSH keys), config files, kube kubeconfigs, and credential stores. NIST and NSA/CISA note credential theft and secret exposure are common runtime objectives. NIST Computer Security Resource Center+1

Network: opens/connects/binds, destinations (IP/port/CIDR), and anomalous egress patterns that indicate exfiltration or C2. Threat research shows active campaigns (e.g., Kinsing) weaponize runtime access rapidly, often “living off the land” inside compromised containers. Forbes+1

Make it observable: consistent context and timelines

For usable forensics and SRE workflows, runtime events should be **enriched** with k8s identifiers (namespace, pod, container) and host/process metadata. This mirrors OpenTelemetry’s semantic conventions (k8s.*, process.*, host.*) so events can correlate naturally with logs/metrics/traces and existing APM/SIEM pipelines.

What “good” looks like in a runtime platform

1. **Live stream & filtering:** per node/pod views with filters (namespace/pod/container), plus node health signals (e.g., OOMKilled) to speed root cause. Guidance from NSA/CISA and NIST stresses consolidating runtime signals for timely detection. CISA+1

2. **Rules & risk scoring:** declarative rules that tag and score events; high-risk findings become human-readable alerts (who/what/where in k8s). This follows Falco’s proven approach to runtime policies. Falco

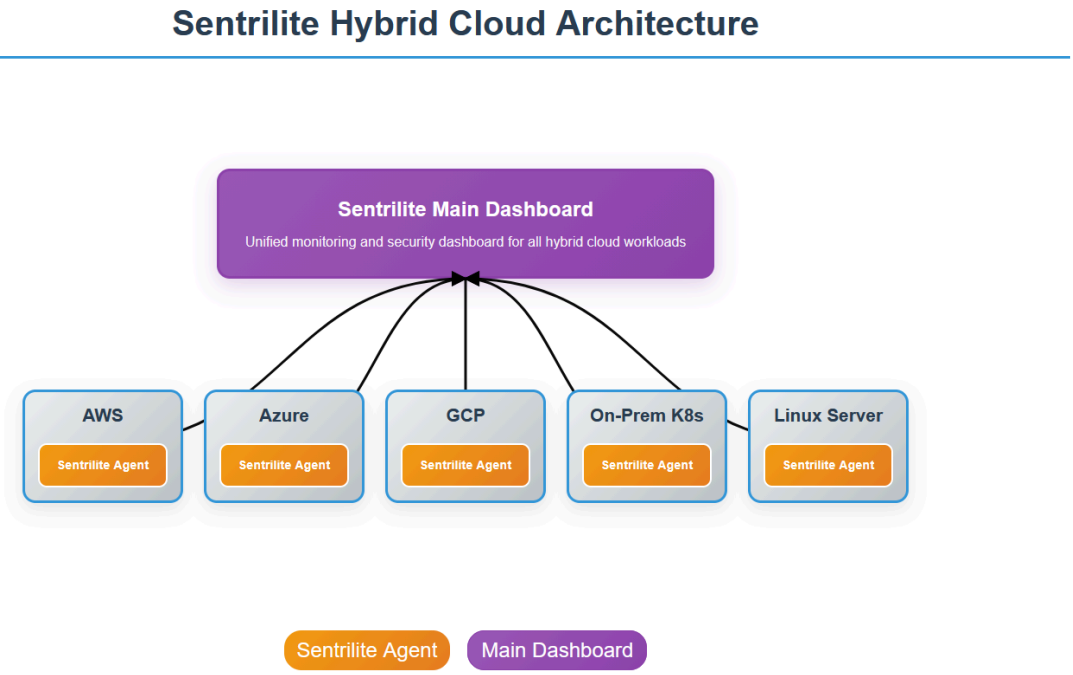
3. **Actionability:** PDF/CSV summaries and SIEM hand-off; align alerts to ATT&CK techniques to aid triage. MITRE ATT&CK

4. **Integrations:** ship to Alertmanager/PagerDuty for on-call; keep open standards for portability (Otel context, webhooks).

What Problems does Sentrilite solve ?

Category	Capability (high level)
Process Visibility	Capture every command real-time (execve) with user, PID/PPID, and container context. Example: trigger rules like "alert on cat /etc/passwd" or block high-risk binaries.
File Activity Monitoring	Detect access to sensitive files (keys, configs, tokens) and flag exfiltration or privilege-escalation attempts.
Network Activity Tracing	Observe socket connects/binds in real-time. Create CIDR-based rules such as "deny egress to 1.2.3.0/24:443".
Live Operations UI	Stream real-time events by node, namespace, or pod. Visualize node health, OOMKilled events, and container restarts.
Custom Rules & Risk Scoring	Declarative JSON rule engine tags, classifies, and scores events. High-risk detections automatically trigger alerts.
Audit & Reporting	Generate timeline reports (PDF/CSV) for compliance, audits, and incident reviews.
Rapid Response Controls	Optional iptables-based allow/deny rules for quick mitigation of suspicious network behavior.
Integrations	Supports external alerting with **Prometheus Alertmanager** and **PagerDuty** out of the box.
AI Insights	Embedded LLM engine summarizes anomalies, trends, and remediation recommendations directly from telemetry streams.

✨ Sentrilite Hybrid Cloud Architecture



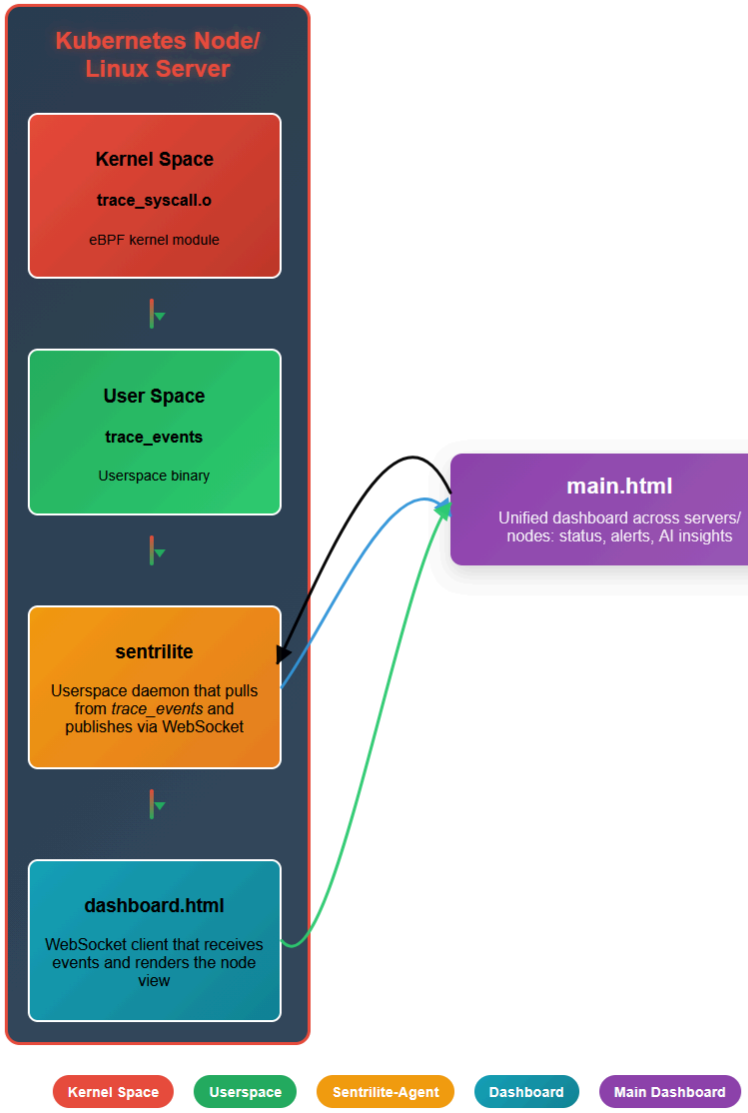
Hybrid Cloud Architecture Description

- **Public Cloud:** AWS, Azure, and GCP, On-Prem K8s, Linux Servers (bare-metal) — agents deployed per VM or node.

- **On-Prem:** On-premises infrastructure with Sentrilite agent Bare-metal or VMs with identical capabilities.

- **Linux Servers:** Individual Linux servers with Sentrilite agents.
- **Sentrilite Main Dashboard:** Centralized monitoring and security dashboard that aggregates data from all agents.
- **Data Flow:** All Sentrilite agents stream telemetry data to the main dashboard for unified observability.

Sentrilite Components



Workflow Description:

- **trace_syscall.o** runs in kernel space and captures system calls and events directly from the kernel
- **trace_events** is a userspace application that communicates with the eBPF program to retrieve captured events
- **sentrilite** daemon processes the events from trace_events and makes them available via WebSocket connections
- **dashboard.html** provides a real-time view of events for individual nodes/servers

- **main.html** provides a unified view across multiple servers/nodes with AI insights and alert management

Data Flow:

- System calls and events are captured at the kernel level by the eBPF module.
- Events are processed and enriched with metadata (PID, command, arguments, etc.)
- Processed events are streamed to WebSocket clients in real-time.
- Multiple dashboard interfaces can connect to monitor different aspects of the system.
- AI insights and alerting are generated based on the streamed events.

✨ Key Features

- Multi-Cloud/On-Prem visibility and management from a single dashboard
- eBPF syscall & network visibility
- Real-time dashboards (Nginx + WebSocket server)
- Custom rules with risk scoring and alerting
- Kubernetes enrichment (namespace/pod/container/UID) when running as a DaemonSet
- OOMKilled alerts and pod watchers (best effort if K8s APIs available)
- Seamlessly integrate with prometheus alert-manager/pagerduty

⚙️ System Requirements

Minimum Requirements

- **Linux Kernel with eBPF support** (Linux 5.8+ recommended)
- **Root privileges** (for loading eBPF programs)
- **Ports:** 80 (dashboard), 8765 (WebSocket)
- **Kubernetes** (optional): Cluster access with ability to run a privileged DaemonSet

General Requirements

- **bpftool:** Load eBPF programs and manage maps
  sudo apt install bpftool # Ubuntu

- **libbpf & headers:** Required by the kernel loader (trace_events)
  - Pre-installed on most modern distros (use bundled binary)
- **nginx:** Required to view dashboard
  sudo apt install nginx

🛠️ Third-Party Integrations (PagerDuty & Alertmanager)

Kubernetes Configuration

Add URLs in a ConfigMap (e.g., `ALERTMANAGER_URL`, optional `PAGERDUTY_EVENTS_URL`) and the PagerDuty routing key in a Secret (`PAGERDUTY_ROUTING_KEY`).

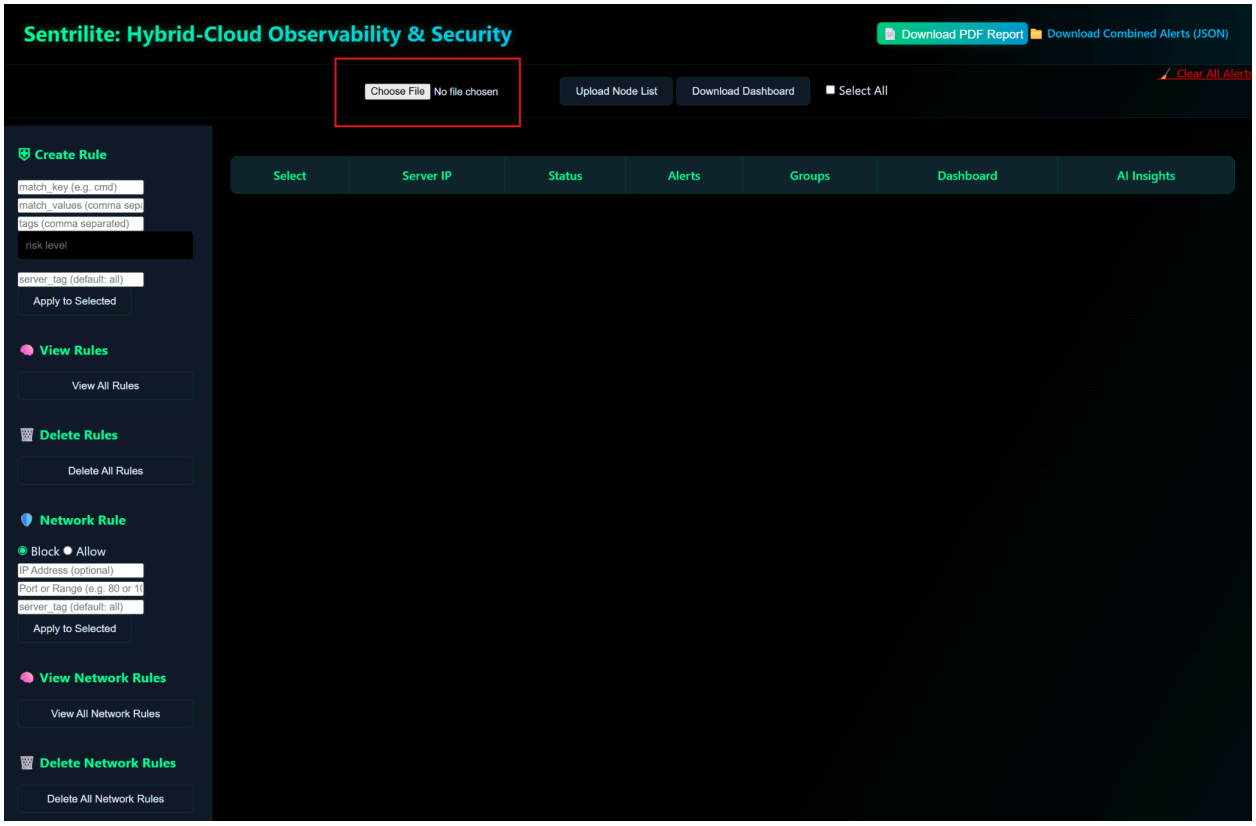
Standalone Linux Configuration

Set the same keys in `sys.conf` (e.g., `ALERTMANAGER_URL=http://...`, `PAGERDUTY_ROUTING_KEY=...`; PD URL defaults to US if omitted).

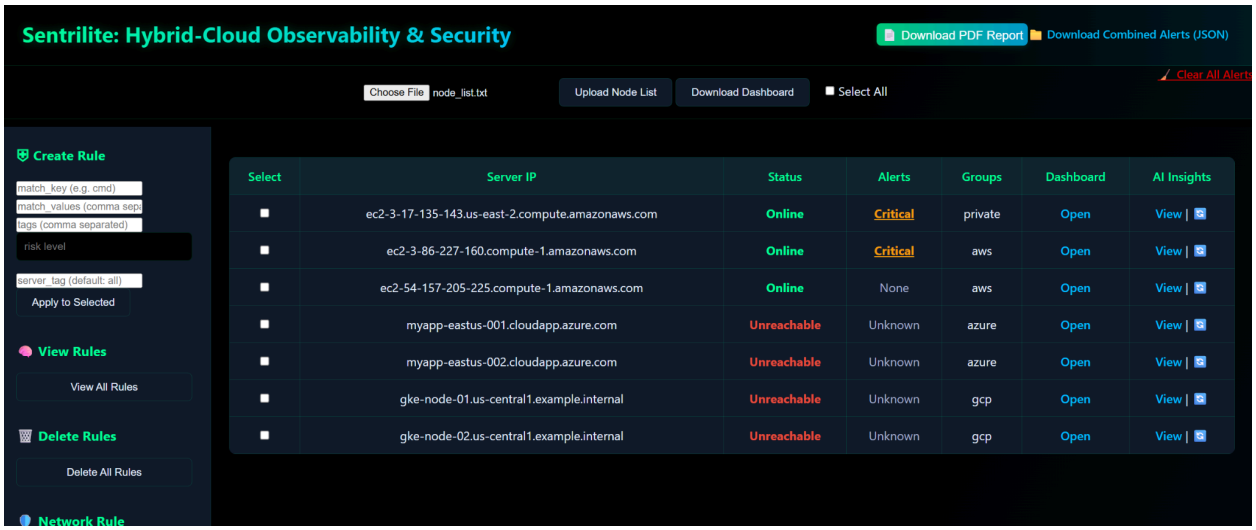
Note: Sentrilite prefers env vars (K8s) and falls back to `sys.conf` (bare metal). Alertmanager must be reachable and supports v2 API (`/api/v2/alerts`). PagerDuty uses Events v2.

✨ Sentrilite: Getting Started

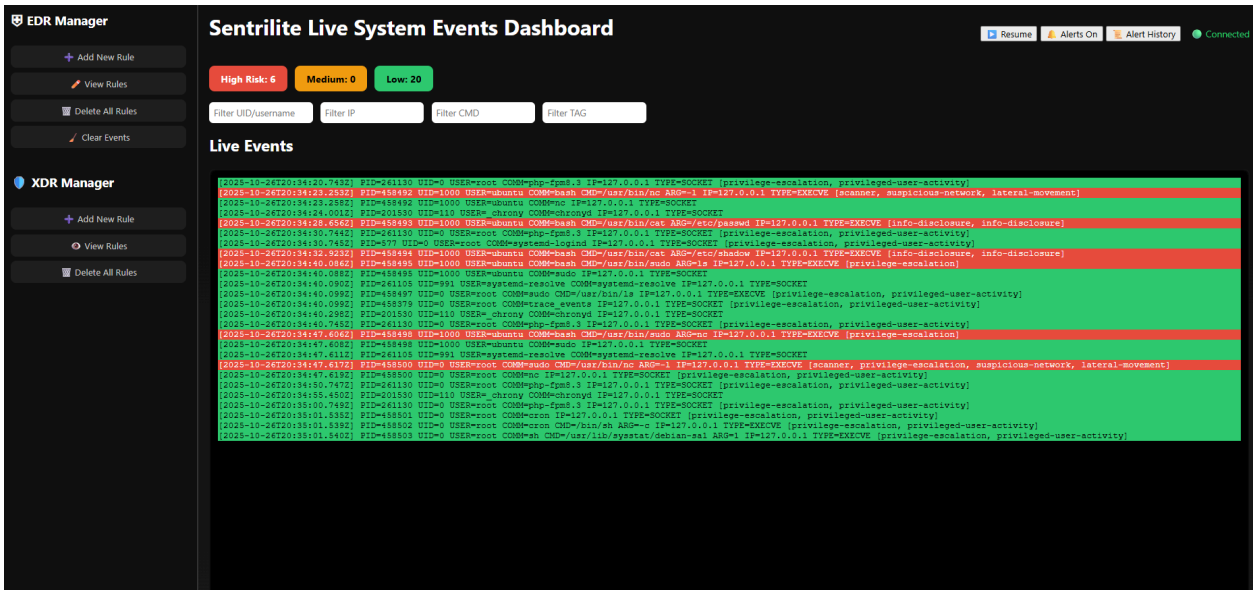
On the Main Dashboard, choose file created from step 3 above and click upload Node List:



Upon Clicking Upload, the nodes are added as follows:

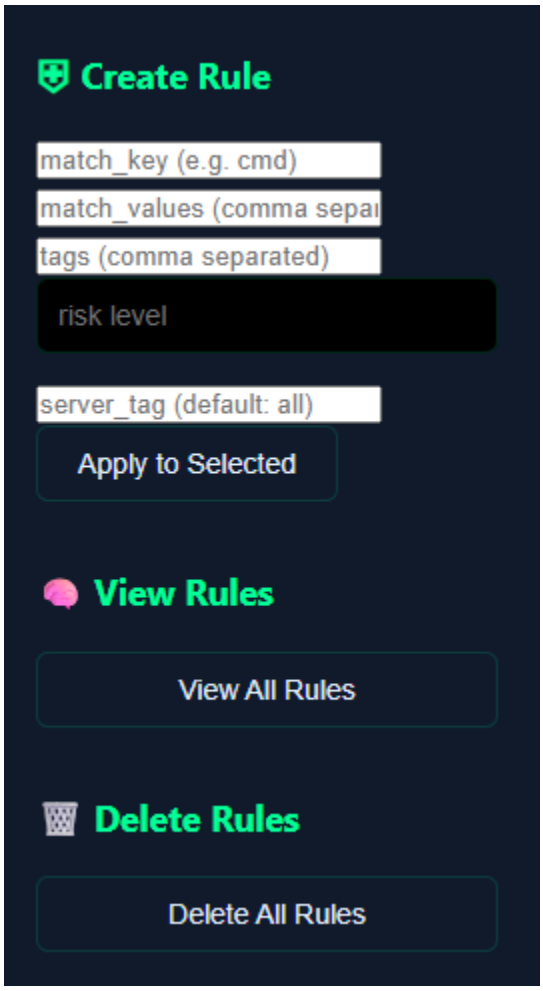


Upon clicking the ‘Open’ Link under the Dashboard of any Node, it will point to the Live dashboard of that Node like the following screenshot:



Back on the Main dashboard, you can create custom Alert Rules:

On the side panel, we have the rule manager:



The keys and their corresponding values can be one of the following:

1. cmd => command or process like ls/sudo/nc
2. arg => The first argument of any command ls “`abc.txt`” . So any command using abc.txt will raise an alert.
3. ip => ipv4 address
4. pid => process id
5. iid => user id (for example uid = 0 for root user)
6. user => user name (for example: root, ubuntu)

Sentrilite comes with some pre-defined rules. Users can create additional custom rules on top of this. The default rules are saved in `rules.json` on the server/node:

```
[
  { "match_key": "cmd", "match_values": ["sudo","su","pkexec"],
"tags": ["privilege-escalation"], "risk_level": 1 },
  { "match_key": "cmd", "match_values":
["nc","ncat","netcat","socat"], "tags":
["suspicious-network","lateral-movement"], "risk_level": 1 },
  { "match_key": "cmd", "match_values": ["nmap","masscan"], "tags":
["recon","port-scan"], "risk_level": 1 },
  { "match_key": "cmd", "match_values":
["iptables","ip6tables","nft"], "tags": ["network-policy-change"],
"risk_level": 1 },
  { "match_key": "cmd", "match_values": ["insmod","modprobe"],
"tags": ["kernel-module-load"], "risk_level": 1 },
  { "match_key": "arg", "match_values":
["169.254.169.254","http://169.254.169.254","https://169.254.169.254"
], "tags": ["cloud-metadata-access"], "risk_level": 1 },
  { "match_key": "arg", "match_values": ["metadata.google.internal"],
"tags": ["cloud-metadata-access"], "risk_level": 1 },
  { "match_key": "cmd", "match_values":
["xmrig","minerd","ethminer","lolMiner","teamredminer"], "tags":
["crypto-mining"], "risk_level": 1 },

  { "match_key": "cmd", "match_values": ["ssh","scp","sftp"], "tags":
["lateral-movement"], "risk_level": 2 },
  { "match_key": "cmd", "match_values": ["curl","wget"], "tags":
["ingress-egress","exfiltration"], "risk_level": 2 },
  { "match_key": "cmd", "match_values": ["tcpdump","tshark"], "tags":
["packet-capture"], "risk_level": 2 },
  { "match_key": "cmd", "match_values": ["mount","umount"], "tags":
["filesystem-change"], "risk_level": 2 },
  { "match_key": "cmd", "match_values":
["kubectl","ctr","crictl","docker","runc"], "tags":
["container-runtime-admin"], "risk_level": 2 },
  { "match_key": "cmd", "match_values": ["tc"], "tags":
["traffic-shaping"], "risk_level": 2 },
  { "match_key": "cmd", "match_values": ["dd"], "tags":
["bulk-io","potential-exfil"], "risk_level": 2 },

  { "match_key": "cmd", "match_values":
["python","perl","ruby","node"], "tags": ["scripting-runtime"],
"risk_level": 3 },
  { "match_key": "cmd", "match_values": ["base64","xxd","tar","zip"],
"tags": ["encode-archive"], "risk_level": 3 },
```

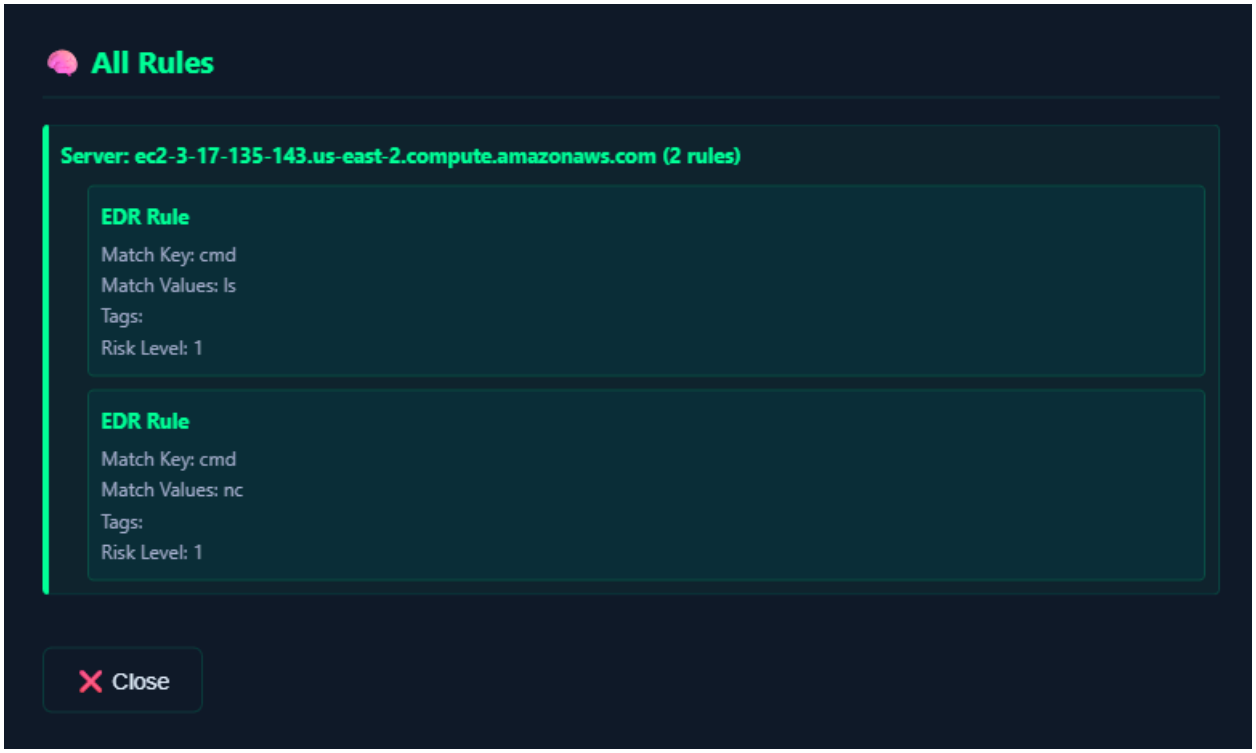
```
  { "match_key": "cmd", "match_values":
["apt","apt-get","yum","dnf","zypper","apk"], "tags":
["package-install"], "risk_level": 3 },
  { "match_key": "user", "match_values": ["root"], "tags":
["privileged-user-activity"], "risk_level": 3 },
  { "match_key": "iid",  "match_values": ["0"], "tags":
["privileged-user-activity"], "risk_level": 3 }
]
```

Sentrilite monitors some system level sensitive files by default. The list of default files are as follows and stored on the node as sensitive_files.json

```
{
  "files": [
    "/etc/shadow",
    "/etc/sudoers",
    "/etc/sudoers.d/*",
    "/root/.ssh/*",
    "/home/*/.ssh/*",
    "/etc/ssh/ssh_host_*",
    "/etc/ssl/private/*",
    "/etc/pki/tls/private/*",
    "/etc/letsencrypt/live/*/*",
    "/etc/letsencrypt/archive/*/*",
    "/var/run/docker.sock",
    "/run/containerd/containerd.sock",
    "/var/run/secrets/kubernetes.io/serviceaccount/token",
    "/etc/kubernetes/admin.conf",
    "/etc/kubernetes/kubelet.conf",
    "/etc/kubernetes/pki/*",
    "/var/lib/kubelet/pki/*",
    "/var/lib/kubelet/kubeconfig",
    "/home/*/.aws/credentials",
    "/home/*/.aws/config",
    "/home/*/.config/gcloud/*",
    "/home/*/.azure/*",
    "/home/*/.docker/config.json",
    "/home/*/.git-credentials",
    "/home/*/.netrc",
    "/etc/krb5.keytab",
    "/home/*/.gnupg/private-keys-v1.d/*",
    "/etc/openvpn/*.key",
    "/etc/wireguard/privatekey",
    "/etc/ipsec.secrets",
    "/etc/mysql/*.cnf",
```

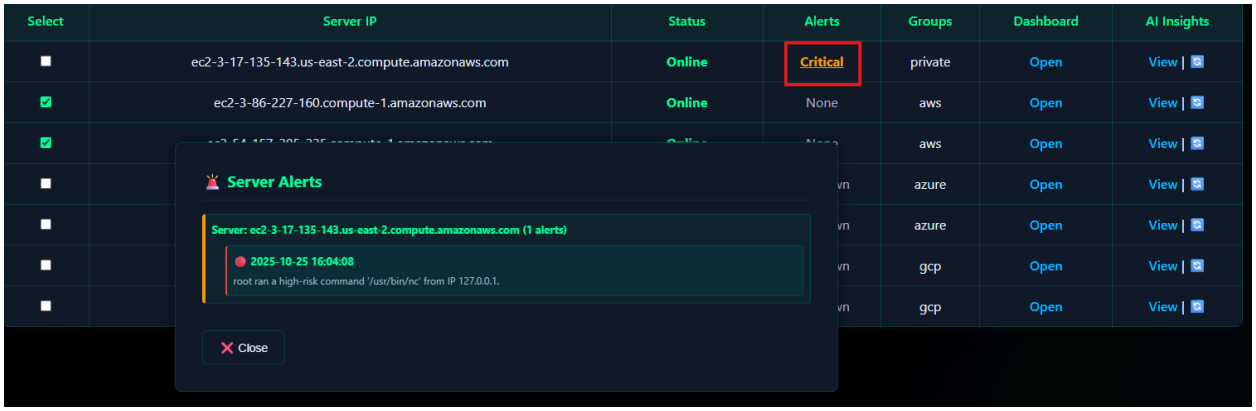
```
      "/root/.my.cnf",
      "/home/*/.pgpass",
      "/etc/mongod.conf",
      "/etc/redis/redis.conf",
      "/etc/nginx/*.conf",
      "/etc/nginx/sites-*/**",
      "/etc/httpd/conf.d/*.conf",
      "/var/log/auth.log",
      "/var/log/secure",
      "/var/log/messages",
      "/var/log/syslog",
      "/var/log/journal/**",
      "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt",
      "/var/run/secrets/kubernetes.io/serviceaccount/namespace",
      "/etc/passwd"
    ]
}
```

Click “View Rules” to see existing rules:



Click Delete Rules to remove all the rules.

Once a rule is satisfied, an alert is triggered and the status of that server/node changes to Critical . Click the critical link to see the current alerts on any node like this:



The same alert is recorded on the server side in the json format with all the metadata in the file: alerts.json. It can be downloaded from the main dashboard):

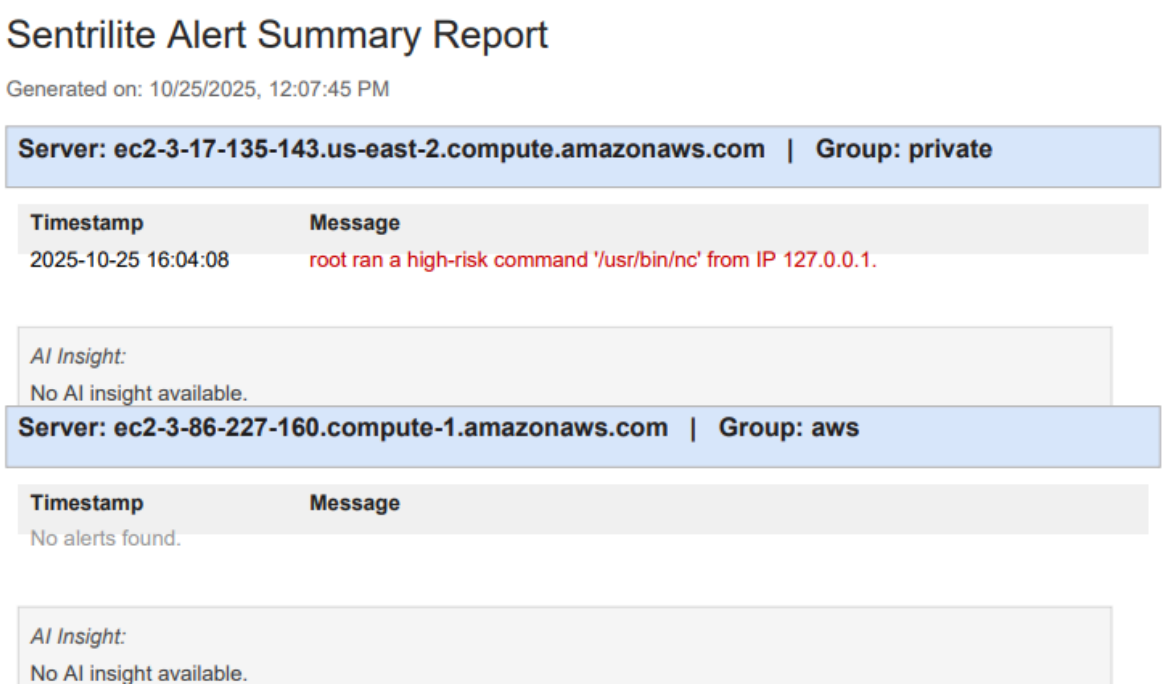
Example Alert Message:


```
[
  {
    "time": "2025-10-25 16:04:08",
    "type": "high_risk",
    "message": "root ran a high-risk command '/usr/bin/nc' from IP
127.0.0.1.",
    "pid": "442651",
    "cmd": "/usr/bin/nc",
    "args": "-l",
    "ip": "127.0.0.1",
    "risk_level": 1,
    "tags": [
      "scanner",
      "privilege-escalation"
    ]
  }
]
```


These alerts will also be routed to pagerduty and alertmanager if properly configured in the sys.conf (linux) or configmap/secrets (kubernetes) covered later in the installation steps.

Click the download pdf report button to generate the reports with all alerts on all the servers: (Note: For LLM insights, you need to install a local LLM server)

Sentrilite One-Click PDF Report



Sentrilite Alert Summary Report

Generated on: 10/25/2025, 12:07:45 PM

Server: ec2-3-17-135-143.us-east-2.compute.amazonaws.com | Group: private

Timestamp	Message
2025-10-25 16:04:08	root ran a high-risk command '/usr/bin/nc' from IP 127.0.0.1.

AI Insight:
No AI insight available.

Server: ec2-3-86-227-160.compute-1.amazonaws.com | Group: aws

Timestamp	Message
No alerts found.	

AI Insight:
No AI insight available.

🛠️ Installation Steps

For Kubernetes Cluster: EKS/AKS/GKE or Private Kubernetes Cluster

1. **Deploy Sentrilite DaemonSet:**

   Helm Installation: In the charts directory:

   ```
   helm upgrade --install sentrilite charts/sentrilite -n
   kube-system --create-namespace
   ```

   OR plain kubectl installation:

   ```
   kubectl apply -f sentrilite.yaml

   kubectl -n kube-system get pods -l app=sentrilite-agent -o wide

   kubectl get nodes | awk '!/NAME/{print $1,",K8s"}' > nodes.txt
   ```

   # Port forward

```
POD=$(kubectl -n kube-system get pod -l app=sentrilite-agent -o
name | head -n1)

kubectl -n kube-system port-forward "$POD" 8080:80 8765:8765
```

2. **Verify deployment:**
   ```
   kubectl -n kube-system get pods -l app=sentrilite-agent -o
   wide
   ```

3. **Create nodes list:**
   ```
   kubectl get nodes | awk '!/NAME/{print $1,",K8s"}' > nodes.txt
   ```

   The file format should like this: Node_ip,group:

   ```
   ec2-3-17-135-112.us-east-2.compute.amazonaws.com,private

   ec2-3-86-227-155.compute-1.amazonaws.com,aws

   ec2-54-157-205-222.compute-1.amazonaws.com,aws

   myapp-eastus-001.cloudapp.azure.com,azure,azure

   myapp-eastus-002.cloudapp.azure.com,azure,azure

   gke-node-01.us-central1.example.internal,gcp

   gke-node-02.us-central1.example.internal,gcp
   ```

For Non-Kubernetes Linux based Cluster

Unzip the bundle:

unzip sentrilite_agent_bundle.zip

1. cd sentrilite

2. **Load the bpf program:**
   ```
   sudo ./install.sh
   ```

   Configure sys.conf:
   ```
   IFACE=
   LICENSE_KEY=./license.key
   PAGERDUTY_EVENTS_URL=""
   PAGERDUTY_ROUTING_KEY=""
   ALERTMANAGER_URL=""
   ```

3. **Launch the Server:**
   ```
   sudo ./sentrilite
   ```

4. **Open the Dashboard:**
   - Copy the `dashboard.html` to `/var/www/html` or web root directory
   - Open `dashboard.html` in your browser: `http://<YOUR-SERVER-IP>/dashboard.html`
   - You should see live events appear in real-time

5. **Log format in the Live Dashboard Web UI:**

   ```
   [2025-04-14T00:12:32.008Z] PID=1234 COMM=ssh CMD=/bin/bash ARG= IP=127.0.0.1 TYPE=EXECVE
   ```

6. **Open the Main Dashboard on your control plane:**
   - Copy the `main.html` & `jspdf.umd.min.js` to `/var/www/html` on your main admin server
   - Open the `main.html` in your browser: `http://<YOUR-SERVER-IP>/main.html`
   - Click choose file and select a file containing your server lists

For more detailed information, refer to `dashboard.README`

Configuration

- **`license.key`** — place in the current directory (baked in image or mounted as Secret)
- **`sys.conf`** — network config, placed in the current directory (baked in image or mounted as ConfigMap)
- **Rule files** (`rules.json`, `sensitive_files.json`, `xdr_rules.json`, `alerts.json`) reside in the working dir; rules can be managed via the dashboard

Alerts & K8s Enrichment

- Events include (when available): `k8s_namespace`, `k8s_pod`, `k8s_container`, `k8s_pod_uid`
- OOMKilled alerts and pod watchers run best-effort when the agent can access K8s APIs

🛠️ Un-installation Steps

For Kubernetes Cluster: EKS/AKS/GKE or Private Kubernetes Cluster

kubectl -n kube-system delete ds/sentrilite-agent
(Optional) If pods hang in Terminating:
```
kubectl -n kube-system delete pod -l app=sentrilite-agent --force
--grace-period=0
```

For Non-Kubernetes Linux based Cluster

Run the following commands as root:

```
sudo ./unload_bpf.sh
```

🔐 Licensing

The project is currently using a trial `license.key`

Support

For licensing, troubleshooting, or feature requests:

- 📧 **Email:** info@sentrilite.com
- 🌐 **Website:** https://sentrilite.com