SOC Architect - Technical Challenge

Part One - Build a Mini SOC

Goal

Design, implement and document a CI/CD pipeline that deploys Wazuh as a SIEM onto a Docker Swarm cluster. The pipeline must:

- Build container images
- Scan them with Trivy
- Run tests (Selenium)
- Deploy via Ansible to Docker Swarm
- Handle secrets securely and manage TLS certificates
- Be reproducible on GitHub Actions self-hosted runners

Deliverables are evaluated on correctness, security, reliability, and clarity of documentation.

Scope & Functional Requirements

1. Wazuh Stack
    a. Deploy a functional Wazuh stack (at minimum: Wazuh indexer, manager, dashboard),
    b. Expose dashboard via HTTPS,
    c. Persist data in Docker volumes,
    d. Optional: multi-node (indexer/manager) topology for bonus points.
2. CI/CD (GitHub Actions + Self-Hosted Runners)
    a. On PR and push to main:
        i. Build container images (custom or extended upstream),
        ii. Scan images with Trivy (pipeline must fail on Critical/High vulnerabilities),
        iii. Run Selenium checks against a temporary environment or existing test env (see “Testing”),

        iv. Deploy to Docker Swarm via Ansible (only on main and only after all checks pass).
    b. Runners: assume at least one self-hosted Linux runner with Docker + Ansible + Python + Trivy + Chrome/Chromedriver (or Playwright+Selenium bindings) installed. Document any extra runner prerequisites.
3. Testing
    a. Implement Selenium tests that:
        i. Validate the Wazuh dashboard is reachable over HTTPS,
        ii. Validate the page title and login form elements are present,
        iii. Optional: programmatic login using a non-admin test account (credentials via secrets), assert landing page element(s).
    b. Include a simple API health probe (e.g., manager API endpoint returns 200/JSON schema).
4. Deployment (Ansible → Docker Swarm)
    a. Use Ansible to:
        i. Ensure Docker Swarm is initialized/joined (idempotent),
        ii. Create necessary overlay networks and secrets/configs,
        iii. Deploy or update the stack via docker_stack or docker stack deploy,
        iv. Roll back on failed update (graceful failure preferred).
    b. Parameterize environment via Ansible group_vars/inventory.
5. Security & Secrets
    a. DO NOT hardcode secrets in Git or in plain YAML.
    b. Use at least one of:
        i. GitHub Actions Encrypted Secrets + Swarm secrets,
        ii. Ansible Vault for inventory/group_vars,
        iii. HashiCorp Vault or SOPS (bonus).
    c. Secrets to protect include: Wazuh credentials, API keys, TLS private keys.
    d. Explain your secret rotation approach and least-privilege choices.
6. Certificate Management
    a. Terminate HTTPS with:
        i. A reverse proxy (e.g., Nginx/Traefik) + Let's Encrypt/ACME or,
        ii. Internal CA/self-signed (document trust/setup).
    b. Automate certificate provisioning/renewal where possible.
7. Quality Gates & Policies
    a. Trivy: fail on Critical/High (configurable via environment variable or policy file),
    b. Include linting for Ansible (ansible-lint) and YAML (yamllint) (bonus),
    c. Require PR checks to pass before merging.

What You Submit

1. Repository containing:
    a. README.md with:
        i. Architecture overview + diagram,
        ii. Prerequisites (runner setup, required tools),
        iii. How to run locally and via CI,
        iv. How secrets/TLS are managed,
        v. Rollback/recovery steps.
    b. GitHub Actions workflow(s) under .github/workflows/*.yml.
    c. Ansible:
        i. inventories/ (sample inventory with placeholders),
        ii. group_vars/ (do not commit real secrets—show vault examples),
        iii. playbooks/deploy.yml, playbooks/teardown.yml,
        iv. Roles or tasks for Swarm init, networks, secrets, stack deploy.
    d. Docker/Compose/Stack:
        i. docker/ (Dockerfiles if you extend images),
        ii. stack/wazuh-stack.yml (Swarm stack spec),
        iii. configs/ and secrets/ templates.
    e. Security:
        i. security/ with TLS automation scripts/config (Traefik/Nginx), policy docs.
    f. Tests:
        i. tests/selenium/ with runnable tests + instructions,
        ii. Optional tests/api/ health checks.
    g. Trivy configuration:
        i. trivy/ policy or config file,
        ii. Example make scan or CI job step.
2. Evidence
    a. Screenshots of pipeline runs (or CI links),
    b. Screenshot of Wazuh dashboard (mask credentials),
    c. Sample Trivy report artifact,
    d. Sample Ansible output (successful deploy).
3. Assumptions Document
    a. Clearly state any assumptions (DNS, domain/TLS, runner permissions, IPs).

Part Two – Create threat incident scenario

During SOC operations, a suspicious activity pattern has been identified:

- On certain Linux hosts, multiple failed SSH login attempts from the same IP are followed by a successful login within a short time window.
- The successful login uses a previously unseen user account on that host.

Goal

Implement a custom Wazuh rule (and decoder if necessary) that:

- Detects this pattern,
- Generates a custom alert (rule ID in the 100000+ range),
- Includes relevant metadata (source IP, username, time window) in the alert.

Requirements

1. Log Source
    a. Use sample auth.log or secure syslog entries (you can simulate them or inject them into Wazuh using logger).
    b. Example pattern to detect:

Jan 14 12:30:12 server sshd[1830]: Failed password for invalid user test1 from 203.0.113.5 port 50234 ssh2
Jan 14 12:30:14 server sshd[1830]: Failed password for invalid user test1 from 203.0.113.5 port 50234 ssh2
Jan 14 12:30:20 server sshd[1830]: Accepted password for backupuser from 203.0.113.5 port 50234 ssh2

2. Detection Logic
    a. Match N failed logins (configurable, e.g., 3+) from the same source IP within 60 seconds, followed by a successful login from that same IP.
    b. Ensure the successful login is from a username not seen before on that host in the past 24 hours (can be simulated with a lookup or by maintaining a temporary baseline list).

    c. Rule must be stateful using Wazuh's if_matched_sid or same_source_ip concepts.
3. Implementation
    a. Place custom rules in /var/ossec/etc/rules/local_rules.xml.
    b. Create a custom decoder if default SSH decoder does not extract the fields you need (e.g., srcip, user).
    c. Assign rule ID ≥ 100000.
    d. Add a meaningful description and mitre.id mapping (e.g., MITRE ATT&CK T1110 – Brute Force).
4. Testing
    a. Demonstrate triggering the alert using injected log events (via logger or by appending to a test log file).
    b. Capture a screenshot of the alert in the Wazuh dashboard and include the raw JSON event.
    c. Show the rule XML in the repo.
5. Bonus
    a. Enrich the alert with geo-IP lookup of the source IP.
    b. Send an alert to a Slack/Teams channel via Wazuh integration.
    c. Add a SOAR playbook step (if your CI/CD deployed one) to auto-block the source IP in a firewall.

High Availability Architecture (Optional)

Design the platform to tolerate failures without losing data or core functionality.

Minimum HA expectations:

- Docker Swarm
    - 3+ manager nodes (quorum), 1–N workers.
    - Swarm overlay networks for front-end, backend, and management.
    - All services define healthchecks, restart policies, and rolling update params.
- Wazuh Components
    - Indexer cluster (OpenSearch/Wazuh Indexer): ≥3 data nodes (replication enabled) + dedicated master eligible nodes (can be the same for small labs if justified). Snapshots enabled.
    - Wazuh Manager: clustered deployment (master/worker), with shared registration state and persistent storage for manager data.

    - Wazuh Dashboard: ≥2 replicas behind a load balancer/reverse proxy (Traefik/Nginx).
- State & Storage
    - Persistent volumes for indexer data and manager data (document storage backend—e.g., NFS/Portworx/Longhorn/CEPH).
    - Clearly document storage performance and replication characteristics and how they impact RPO/RTO.
- Ingress / TLS
    - HA reverse proxy with ACME (Let's Encrypt) or internal PKI, with automatic renewal and zero-downtime reload.
- Secrets
    - Centralized secret handling (Swarm secrets / Ansible Vault / HashiCorp Vault / SOPS). No secrets in Git or logs.

HA verification demo (must include video/gif or screenshots + logs)

- Take one Swarm manager down: show cluster stays healthy.
- Take one indexer node down during ingest: show no data loss (replicas) and dashboard stays searchable.
- Restart a dashboard replica: show uninterrupted access via load balancer.

Architecture Documentation (HLD & LLD) (Optional)

Provide two documents in /docs:

A) High-Level Design (HLD)

- Context & goals: assumptions, constraints, non-functional requirements (availability, RPO/RTO, capacity).
- Logical architecture: SIEM, SOAR (if used), CI/CD, ingress, secrets, monitoring.
- HA strategy: quorum, replicas, failover paths.
- Data flows: agent → manager/indexer, dashboard queries, CI/CD to Swarm via Ansible.
- Security posture: trust boundaries, TLS, secret storage, IAM/least privilege.
- Capacity & scaling: #agents, EPS targets, data retention, index sizing.
- Risks & mitigations.

B) Low-Level Design (LLD)

- Inventories: node roles, IPs, labels, ports.
- Service specs: exact stack YAML, env vars, resource limits, healthchecks, update configs.
- Storage mapping: volumes, mount points, backup/snapshot schedules.
- Network: overlay nets, subnets, LB listeners, firewall rules.
- CI/CD jobs: step-by-step (build → Trivy → Selenium → deploy), artifacts, gates.
- Secrets & certs: locations, rotation process, ACLs, renewals.
- Runbooks: bootstrap, day-2 ops (scale out/in), common failures.

Provide an architecture diagram in the HLD and a component/service diagram in the LLD.

Disaster Recovery (DR) Plan with Demo (Optional)

Deliver a DR plan and a reproducible demo.

Plan must include

- RPO/RTO targets and justification.
- Backups:
    - Wazuh Indexer/OpenSearch snapshots (repo type, schedule, retention).
    - Wazuh Manager critical state (keys/registrations/config).
    - Config/stack definitions (IaC).
- Restore procedures:
    - Clean-room restore to a fresh cluster (steps & validation checks).
    - Partial restore (single node or index) procedures.
- Verification: post-restore integrity checks (index status green, dashboards load, agent connectivity).
- Off-site/remote snapshot storage (if simulated, explain how to switch endpoints).

DR demo (must include video/gif or step screenshots)

1. Take fresh snapshots.
2. Simulate a site failure (stop/remove all services).
3. Recreate cluster via Ansible and restore from snapshots.
4. Show dashboard and sample historical alerts available again.

Update & Data Migration Planning (Optional)

Update approach

- Use rolling updates in Swarm with health gates and max parallelism.
- Version all images/configs; keep rollback path (docker service update --rollback).
- Maintain backward-compatible config changes where possible.

Data migration scenarios (cover at least two)

1. Indexer major upgrade requiring index migration:
    a. Snapshot before, rolling node replacement, post-migration reindex if needed, validation steps.
2. Wazuh Manager version bump:
    a. Staged rollout (non-prod → prod), schema or API changes validated via tests.
3. Storage backend change (e.g., NFS → CSI/Portworx/Longhorn):
    a. Controlled cutover: drain services, sync data, flip mounts, verify integrity.

Testing for updates

- Include a pre-update checklist, canary deployment, and post-update Selenium/API smoke tests.
- Keep maintenance windows and comms template.

Submission

- Share a GitHub repo URL with public read access (or invite https://github.com/toufik-airane).
- Include a short demo note (or 2–3 screenshots) showing a successful CI run and deployed dashboard.
- Deadline: see date in the email.

Final Note

Completing the entire challenge is not mandatory. We encourage candidates to submit as much as they can accomplish. Your submission should reflect both technical depth and clarity of communication. We are looking for practical, real-world solutions that demonstrate your ability to design, implement, and optimize a SOC environment while

considering operational efficiency, security best practices, and effective collaboration between people, processes, and technology.

Good luck and have fun.