Presentation Transcript

SLIDE 1:

Good morning everyone. Today I'm presenting the Digital Forensics Agent System, or DFAS, which is a BDI-inspired multi-agent system designed for automated digital evidence collection. This project shows that the practical application of intelligent agent principles to solve the real world forensic challenges while maintaining the strict accordance with the international standards.

SLIDE 2:

Let me start by explaining what dfas actually is and why we need it. Dfas is a professional-grade digital forensics tool that automates the evidence collection process. In traditional forensic investigations, investigators manually search for files, calculate hashes and document everything which is time taking and can contains the error. DFAS automates this entire workflow.

The system is built using a Belief Desire Intention (BDI) architecture, which is a well-established agent paradigm in artificial intelligence. This approach allows our agents to maintain beliefs about the current state of the system which have desires or goals they want to achieve, and form intentions about which actions to take.

Our key objectives were clear from the start. First, we needed to automate evidence discovery and metadata extraction to save time and reduce human error. Second, we had to maintain compliance with international forensic standards like ISO 27037 and NIST SP 800-86 to ensure our evidence would be admissible in court. Third, we implemented SHA-256 cryptographic hashing for integrity verification, which is the gold standard in digital forensics. Fourth, we generate multiple report formats including CSV, JSON, and SQLite databases to support different stakeholders and tools. Finally, we create encrypted evidence packages using AES-GCM encryption for secure storage and transport.

The multi agent approach was a design choice. Each agent specializes in a specific forensic task, which follows the principle of separation of concerns. Agents work concurrently, which improves performance significantly compared to sequential processing. The modular design means we can easily extend or modify individual agents without affecting the entire system. The BDI architecture make sure that that agents exhibit goal-oriented behavior, making decisions based on their current beliefs and desired outcomes. Finally, queue-based communication between agents ensures data integrity and allows for backpressure handling when one agent becomes slower than others.

SLIDE 3:

Now let's look at the architecture. We have four main agents, each with distinct responsibilities.

The Orchestrator Agent sits at the top of our hierarchy. It loads the forensic collection policy from a YAML configuration file, initializes all subordinate agents, monitors system-wide execution and health, manages inter-agent communication queues and handles graceful shutdown and error recovery. Think of it as the conductor which coordinating all the other agents.

The Discovery Agent is our file scanner. It recursively scans specified filesystem paths, filters files by extension, size, and location, implements exclude patterns to avoid system files and directories, populates the processing queue with discovered files and logs discovery statistics. In our test run, this agent successfully discovered seventeen files matching our criteria.

The Processing Agent handles the heavy computational work. It calculates SHA-256 cryptographic hashes for each file, extracts comprehensive metadata including timestamps, ownership, and file size, detects actual file types using the libmagic library rather than relying on extensions, stores evidence records in our SQLite database, and handles locked files and access errors gracefully without crashing the entire system.

The Packaging Agent creates the final deliverables. It exports evidence records to both CSV and JSON formats for different audiences, creates encrypted ZIP packages using AES-GCM authenticated encryption, generates chain of custody documentation for legal compliance, calculates package integrity hashes, and organizes all outputs in timestamped directories for easy archival.

The diagram you see here shows how these agents communicate through message queues, which is a key aspect of our design. This queue-based approach allows agents to work together.

SLIDE 4:

The technical implementation relies on carefully chosen technologies and standards. Let me explain our key decisions.

For cryptographic hashing, we use SHA-256, which is specified in FIPS 180-4. We chose this specifically because it's widely accepted in court proceedings internationally. The 256-bit hash provides strong collision resistance, meaning it's computationally infeasible for two different files to produce the same hash. We use this hash both for individual file integrity verification and for package verification.

File type detection was a crucial decision. We use libmagic as our primary detection method because it reads file headers, which are much harder to spoof than file extensions. If libmagic is unavailable, we fall back to Python's mimetypes module. As a final fallback, we have a manual extension mapping. This multi-tier approach prevents file type spoofing, which is critical in forensic investigations where adversaries might try to hide evidence by renaming files.

For data storage, we chose SQLite. It's a portable single-file database format with zero configuration required, making deployment simple. It provides ACID compliance for data integrity, and supports the forensic queries we need for timeline analysis and reporting.

Our encryption uses AES-GCM, which is authenticated encryption. This means it provides both confidentiality and authenticity. The 256-bit key strength is industry standard, and the authentication tag makes tampering detectable. This is crucial for maintaining chain of custody in forensic evidence.

All these choices align with standards from ISO, NIST and ACPO, which are the internationally recognized frameworks for digital forensics.

SLIDE 5:

Let me walk you through the actual evidence collection process, which happens in five phases.

Phase one is configuration. The system reads the dfas_config.yaml policy file, which specifies what to collect, where to look, and what to exclude. It loads the case ID, scan paths, and filtering rules, initializes the SQLite evidence database, sets up output directories for packages, and validates all configuration parameters before proceeding.

Phase two is discovery. The Discovery Agent scans the specified filesystem paths recursively. It applies extension filters, so in our configuration we're looking for PDFs, Word documents, text files, and images. It checks file sizes against a maximum threshold, which defaults to one hundred megabytes but is configurable. It excludes system directories like venv and underscore-underscore-pycache, and enqueues discovered files for processing.

Phase three is processing, where the real forensic work happens. The Processing Agent reads files from the queue which calculates the SHA-256 hash for each file, extracts metadata including all three timestamps: created, modified, and accessed, as well as the file size and ownership information. It detects the actual file type using libmagic, and stores a complete evidence record in the database.

Phase four is packaging. The Packaging Agent exports evidence records to both CSV format for Excel and JSON format for programmatic access. It creates an encrypted ZIP archive that

includes the database, all reports, and manifests. It records chain of custody entries showing who collected the evidence, when it was collected, and what actions were performed. Finally, it generates a package integrity hash.

Phase five is verification. We perform a final hash calculation for the entire package, create a complete chain of custody audit trail, generate a summary report, record the collection timestamp, and ensure all actions are logged to our dfas.log file for later review.

SLIDE 6:

Now let me show you the actual execution results from our test run. When we executed DFAS, the system displayed this console output. You can see it loaded the case ID, which is a UUID to uniquely identify this collection. It completed discovery and found seventeen files matching our criteria. All seventeen files were successfully processed with no errors.

The execution took almost three seconds for seventeen files, which shows excellent performance. The system encountered no errors or failed file reads which showing the robustness of our error handling.

Let's look at the files that were generated. First, we have the evidence database file, which is a SQLite database containing all evidence records and the complete chain of custody audit trail. This database is queryable for forensic analysis.

Secondly, we have the CSV report, which is timestamped. This is an excel-compatible spreadsheet format containing all file metadata. It's suitable for non-technical stakeholders like attorneys or managers who need to review the evidence.

Third, we have the json report which is also timestamped. This is a machine-readable format for API integration and supports automated forensic tools. It includes the complete metadata structure.

Finally, we have the evidence package itself, which is a timestamped ZIP file. This is AES-GCM encrypted and contains the database and all reports. We calculated and recorded a SHA-256 hash for the entire package.

SLIDE 7:

The CSV report is particularly useful for legal proceedings and non-technical stakeholders. It contains all evidence record fields in a flat table structure. The columns include the file path, SHA-256 hash, file size, and all three timestamps. It also includes the owner, detected file type, and extension. Finally, it shows who collected the evidence and when. This format can be opened directly in Excel or imported into other forensic tools.

The JSON report helps with different needs. It provides a structured format for programmatic access, making it API-friendly for integration with other systems. It supports automated forensic workflows where other tools need to consume our data. It saves all metadata relationships in a hierarchical structure.

The SQLite database is the most powerful format for analysis. For example, you can easily query for all files modified within a specific time window, or all files owned by a particular user. It enables timeline analysis, which is important in forensic investigations. It stores the chain of custody in a separate table, maintaining a clear audit trail. And it can be analyzed with standard SQL tools that investigators are already familiar with.

SLIDE 8:

complete testing was important to make sure the reliability. We executed twelve tests covering all major components of the system. All twelve tests passed successfully with a one hundred percent success rate.

The test breakdown shows good coverage across all components. We have two tests for the database manager which verifying database initialization and record insertion. We have three tests for the processing agent, covering SHA-256 calculation, file type detection and metadata extraction. The discovery agent has three tests covering file discovery, extension filtering, and file size filtering. The packaging agent has three tests for CSV export, JSON export and package creation And we have one complete integration test that validates the complete workflow.

Our key validations include verifying SHA-256 accuracy against known test vectors, make sure the hash consistency across multiple calculations of the same file, testing file type detection with multiple formats including spoofed extensions which confirming database integrity with concurrent writes, testing large file filtering to ensure files over one hundred megabytes are excluded as configured and validating the complete end-to-end workflow from discovery through packaging.

SLIDE 9: development challenges Solved

During development, we solved different important challenges.

Cross-platform compatibility was our first challenge. Windows and Linux handle file permissions differently. We fixed this using pathlib, which works the same way on all systems. We also wrote special code for each operating system to detect file owners.

Locked files were another problem, especially on Windows where open files cannot be read. We added retry logic, so the system waits and tries again if a file is locked. We also made the system continue working even if one file fails, instead of stopping completely.

Performance with large datasets was a concern. Processing files one by one is too slow when dealing with thousands of files. We solved this with our queue-based system that processes multiple files at the same time.

File type spoofing is dangerous in forensics because attackers can rename files to hide evidence. We solved this using libmagic, which checks the actual file content instead of just the file name.

Memory usage with large files was our final challenge. Reading entire large files into memory can crash the system. We fixed this by reading files in small four-kilobyte pieces instead. This allows us to process very large files without using too much memory.

The key learning is that good error handling and backup plans are necessary for forensic tools that must work reliably in difficult situations.

SLIDE 10: Conclusion

Let me conclude by summarizing our achievements and discussing future work. We successfully created a fully functional multi-agent forensic collection system. The BDI architecture was successfully implemented with agents that reason about their environment and make decisions. All twelve unit tests pass with one hundred percent success rate, which shows the code reliability. We achieved cross platform compatibility across Windows, Linux and macOS.

My final statement is this: DFAS shows that intelligent agent principles can be effectively applied to solve complex real-world digital forensics challenges while maintaining strict accordance with international standards and court admissibility requirements. This project shows the practical value of agent oriented software engineering in a domain where reliability, auditability and legal compliance are absolutely important.

Thank you for your attention. I'm happy to answer any questions.