Epic: Quarantine & Data Ingestion Security

Epic ID: VAULT-E6 **Owner:** CTO **Status:** Scoping **Phase:** MVP (Stages 1-3) / Phase 2 (Stage 4) **Estimated Total Effort:** 80-110 hours

---

Goal

Build a multi-stage quarantine pipeline that scans, validates, and sanitizes every file entering the Vault Cube — whether uploaded via browser, imported from USB, or submitted as training data. This pipeline is the security perimeter for an air-gapped system and a key enterprise sales differentiator.

Why This Matters

Air-gapped doesn't mean safe. The physical transfer mechanisms (USB drives, LAN uploads) become the attack surface. Without quarantine:

- A malware-laden PDF uploaded for RAG analysis could compromise the system
- A poisoned training dataset could backdoor a fine-tuned model
- A pickle-format model file could execute arbitrary code on import
- PII in training data could create compliance liability

Every file entering the system must be treated as untrusted until proven otherwise.

Architecture Overview

```
UNTRUSTED INPUT              QUARANTINE ZONE                    PRODUCTION
(USB / Upload)          (Isolated staging directory)            STORAGE


                       ┌──────────────────────────────┐
   Files ──────────→│ Stage 1: File Integrity      │
                       │  • Magic byte verification   │
                       │  • Structure validation       │
                       │  • Size/count limits          │
                       │  • Archive bomb detection     │
```

```
└──────────────┬──────────────┘
               │  PASS
┌──────────────▼──────────────┐
│ Stage 2: Malware Scan        │
│  • ClamAV (offline sigs)     │
│  • YARA custom rules         │
│  • Hash blacklist            │
└──────────────┬──────────────┘
               │  PASS
┌──────────────▼──────────────┐
│ Stage 3: Content Sanitize    │
│  • Strip macros/JS/ActiveX   │
│  • Re-encode images          │
│  • Rebuild PDFs clean        │
│  • Scrub metadata            │
└──────────────┬──────────────┘
               │  PASS
┌──────────────▼──────────────┐
│ Stage 4: AI Safety (Phase 2) │
│  • Training data validation  │
│  • Prompt injection detect    │
│  • Data poisoning heuristics │
│  • PII scanning              │
│  • Model file validation     │
└──────────────┬──────────────┘
               │
               │
┌──────────────▼──────────────┐
│         DECISION             │
│ ✅ CLEAR → Production         │
│ ⚠️ FLAG → Quarantine Hold     │
│          (Admin review)      │
│ ❌ BLOCK → Auto-reject        │
└─────────────────────────────┘
```

Task Breakdown

Stage 1: File Integrity & Validation

Task	Description	Effort	Dependencies
1.1	**Quarantine directory structure**: Create isolated staging area (`/opt/vault/quarantine/`), separate from production storage. Set filesystem permissions so	2 hrs	None

	production services cannot read quarantine files.		
1.2	**File type verification service**: Implement magic byte checking using `python-magic` . Map claimed MIME type vs actual content. Support: PDF, DOCX, XLSX, TXT, MD, CSV, JSON, JSONL, PNG, JPG, safetensors, GGUF.	4 hrs	None
1.3	**File structure validation**: Per-format deep validation — verify internal structure is well-formed (e.g., valid PDF cross-reference table, valid ZIP structure for DOCX, valid JSON for JSONL).	6 hrs	Task 1.2
1.4	**Size and count limits**: Configurable max file size (default 10GB for models, 500MB for documents), max files per upload batch, total quarantine storage limit with alerts.	2 hrs	Task 1.1
1.5	**Archive bomb detection**: Check compression ratios on ZIP/tar files. Reject nested archives beyond 2 levels. Limit decompressed size to 10x compressed size.	3 hrs	Task 1.2

Stage 1 Total: 17 hours

Stage 2: Malware Scanning (Offline)

Task	Description	Effort	Dependencies
2.1	**ClamAV installation and offline config**: Install ClamAV daemon. Disable freshclam auto-update (air-gapped). Pre-load signature database. Configure as systemd service.	3 hrs	None
2.2	**ClamAV scanning integration**: Python service that submits files to clamd socket, parses results, maps findings to quarantine status (clean/flagged/blocked).	4 hrs	Task 2.1
2.3	**YARA rule engine**: Install YARA, create custom ruleset for AI-specific threats: embedded Python in safetensors, suspicious pickle opcodes, known prompt injection patterns in text files.	6 hrs	None
	Hash blacklist: Local SQLite table of known-bad file hashes (SHA-256). Check every file before deeper		

2.4	scanning. Pre-populated with known malicious model files.	2 hrs	None
2.5	**Signature update mechanism**: Script to extract ClamAV signatures and YARA rules from USB update bundles. Verify GPG signature before importing. Track signature version and date.	4 hrs	Task 2.1, Update mechanism
2.6	**Signature staleness monitoring**: Expose signature age to system health API. Dashboard widget showing last update date with color-coded freshness (green/yellow/red).	2 hrs	Task 2.5

Stage 2 Total: 21 hours

Stage 3: Content Sanitization

Task	Description	Effort	Dependencies
3.1	**PDF sanitization**: Strip JavaScript, embedded files, launch actions, form submissions. Rebuild as clean PDF. Use pikepdf or build on dangerzone approach (render to pixels, re-PDF).	6 hrs	None
3.2	**Office document sanitization**: Strip VBA macros, ActiveX controls, external data connections, OLE objects from DOCX/XLSX. Preserve content and formatting. Use python-docx + openpyxl for targeted stripping.	6 hrs	None
3.3	**Image re-encoding**: Re-encode all images (PNG, JPG, etc.) through PIL/Pillow to strip steganography payloads, embedded data, and malformed headers. Validate dimensions are sane.	3 hrs	None
3.4	**Metadata scrubbing**: Strip EXIF data from images, author/revision info from Office docs, XMP metadata from PDFs. Configurable: strip all vs. preserve selected fields.	3 hrs	Tasks 3.1-3.3
3.5	**Sanitization report**: Generate per-file report of what was stripped. Store with file metadata. Surface in quarantine scan results API.	2 hrs	Tasks 3.1-3.4

Stage 3 Total: 20 hours

Stage 4: AI-Specific Safety Checks (Phase 2)

Task	Description	Effort	Dependencies
4.1	**Training data format validation**: Verify JSONL structure, required fields (prompt/completion or messages format), character encoding. Report malformed rows with line numbers.	4 hrs	None
4.2	**Training data quality analysis**: Statistical profiling — distribution of lengths, class balance, duplicate detection, outlier identification. Generate quality report with warnings.	6 hrs	Task 4.1
4.3	**Prompt injection detection**: Pattern matching + heuristic scoring for known injection patterns in training data ("ignore previous instructions", "system:", role confusion attempts). Configurable sensitivity.	6 hrs	None
4.4	**PII scanning engine**: Regex patterns for structured PII (SSN, credit card, phone, email, medical record numbers) + local NER model (spaCy or similar) for names, addresses, dates of birth.	8 hrs	None
4.5	**PII action configuration**: Admin-configurable response to PII findings: `flag` (hold for review, show redacted preview), `redact` (auto-mask and proceed), `block` (reject file).	4 hrs	Task 4.4
4.6	**Model file validation**: For safetensors imports: verify format (reject pickle/PT files), validate tensor shapes match declared architecture, check file hash against known model registry (pre-loaded).	4 hrs	None
4.7	**Data poisoning heuristics**: Flag training examples with anomalous characteristics: extreme length outliers (>3 std dev), repetitive patterns suggesting duplication attacks, high perplexity examples that don't fit the dataset distribution.	6 hrs	Task 4.2

Stage 4 Total: 38 hours

Pipeline Integration & API

Task	Description	Effort	Dependencies
5.1	**Pipeline orchestrator**: Async job runner that sequences files through stages. Handles parallel scanning of multiple files. Manages state transitions (scanning → flagged → approved/rejected). Uses Celery or simple asyncio task queue.	6 hrs	All stages
5.2	**Quarantine API endpoints**: Implement all 9 quarantine endpoints in FastAPI gateway. Integrate with pipeline orchestrator.	6 hrs	Task 5.1
5.3	**Quarantine hold workflow**: Admin review UI integration — list held files, show findings per stage, approve/reject with reason, audit log integration.	4 hrs	Task 5.2
5.4	**Integration with upload flows**: Wire quarantine into existing file upload paths — document upload (RAG), training data submission, model import, USB import. All paths funnel through quarantine.	4 hrs	Task 5.2, other epics
5.5	**Integration testing**: End-to-end tests with known-bad files: EICAR test file, macro-laden DOCX, pickle model file, JSONL with prompt injections, files with embedded PII. Verify each stage catches its targets.	6 hrs	All tasks

Integration Total: 26 hours

Effort Summary

Component	Effort	Phase
Stage 1: File Integrity	17 hrs	MVP
Stage 2: Malware Scanning	21 hrs	MVP
Stage 3: Content Sanitization	20 hrs	MVP

Stage 4: AI-Specific Checks	38 hrs	Phase 2
Pipeline Integration & API	26 hrs	Split
Total	**122 hrs**	
MVP Scope (Stages 1-3 + Integration)	**~78 hrs**	
Phase 2 Addition (Stage 4)	**~44 hrs**	

Key Technology Choices

Component	**Tool**	**Why**
File type detection	python-magic (libmagic)	Industry standard, no network, fast
Malware scanning	ClamAV	Only mature open-source AV with offline mode
Custom threat rules	YARA	Flexible pattern matching, standard in security
PDF sanitization	pikepdf + dangerzone approach	Proven, offline, thorough
Office sanitization	python-docx, openpyxl	Direct XML manipulation, no LibreOffice dependency
Image processing	Pillow (PIL)	Standard, fast re-encoding
PII detection	Regex + spaCy NER	Offline, accurate, configurable
Task queue	asyncio (MVP) / Celery (scale)	Start simple, scale if needed
File hashing	hashlib (SHA-256)	Standard library, no dependencies

---

Risks & Mitigations

Risk	Impact	Mitigation
ClamAV signature staleness	Misses recent malware	Bundle fresh signatures with every update; dashboard warnings for stale sigs
False positives blocking legitimate files	User frustration, lost trust	Admin override with quarantine hold; tunable strictness levels per stage
PII scanner over-flagging	Noise in results	Confidence scoring, configurable thresholds, preview before action
Large file scanning performance	Slow imports, poor UX	Async scanning with progress reporting; parallel processing of multiple files
New file formats not supported	Bypass quarantine	Strict allowlist approach — unknown formats are blocked by default, not passed through
Sanitization breaks document formatting	Customer data degradation	Sanitization preserves content fidelity; original stored in quarantine as backup until admin confirms clean version is acceptable

---

Success Criteria

- Every file entering production storage has a quarantine scan record in the audit log
- Known-bad test files (EICAR, macro DOCX, pickle models, injection JSONL) are caught 100% of the time
- Admin can review, approve, and reject flagged files through the management UI
- ClamAV signature age is visible on the dashboard
- PII scanning correctly identifies SSN, credit card, and medical record number patterns in test datasets
- Zero production files bypass quarantine — enforced at the filesystem permission level

---

Open Questions

1. **Should sanitized files replace originals or exist alongside them?** Current plan: sanitized version goes to production, original retained in quarantine archive for admin reference. Storage implications need sizing.
2. **How aggressive on PII default?** Recommendation: `flag` mode as default (alert admin, don't auto-redact). Healthcare customers may want `redact` as default. Make it a first-boot setup choice.
3. **Quarantine for inference prompts?** Currently scoped for files only. Should the API gateway also scan user prompts for injection attacks? This is a different problem (real-time vs. batch) and likely a separate epic.
4. **File format allowlist scope for MVP:** Recommend starting with: PDF, DOCX, XLSX, TXT, MD, CSV, JSON, JSONL, PNG, JPG, safetensors. Add more formats in Phase 2 as customer demand dictates.