Security Architecture Assessment: Integrinow DMS v2

1. Architectural Foundations and Security Philosophy

In a multi-tenant document environment, the strategic imperative of "Defense-in-Depth" cannot be overstated. The Integrinow DMS v2 architecture is engineered specifically to support a "defensible chain of custody," ensuring that no document reaches durable storage without first passing through a non-bypassable security gauntlet. This philosophy mandates that security is not a single gateway, but a series of overlapping controls—logical, cryptographic, and procedural—that must be satisfied before data is promoted from a transient upload to an organizational record.The multi-tenant isolation model is the primary defensive layer. The system enforces strict organizational scoping at the database and object-storage levels, ensuring that the principle of least privilege is baked into the query logic itself.| System Role | Security Scoping || ------ | ------ || **Customer** | Restricted to own organization (org_id) only. || **Processor** | Cross-organization visibility restricted to documents they have personally handled. || **Reviewer** | Cross-organization visibility restricted to the NEEDS_REVIEW queue only. || **Admin** | System-wide visibility for auditing, chain verification, and disaster recovery. |
The logical integrity of the document lifecycle is enforced by a "Fixed State Machine." Every document must move through a rigid progression (e.g., RECEIVED to AV_SCANNING to NEEDS_REVIEW). The implementation of the "no-backward-transition" rule is a critical security control; it prevents a document from being moved back to an earlier stage of the pipeline to bypass an automated scan or a human review. Furthermore, the terminal states of REJECTED and ARCHIVED function as "dead-ends," effectively eliminating logical bypasses and ensuring a predictable, auditable path for every asset.This logical state machine provides the necessary framework for the physical automated pipeline that enforces these security transitions.

2. The Multi-Layered Protection Pipeline

The architecture utilizes an asynchronous Celery worker pipeline to decouple the public-facing request path (Flask) from the intensive security processing. This separation is a strategic choice for system resilience: it isolates the attack surface of the heavy processing tools from the API and ensures that resource-intensive operations, such as malware scanning or high-entropy encryption, do not compromise the availability of the management interface.The automated pipeline is deconstructed into four primary security stages:

1. **Threat Detection:** Documents are streamed to a ClamAV scanner using the INSTREAM protocol. This implementation represents a critical remediation of a historical vulnerability where path-based scanning was used; by streaming bytes directly to the daemon, the system avoids the risks associated with insecure file paths and local file system access.
2. **Integrity Baseline:** Immediately following a clean scan, the system generates a SHA-256 hash of the "as-uploaded" bytes. This serves as the immutable reference point for "what the customer actually uploaded," providing a baseline for all subsequent transformations.
3. **Automated Processing:** A pluggable transformation stage is included to handle format-specific logic. Currently, this acts as a robust placeholder—timestamping plain

text and Markdown files while leaving binary formats (PDFs, images) untouched to prevent corruption in the absence of specific client requirements.

4. **Final Persistence:** Only after all previous stages succeed does the document transition from the transient, non-locked "Quarantine" bucket to the hardened, WORM-protected "Vault" storage.The asynchronous model ensures that sensitive data is never "at rest" in a vulnerable, unencrypted state. The isolation between the volatile Quarantine bucket and the hardened Vault bucket ensures that the attack surface of unverified data is strictly limited to the duration of the scan and transformation.The concluding step of this pipeline is the application of the cryptographic mechanisms that secure the data within the Vault.

3. Cryptographic Framework: Envelope Encryption & Vault Integration

The Integrinow DMS v2 employs "Envelope Encryption" to achieve per-file cryptographic isolation. By separating the Data Encryption Key (DEK), which encrypts the file content, from the Key Encryption Key (KEK), which protects the DEK, the system centralizes trust within HashiCorp Vault. This approach ensures that even a total compromise of the application database does not result in the exposure of document contents, as the KEK never leaves the Vault's hardened boundary.The cryptographic lifecycle involves local generation of a DEK using the Fernet algorithm. The document is encrypted, and the DEK is then "wrapped" via the Vault Transit Engine (transit/encrypt/dms-file-kek). Only the wrapped ciphertext of the DEK is persisted in the PostgreSQL database.The Vault operational configuration is designed for enterprise-level assurance:

- **Shamir Secret Sharing:** The Vault requires a 3-of-5 threshold of shares for manual unsealing, preventing any single administrator from accessing the cryptographic master keys.
- **Auto-Unseal Capability:** For production environments, the system supports a cloud-native path using AWS KMS, GCP KMS, or Azure Key Vault to automatically decrypt the master key at boot, removing human-in-the-loop risks.
- **Root Token Hygiene:** The initial root token is used only for bootstrap and is immediately revoked. Any subsequent root-level access requires a deliberate, multi-custodian regeneration process.The KeyManagementService utilizes a "Fail-Fast" design. It validates Vault authentication and seal status during initialization; should Vault be sealed or the token be invalid, the system raises a RuntimeError immediately. This prevents the pipeline from failing silently or inadvertently processing documents without the required cryptographic protections.Once the data is encrypted, the system shifts focus to maintaining the physical and logical immutability of the storage and its audit trails.

4. Immutable Data Integrity: WORM Storage & Hash-Chained Auditing

Non-repudiation is achieved through the combination of WORM (Write-Once-Read-Many) storage and a hash-chained audit log. Together, these create a mathematically verifiable record of all system activity that is resistant to historical tampering.The MinIO "Vault Bucket" is configured with Object Locking in GOVERNANCE mode. This mode was selected over COMPLIANCE mode to provide necessary operational flexibility, allowing privileged

administrators to manage retention while still blocking ordinary deletes and overwrites—a policy decision that supports the system's role as a platform for diverse client requirements.The system utilizes a hash-chained audit log to track every state change. Every AuditLogEntry is linked to its predecessor's hash, and the write process is serialized using **PostgreSQL advisory transaction locks** (pg_advisory_xact_lock). This specific locking mechanism is essential to prevent "chain-forking" during concurrent Celery worker operations, ensuring the integrity of the linear chain.

```
{
  "id": 1024,
  "prev_hash": "a58f32...",
  "entry_hash": "b72e11...",
  "actor_id": "uuid-here",
  "action": "ENCRYPTED_TO_VAULT",
  "resource_type": "document",
  "resource_id": "doc-uuid",
  "event_metadata": {
    "vault_key": "storage-path",
    "status": "NEEDS_REVIEW"
  },
  "created_at": "2023-10-27T10:00:00Z"
}
```

The verify_chain() function acts as a forensic validator. By recomputing every hash in sequence, it can detect any historical UPDATE or DELETE operation performed directly on the database. This provides a high degree of confidence in the system's historical records.The assurance of data immutability is further supported by the governance of the software itself and the resilience of its recovery mechanisms.

5. Operational Governance and Source Integrity

Environmental integrity is maintained by extending security controls beyond the data layer to the source code and the backup ecosystem. The architecture ensures that tampered code cannot execute and that the system can be recovered from a total failure without compromising security properties.The Source Integrity Verification (SIV) system fingerprints critical project files and signs the manifest using Ed25519 keys within Vault. A key security property of this system is its **asymmetric permission model** : the running application is granted a "verify" policy but is strictly denied a "sign" policy. This ensures that if a process is compromised, the attacker cannot forge a new signature to hide their modifications. The verify_at_startup() function enforces this check before the application binds to the network.The Enterprise Backup & Restore strategy ensures business continuity:

- **Envelope-Encrypted Archives:** All backup archives (tar.gz.enc) are encrypted using the same Vault-backed infrastructure as document contents.
- **Multi-Cloud Replication:** Through BACKUP_REMOTE_TARGETS, the system supports simultaneous replication to AWS, GCP, and Azure, with cloud credentials stored securely in Vault.Authorization enforcement is handled at the query level. RBAC

and JWT tokens are validated against the org_id of the requested resource. This ensures that cross-tenant data leakage is prevented not just in the UI, but as a core constraint of the data access layer.This comprehensive approach to governance brings the system to a state where risks can be clearly identified and prioritized.

6. Assessment of Technical Debt and Security Gaps

In a professional cybersecurity context, security is an iterative process. The following gaps are identified not as failures of the current architecture, but as prioritized roadmap items to improve the system's long-term maturity and resilience.**Known Gaps / Technical Debt:**

- **Credential Management:** The Vault application token currently utilizes a non-expiring TTL (ttl: 0). This is a significant risk window; production deployments should transition to short-lived, renewable tokens.
- **Operational Automation:** Initial Vault configuration, including transit engine mounting and KEK creation, remains a manual process. This should be automated via infrastructure-as-code to ensure consistency and repeatability.
- **Protocol Hardening:** The internal application uses a self-signed TLS certificate. While proxied by a valid certificate at the public edge, end-to-end encryption should be hardened with a managed internal CA or automated certificate rotation.
- **Software Lifecycle:** The absence of a schema migration tool (e.g., Alembic) represents technical debt that complicates the safe evolution of the data model and increases the risk of manual database errors.The two most critical gaps are the **non-expiring Vault token** and the **lack of a schema migration tool** . The former increases the blast radius of a credential leak, while the latter threatens the maintainability of the data-integrity constraints as the system scales.In conclusion, the Integrinow DMS v2 security posture is robust and well-suited for use-cases requiring a defensible chain of custody. Its implementation of envelope encryption, WORM storage, and tamper-evident auditing provides a professional-grade foundation for sensitive document management.