Team Project Report

Design Proposal for Digital Forensics Agent System (DFAS)

(Intelligent Agents Module)

Group: Group C

Date: 08/09/2025

Table of Contents

1. Introduction

This project builds a small, reliable digital-forensics agent for file-system collection and triage. It locates specified file types using read-only acquisition and chain-of-custody logging, processes them (hashing, metadata, optional YARA), and securely stores or sends a tamper-evident encrypted bundle plus CSV/JSON reports aligned with standard digital-evidence guidance.

2. Scope and context

We will build a small set of cooperating agents that run on a workstation or server. They will:

- Search local disks for target file types (e.g., *.docx, *.xlsx, *.pdf, *.jpg, *.zip, *.pst*).
- Record evidence: file path, size, times, file type (magic), hashes (SHA-256), and who/what collected it.
- Process the files: hash, optional YARA pattern scan, and archive + encrypt the results.
- Store/send outputs: write a local evidence store and, if allowed, upload the encrypted package to a secure server/S3.
- Keep a chain-of-custody log that cannot be edited.

This follows accepted guidance for handling digital evidence: identify → collect/acquire → preserve → analyze → report (ISO/IEC 27037; NIST SP 800-86; ACPO principles).

3. System requirements (what we need to run it)

Runtime

- OS: Windows 10/11, Ubuntu 22.04+, macOS 13+ (tested on Windows + Ubuntu).

- Language: Python 3.11+.

Python packages

- watchdog (file system events; optional live mode)
- yara-python (pattern scan with YARA rules)
- python-magic or filemagic (libmagic wrapper for file type identification)
- cryptography (AES-GCM encryption of archives)
- py7zr or stdlib zipfile (archiving)
- pydantic (typed models for evidence metadata)
- sqlalchemy + sqlite3 (local evidence catalog)
- loguru (structured logs)
- retrying/tenacity (robust retries)
- requests or boto3/minio (HTTPS/S3 upload)
- tqdm (progress when run interactively)

External tools (optional)

- The Sleuth Kit (TSK) for low-level file system analysis when disk images are used.
- Autopsy (only if a GUI review is needed). Not required for agent runtime.

Infrastructure (optional)

- Message queue (e.g., RabbitMQ/Redis) if we scale to multiple hosts.
- S3-compatible bucket (e.g., MinIO or AWS S3) for central evidence storage.

Why these?

- Python is widely used in DFIR, portable, and has mature libraries.
- libmagic reliably detects file types using headers, which reduces reliance on file extensions.
- YARA is a standard way to match known patterns (e.g., malware or sensitive keywords) during triage.

- SHA-256 (from FIPS 180-4) is a standard hash for integrity.
- AES-GCM gives encryption and integrity for evidence packages.

4. Architecture overview

We use multiple small agents that follow a simple Belief-Desire-Intention (BDI) style:

- Beliefs: current facts (found files, hashes, queue states).
- Desires: goals (discover files, package evidence, transmit package).
- Intentions: chosen actions (scan now, hash batch, upload now).

Agents and roles

- Orchestrator Agent – loads policy, assigns work, monitors health.
- Discovery Agent – walks file system, filters by file type/size/date/path, emits "evidence candidates".
- Processing Agent – hashes files, extracts metadata, runs optional YARA, writes evidence records.
- Packaging Agent – builds encrypted archives, writes CSV/JSON report, records chain-of-custody.
- Transport Agent – uploads to secure server/S3 with retries; validates hash after upload.

Data stores

- Evidence DB (SQLite): metadata + audit trail (append-only table with cryptographic seal).
- Local Vault: staged files and encrypted archives (with read-only mount when possible).

Why multi-agent?

- Clear separation of duties; easier testing and scaling.
- Fail fast and retry per stage (resilient to network or permission errors).

- Maps well to BDI ideas and to agent engineering methods.

5. Detailed design choices

5.1 Agent model: BDI-inspired micro-agents

- Reasoning: BDI (Belief–Desire–Intention) gives a practical way to model goal-directed behavior with plans and reactive triggers. Our agents need simple, robust plans (e.g., “scan, then hash, then package”) that can pause/resume if a step fails.
- Implication: Each agent keeps a small state (beliefs), a goal set (desires), and a plan queue (intentions). Plans are rule-based (e.g., “If queue length > N, spawn another hasher”).

5.2 Evidence handling: follow ISO/NIST/ACPO

- Reasoning: To be court-ready, we must protect integrity and maintain an audit trail. We adopt ISO/IEC 27037 for identification/collection/preservation, ISO/IEC 27042 for analysis, ISO/IEC 27043 for investigation process, ISO/IEC 27041 for method assurance, NIST SP 800-86 for incident forensics integration, and ACPO four principles for UK contexts. We implement SHA-256 hashing per FIPS 180-4 and keep a tamper-evident log.
- Implication: Read-only acquisition when possible; never alter original data; keep detailed logs of who/what/when/where; verify hashes at each transfer.

5.3 File identification: libmagic over extensions

- Reasoning: File extensions can be wrong or missing. libmagic reads headers (“magic numbers”), so detection is more reliable in triage. We still record the original extension for context. Known caveat: libmagic has limits; we mitigate by combining with YARA and size/path filters.

5.4 Pattern scanning: YARA (optional policy)

- Reasoning: Many teams already maintain YARA rule sets to tag malware or sensitive content during triage. Running YARA at collection time adds value without heavy compute if we cap file sizes and throttle.

5.5 Integrity + confidentiality: SHA-256 + AES-GCM

- Reasoning: SHA-256 is a widely accepted integrity hash; AES-GCM gives authenticated encryption, so tampering is detectable. Keys are in OS key vault/Key Management Service (KMS), never hard-coded.

5.6 Storage: SQLite catalog + encrypted archive

- Reasoning: SQLite is simple, portable, and good for a single-host agent. Evidence files are stored as immutable encrypted archives. For central storage, we use S3-compatible buckets with server-side encryption.

5.7 Orchestration and messaging: lightweight queues

- Reasoning: A local in-process queue (or Redis/RabbitMQ if scaled) lets agents be decoupled. If the hasher slows down, the scanner can pause.

5.8 Development methodology: Agile + Prometheus AOSE artifacts

- Reasoning: We deliver in small sprints and use Prometheus (an agent-oriented method) artifacts—goals, roles, scenarios, and data—because it is designed for practitioners and fits BDI-like agents.

6. Approaches and methodology

Process

- Sprints of 1–2 weeks: backlog → design sketch → implement → test → demo → refine.

- Prometheus-style artifacts: system goals (e.g., “Collect files of types X within Y hours”), roles (Discovery/Processing/etc.), scenarios (normal vs. network down), and data definitions (evidence record schema).

Development techniques

- Test-driven development (unit tests for hashing, YARA tagging, archive integrity; golden sample corpus).
- Static typing with Pydantic models (clear schemas for evidence records).
- Secure coding: no plaintext keys; least privilege file access; robust error handling and retries.
- Observability: structured logs + metrics (files/min, errors, queue depth).

Deliverables

- Agents as installable Python packages, a CLI, and policy YAML.
- Admin guide and quick-start script.
- Sample dashboard (optional) that reads the SQLite/CSV export.

7. Processing pipeline

1. Plan: Orchestrator loads policy (targets, paths to include/exclude, file size caps, YARA on/off).
2. Discover: Discovery Agent walks paths (or uses Watchdog in live mode), applies filters, and places candidates on the queue.
3. Process: Processing Agent reads candidates, computes SHA-256, extracts metadata (size/times/owner/type via libmagic), runs YARA if enabled, writes an evidence record to SQLite and to a newline-delimited JSON (NDJSON) log.
4. Package: Packaging Agent groups records into cases/batches, creates CSV + JSON report, builds an AES-GCM encrypted 7z/ZIP with files + report, and appends to the chain-of-custody table (who/what/when, plus hash of package).
5. Transmit: Transport Agent uploads the archive over TLS (HTTPS/S3), verifies the server-side checksum, and stores the receipt.

6. Present: Analysts can open the CSV/JSON, or ingest into a dashboard/Autopsy for deeper analysis.

8. Security, safety, and compliance

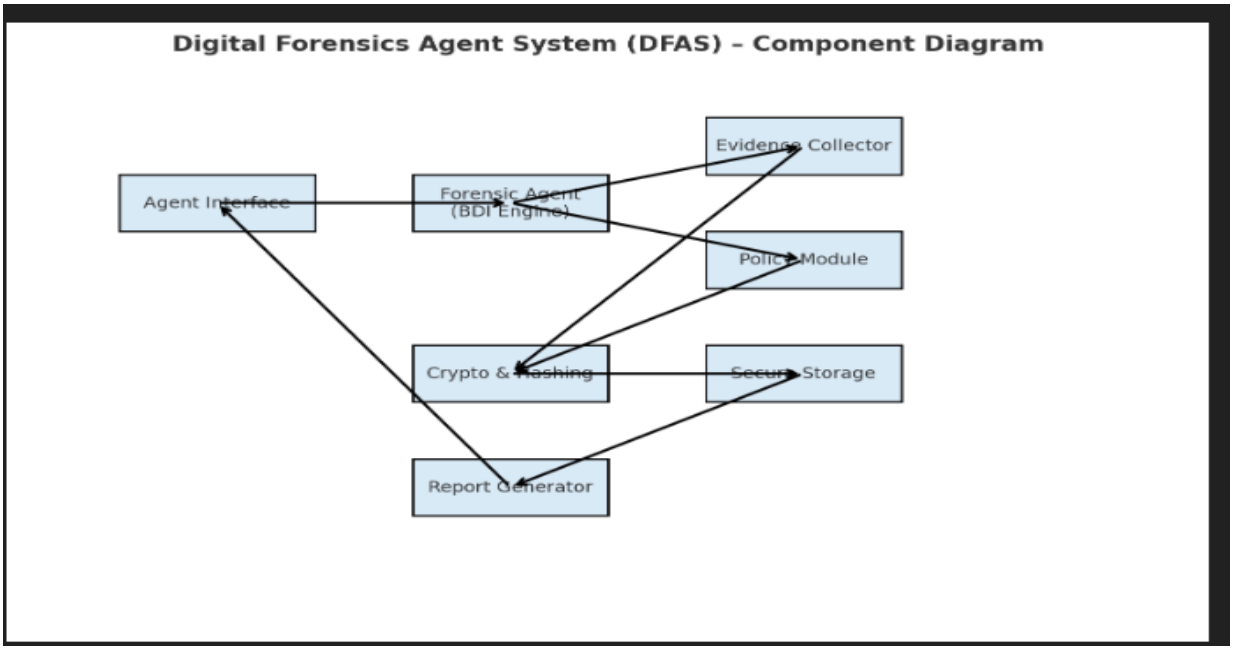
- Immutability: Original files are read-only; we copy into archives; no in-place edits.
- Integrity: SHA-256 at source; verify at package; verify after upload.
- Confidentiality: AES-GCM with rotated keys (OS key vault/KMS). Access is role-based.
- Auditability: Append-only logs with periodic hash chaining (each entry stores the hash of the previous entry to detect tampering).
- Privacy: Policy can exclude paths (e.g., personal folders) and set size/type limits. Data minimization is the default.

9. Challenges and how we will handle them

1. Chain of custody & integrity – If logs are changed, evidence may be challenged. Mitigation: append-only logs, hash-chained audit records, independent verification using SHA-256; follow ISO/IEC 27037 and ACPO principles.
2. File type spoofing – Extensions can lie. Mitigation: use libmagic to read headers; cross-check with YARA and mime; flag mismatches for review.
3. Large data volumes – Scans can be slow. Mitigation: path whitelists/blacklists, size caps, multi-process hashing, resume from last checkpoint, and queue back-pressure.
4. Locked/in-use files – Some files cannot be read. Mitigation: retry with back-off; capture metadata even when content cannot be copied; where lawful, use shadow-copy (Windows VSS) or LVM snapshots.
5. Performance vs. thoroughness – Deep scans can be costly. Mitigation: two modes: triage (fast) and full (complete). Clear policy switches.
6. Security of stored evidence – If archives leak, it is a breach. Mitigation: AES-GCM encryption, minimum cipher strength, rotated keys, secure transport, and strict access control.

7. Tool assurance – We must show the tool is fit for purpose. Mitigation: follow ISO/IEC 27041 for method assurance; keep versioned build artifacts; run known-answer tests (KATs) and keep validation packs.
8. OS diversity – File APIs differ. Mitigation: abstract file access; test on Windows and Linux; document limits on macOS if any.
9. Rule quality (YARA) – Bad rules create noise. Mitigation: rule review, namespacing, disable on slow disks, size/type caps; keep a trusted rule set.

10. Design diagrams (text-UML)



10.1 Main Sequence

Operator -> Orchestrator: Load policy
Orchestrator -> Discovery: Start scan(paths, filters)
Discovery -> Queue: Enqueue(file_candidate)
Processing -> Queue: Dequeue(file_candidate)
Processing -> File System: Read file (RO)

Processing -> Processing: SHA-256, libmagic, (YARA?)

Processing -> Evidence DB: Insert evidence_record

Packaging -> Evidence DB: Fetch batch

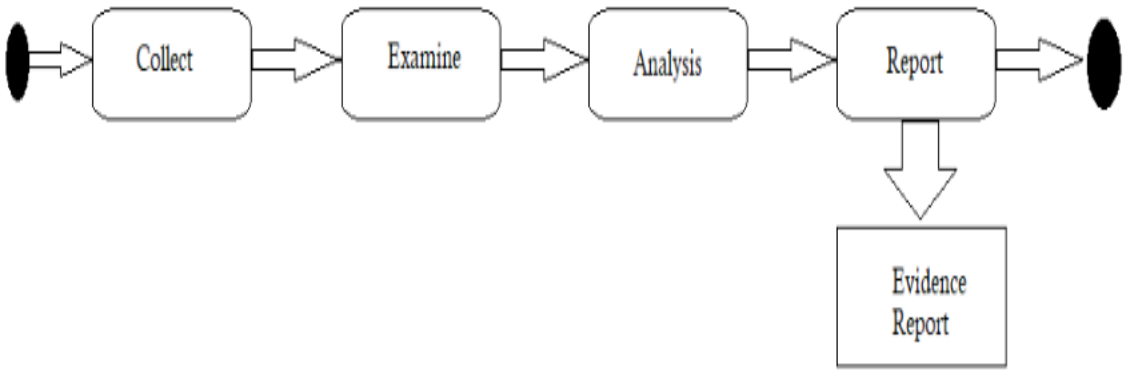
Packaging -> Packaging: Build CSV/JSON + Archive (AES-GCM)

Packaging -> Evidence DB: Append chain-of-custody entry (hash)

Transport -> Secure Server: Upload archive (TLS)

Secure Server -> Transport: Return checksum/receipt

Transport -> Evidence DB: Store receipt + verify hash



11. Data model (concise)

Evidence Record

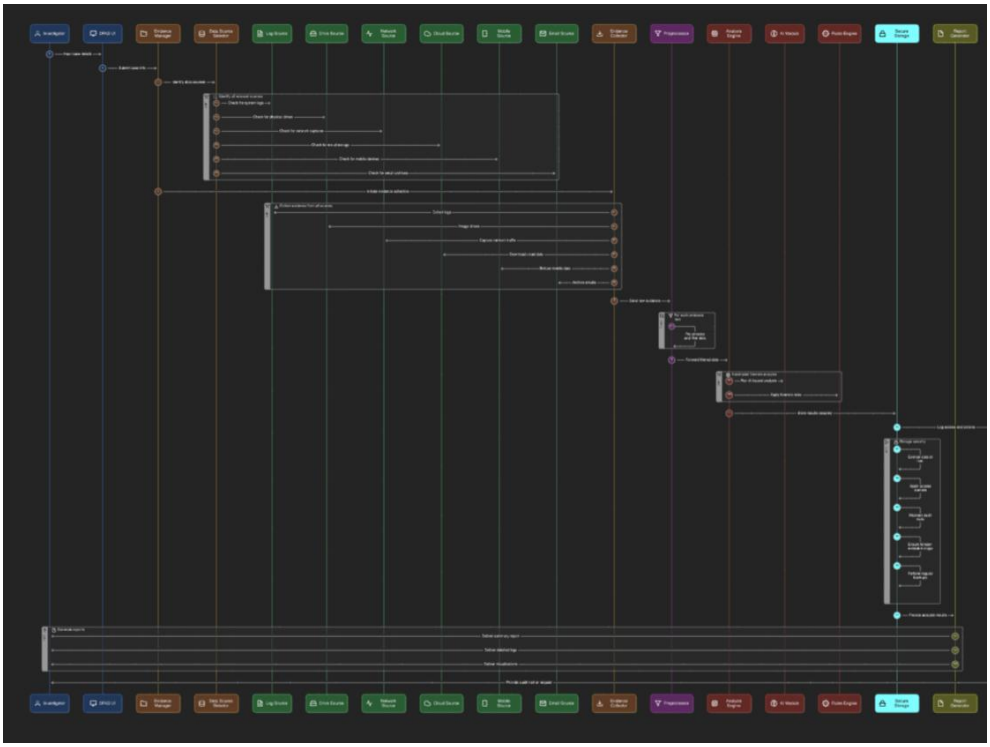
- id (uuid), case_id, file_path, rel_path, size, created/modified/accessed times, owner, type (libmagic), extension, sha256, yara_tags[], collected_by (host/user/version), collected_at (UTC), notes.

ChainEntry

- id, case_id, action (collect/package/upload/verify), actor (agent id), timestamp, prev_hash, entry_hash, details.

PackageManifest

- package_id, case_id, file_count, total_bytes, sha256, cipher (AES-GCM), key_id, policy_version.



12. Build & deployment

- Install: python -m pip install dfas (internal package). dfas init to create policy; dfas run to execute.
- Config: YAML policy (targets, paths, excludes, size caps, YARA on/off, upload endpoint/KMS key id).
- Run modes: scan-once (batch) or watch (continuous).
- Packaging: Docker image (if needed) with non-root user; Linux capability drop.
- Logs: JSON logs to disk + optional syslog; rotate.

13. Testing & validation

- Unit tests: hashing, libmagic type mapping, YARA runner, archive build, AES-GCM.

- Integration: end-to-end on sample corpus (good files, spoofed extensions, locked files, big files).
- Repeatability: same input → same hashes; record seeds/versions.
- Assurance (27041): validation pack with expected outputs and checksums; signed test report.

14. Strengths, weaknesses, and tradeoffs

Strengths

- Standards-aligned evidence handling (ISO/NIST/ACPO).
- Modular agents (isolate faults; scale where needed).
- Simple, portable runtime (Python + SQLite + archives).

Weaknesses

- Python is slower than C/C++; heavy scans can take longer.
- libmagic and YARA need maintenance (rules, signatures).
- Local SQLite may not suit very large multi-user deployments (we can switch to Postgres).

Why we accept them

- For triage and first-pass collection, speed is good enough; we can scale by parallel workers and selective scans. Maintenance is a normal DFIR task (update rules monthly). For bigger sites, we provide a DB adapter.

15. Implementation roadmap (brief)

- Sprint 1: CLI + config + Discovery → Processing → SQLite → CSV.
- Sprint 2: Packaging (AES-GCM 7z/ZIP), chain-of-custody, receipts.
- Sprint 3: YARA optional scan; S3 upload with verify.

- Sprint 4: Live mode (Watchdog), dashboards/reports, hardening and validation pack.

16. Conclusion

In sum, the agent fulfills its mandate: it reliably finds and acquires target files, processes them (hashing, metadata, optional YARA), and delivers a tamper-evident encrypted archive with CSV/JSON outputs ready for dashboards. The workflow is repeatable, auditable, and aligned with standard digital-evidence guidance supporting fast, defensible analysis.

17. References

- ACPO (2012) *ACPO Good Practice Guide for Digital Evidence (v5).* Association of Chief Police Officers. Available at: https://www.digital-detective.net/digital-forensics-documents/ACPO_Good_Practice_Guide_for_Digital_Evidence_v5.pdf (Accessed: 28 August 2025).
- Cichonski, P. et al. (2012) *Computer Security Incident Handling Guide (SP 800-61r2).* NIST. Available at: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-61r2.pdf (Accessed: 28 August 2025).
- ISO/IEC (2012) *27037: Information technology — Security techniques — Guidelines for identification, collection, acquisition and preservation of digital evidence.* ISO/IEC. (Accessed: 28 August 2025).
- ISO/IEC (2015) *27042: Information technology — Security techniques — Guidelines for the analysis and interpretation of digital evidence.* ISO/IEC. (Accessed: 28 August 2025).
- ISO/IEC (2015) *27041: Information technology — Security techniques — Guidance on assuring suitability and adequacy of incident investigative method.* ISO/IEC. (Accessed: 28 August 2025).
- ISO/IEC (2015) *27043: Information technology — Security techniques — Incident investigation principles and processes.* ISO/IEC. (Accessed: 28 August 2025).
- Kent, K. et al. (2006) *Guide to Integrating Forensic Techniques into Incident Response (SP 800-86).* NIST. Available at: https://csrc.nist.gov/pubs/sp/800/86/final (Accessed: 28 August 2025).
- NIST (2015) *FIPS PUB 180-4: Secure Hash Standard (SHS).* NIST. Available at: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf (Accessed: 28 August 2025).
- Padgham, L. and Winikoff, M. (2004) 'The Prometheus Methodology for building agent-oriented systems', available via ResearchGate (Accessed: 28 August 2025).

- Rao, A.S. and Georgeff, M.P. (1995) ‘BDI Agents: From Theory to Practice’, *Proceedings of ICMAS’95.* AAAI Press. (Accessed: 28 August 2025).
- The Sleuth Kit Project (2003–2023) *The Sleuth Kit (TSK) documentation.* Available at: https://www.sleuthkit.org/sleuthkit/ (Accessed: 28 August 2025).
- Victor M. Alvarez (2023) *YARA Documentation (Release 4.x).* Available at: https://yara.readthedocs.io/ (Accessed: 28 August 2025).
- Wooldridge, M. (2009) *An Introduction to MultiAgent Systems (2nd ed.).* Wiley. (Accessed: 28 August 2025).