**Data Engineer Perspective**

**Main Goal:** Build robust, scalable, and secure data pipelines to collect, store, and process threat intelligence data.

**Responsibilities**

1. **Data Acquisition & Ingestion**
   * Build connectors for threat intel feeds (e.g., MISP, AlienVault OTX, PhishTank, AbuseIPDB, VirusTotal).
   * Handle both **pull-based APIs** and **push-based webhook feeds**.
   * Implement **rate-limiting & retry logic** for unreliable APIs.
   * Types of threat data:
     1. **Indicators of Compromise (IOCs)** → IP addresses, domains, URLs, file hashes, email addresses, registry keys, etc.
     2. **Threat actor profiles** → names, aliases, motivations, attack patterns.
     3. **Vulnerability intelligence** → CVE, CWE, exploit information.
     4. **TTPs** (Tactics, Techniques, and Procedures) → mapped to MITRE ATT&CK.
   * Data formats: JSON, CSV, XML, STIX/TAXII, proprietary APIs.
   * **Where to get it?**
     1. **OSINT** (Free): AlienVault OTX, AbuseIPDB, FeodoTracker, PhishTank, URLHaus, MISP community feeds.
     2. **Paid/commercial**: Recorded Future, ThreatConnect, Anomali ThreatStream, Intel 471.
     3. **Industry-specific**: FS-ISAC (financial), Health-ISAC (healthcare).
     4. **Government/standards**: CISA feeds, NVD (National Vulnerability Database).
2. **Ingestion Layer**
   * Technologies: Python scripts, Apache NiFi, Kafka Connect, or Fluentd for streaming.
   * Why? They allow high-volume, fault-tolerant ingestion from multiple APIs.
3. **Data Normalization**
   * Technologies: Python (pandas, PySTIX2), Logstash, or custom ETL pipelines.
   * Why? To map multiple formats (CSV, JSON, STIX, TAXII) into a unified schema.
   * Tasks:
     1. Standardize diverse IOC formats into an **internal unified schema** (e.g., IPs, domains, hashes, CVEs).
     2. Use STIX 2.1 or TAXII for interoperability.
     3. Implement deduplication logic for repeated IOCs from multiple sources.
4. **Database Schema Design**
   * Task: Design a schema for storing:
     1. Raw IOCs
     2. Enriched IOC data
     3. Source metadata
     4. TLP classification
   * Technologies:
     1. **Short-term**: Elasticsearch / OpenSearch → fast search and indexing.
     2. **Long-term**: PostgreSQL or MongoDB for structured storage.
     3. Optional: Neo4j for graph-based threat actor relationship mapping.
5. **Data Quality & Validation**
   * Ensure IOCs are syntactically valid (e.g., regex for IPs/domains, hash length).
   * Implement data quality dashboards.
6. **Pipeline Monitoring & Logging**
   * Track ingestion errors, enrichment failures, and processing delays.
   * Integrate with alerting systems (Slack, PagerDuty, etc.).
7. Security Considerations
   * All feeds pulled via secure channels (HTTPS/TLS 1.3).
   * API keys encrypted (HashiCorp Vault or AWS Secrets Manager).
   * Rate-limiting & caching to prevent IP blocking.

**Expected Outcomes**

* Reliable ingestion pipelines with **99%+ uptime**.
* Standardized, clean IOC database ready for search, enrichment, and analysis.
* Reduced duplicates & improved data integrity.
* Scalable architecture for adding new threat feeds quickly.

**Core MVP Features for Phase 1**

✅ Multi-source Threat Feed Ingestion (OSINT + Commercial)  
✅ STIX/TAXII 2.x support  
✅ Data normalization to internal schema  
✅ Deduplication (e.g., via IOC hash or composite keys)  
✅ IOC tagging (type, source, severity, confidence)  
✅ Simple dashboard or API to query latest IOCs

**Implementation Steps**

1. **Set up storage layer** (PostgreSQL + Elasticsearch/OpenSearch).
2. **Integrate first OSINT feed** (AlienVault OTX) to validate ingestion pipeline.
3. **Implement STIX/TAXII connector** (use stix2 & taxii2-client Python libraries).
4. **Build normalization function** to unify different formats.
5. **Add deduplication logic** (hashing IOCs before insertion).
6. **Implement tagging system** for source, severity, confidence.
7. **Set up API layer** for internal/external queries.
8. **Implement logging & monitoring** (Elastic APM, Prometheus, Grafana).
9. **Run small-scale test** with 2–3 feeds, validate with security engineers.
10. **Scale up to full feed set** once stable.

**Best Practices & Tips**

* **Start small** → one OSINT + one STIX/TAXII feed before scaling.
* **Keep everything modular** → adding/removing feeds should not require core changes.
* **Implement caching** to prevent API overuse.
* **Version-control your schema** to handle future feed changes.
* **Enable API pagination handling** so you don’t lose large datasets.
* **Document all feeds** (source, format, authentication, polling interval).