Security Architecture Whitepaper

Version 2.1 | January 2026

The First AI Memory System with Learned Executable Procedures

Dead Reckoning Foundry
security@deadreckoningfoundry.com

Executive Summary

AI agents are beginning to handle sensitive enterprise data. From technical documentation to business procedures, the information flowing through these systems demands protection that traditional application security cannot provide. The Automaton Memory System (AMS) was designed from day one with security as a foundational principle, not an afterthought.

The following sections detail how AMS protects sensitive data, ensures safe code execution, and maintains system integrity.

What Sets AMS Apart

AMS introduces a unique capability to the AI agent ecosystem: learned executable procedures called Automatons. While static skill frameworks provide instructions for what agents should do, AMS provides the execution layer that actually runs code, tracks success and failure, and improves over time through Bayesian learning. Four security capabilities protect this powerful functionality.

1. **Seven-Layer Input Validation:** Every piece of data entering AMS passes through seven distinct security checks before storage, including control character filtering, pattern analysis for injection attempts, and sensitive data detection.
2. **Sandboxed Code Execution:** Automatons execute real Python and Bash code in isolated Docker containers with network disabled, memory limits enforced, and filesystem access restricted.
3. **Memory Integrity Controls:** Memories connect through typed relationships with confidence scores, and version control through supersession tracking ensures nothing is lost and everything can be audited.
4. **Privacy by Design:** Automatic detection and optional redaction of personally identifiable information happens before storage, with privacy policies configurable for different deployment contexts.

Automatons: The Learned Execution Layer

AMS Automatons complement existing agent skill frameworks by providing what static instructions cannot provide on their own. They deliver actual code execution with performance tracking. Static skill definitions tell an agent what to do. Automatons provide the learned, executable procedures that accomplish the task and improve with each execution.

Instruction frameworks define what an agent can do. AMS provides the runtime that actually does it, tracking success rates with Bayesian statistics and surfacing the most reliable procedure for any given task.

1. AI Agent Memory Threat Landscape

AI agent memory systems face security challenges that traditional application frameworks were never designed to handle.

Research from MITRE ATLAS, academic institutions, and security practitioners has identified several attack vectors specific to these systems. The table below summarizes the most significant threats.

Threat Category	Description and Impact
Memory Poisoning	Attackers inject malicious content into agent memory, causing the agent to make incorrect decisions or leak sensitive information in future interactions.
Context Window Leakage	Prompt injection attacks extract historical conversations and stored data through carefully crafted queries that trick the agent into revealing protected information.
Tool Invocation Attacks	Agents with tool access can be manipulated to extract sensitive information through seemingly legitimate operations, exploiting the trust relationship between agent and tools.
Vector Database Attacks	Similarity searches can reveal training data patterns through membership inference and proximity analysis, allowing attackers to probe the embedding space.
Cross-Session Persistence	User data bleeds across different interactions due to improper session boundaries and memory isolation.
Procedural Code Injection	Malicious code inserted into learned procedures executes with elevated privileges during agent operations. This is particularly dangerous in systems that store and execute learned behaviors.

1.1 AMS Security Philosophy

AMS adopts a defense-in-depth strategy that assumes any single security control can fail. Multiple overlapping layers ensure that a breach at one level does not compromise the entire system.

Four principles guide this philosophy. Zero trust architecture means all inputs are treated as potentially malicious, regardless of source. Even data from trusted systems undergoes full validation. The principle of least privilege ensures automatons execute with only the permissions required for their specific task. A procedure that reads files cannot write to the network. Fail-secure defaults mean that when security validation fails, operations are blocked rather than allowed. Audit trail preservation ensures all security-relevant events are logged for forensic analysis. Nothing happens in the system without a record.

2. Seven-Layer Security Validation Pipeline

Every write operation in AMS passes through a rigorous seven-layer security pipeline before data reaches storage. The layers execute sequentially, and if any layer fails validation, the operation is immediately blocked with an appropriate error.

2.1 Input Sanitization

The first two layers clean and analyze incoming data. Layer 1 strips out control characters using regular expressions: null bytes, escape sequences, non-printable characters. Legitimate Unicode text passes through. Whitespace gets normalized to block encoding-based bypasses. The result is protection against SQL injection, command injection, log injection, and null byte attacks.

Layer 2 looks for patterns. It matches against known attack signatures including system prompt manipulation, role impersonation, and instruction overrides. Machine learning-based anomaly detection catches what the rules miss. Together, these two layers stop the most common attacks before they get any further.

2.2 Sensitive Data Detection

Layer 3 hunts for personally identifiable information. Regex patterns catch the usual suspects including Social Security Numbers, email addresses, phone numbers, credit cards, API keys, passwords, IP addresses, and AWS credentials. When something matches, configurable redaction swaps it for a masked placeholder. The semantic meaning stays intact, but the sensitive data never hits storage. Without this layer, a single careless memory write could create a compliance nightmare under GDPR, CCPA, or HIPAA.

2.3 External Link and Content Validation

Layer 4 prevents storage of links to known malicious domains or suspicious URLs. URLs are extracted and validated against blocklists of known malicious domains. The system checks for suspicious patterns including URL shorteners with obfuscated destinations, data URLs with embedded payloads, and internal network addresses that could enable Server-Side Request Forgery (SSRF) attacks.

Layer 5 enforces content length limits to prevent denial-of-service through oversized payloads. Configurable limits apply to memory content size (default 10MB), title length, tag count, and metadata payload size. These limits are enforced before expensive operations like embedding generation, preventing both resource exhaustion and cost amplification attacks on the embedding API.

2.4 Filesystem and Metadata Protection

Layer 6 locks down file operations for Obsidian sync. Every file path gets converted to canonical form and checked against the configured base directory. Attempts to escape using '../' or encoded variants get blocked. Symlink resolution closes that loophole too. No directory traversal, no arbitrary file access, no sneaking into configuration files.

Layer 7 handles metadata. JSON and YAML structures go through schema validation to ensure they match expected formats. Deep inspection catches nested injection payloads. Reserved fields like 'memory_id' and 'created_at' are protected from user overwrites. NoSQL injection, YAML deserialization attacks, schema pollution: none of it gets through.

The complete pipeline flow is: Input → Control Filter → Pattern Analysis → PII Detection → Link Validation → Size Limits → Path Check → Metadata → Storage.

3. Sandboxed Code Execution for Automatons

Automatons are executable procedures that the system learns from repeated successful interactions. They contain real code, so isolation is non-negotiable. AMS uses two sandboxing mechanisms. Docker containers serve as the primary method, and a restricted Python executor provides fallback capability.

3.1 Docker Isolation

The preferred execution environment uses Docker containers with strict security constraints:

- **Network Isolation:** Containers run with network disabled by default, preventing data exfiltration and external communication.
- **Resource Limits:** Memory (256MB), CPU (0.5 cores), and execution time (30 seconds) are strictly bounded.
- **Read-Only Filesystem:** Container filesystems are mounted read-only except for a temporary working directory.
- **No Privileged Operations:** Containers run as non-root users with dropped capabilities.
- **Ephemeral Containers:** Each execution creates a fresh container that is destroyed after completion, preventing state persistence attacks.

3.2 Restricted Python Fallback

When Docker is unavailable, a restricted Python executor provides secondary protection. Only safe built-in functions are available (len, str, int, list, etc.), while dangerous functions like eval, exec, open, and __import__ are blocked. Import statements are intercepted, with only pre-approved modules (json, math, datetime, re) permitted. Code is parsed via AST analysis before execution, and suspicious patterns like attribute access to __class__ or __globals__ trigger immediate rejection. Signal-based timeout enforcement kills long-running code to prevent infinite loops.

3.3 Bayesian Learning and Trust

Automatons track their execution history using Bayesian statistics. This provides security benefits beyond simple success/failure logging. New automatons start with uncertain confidence, and only after sufficient successful executions does an automaton earn high trust scores. Sudden drops in success rate can indicate tampering or environmental changes, triggering alerts. When multiple automatons could handle a task, the system suggests those with proven track records first.

Execution follows a strict protocol. Request validation verifies the automaton exists, the user has execution permission, and parameters match the expected schema. Sandbox selection then checks the automaton's requires_sandbox flag and selects Docker or restricted Python. Environment preparation creates an isolated execution environment with only necessary dependencies. Monitored execution runs the code with resource monitoring, capturing stdout, stderr, and return values. Result sanitization validates output and strips any leaked environment information. Cleanup then destroys the execution environment and updates Bayesian tracking.

4. Operational Security

AMS production deployments follow security best practices for infrastructure. PostgreSQL hardening includes connection pooling, SSL/TLS encryption, role-based access control, and prepared statements to prevent SQL injection. Redis security includes password authentication, TLS encryption, and binding to localhost to prevent unauthorized cache access. API authentication uses bearer tokens with configurable expiration and rotation policies. Database and cache services are not exposed to public networks; only the FastAPI application accepts external connections.

Vector embeddings require special consideration as they can leak information about training data. A 24-hour Redis cache reduces API calls to embedding providers, limiting exposure of content to external services. Hybrid search combines vector similarity with keyword matching, reducing reliance on embeddings alone and making inference attacks more difficult. Embedding queries are logged for audit purposes, but content is not retained after embedding generation. Bidirectional synchronization with Obsidian vaults introduces file system security considerations. All file operations are constrained to the configured vault directory through Layer 6 of the security pipeline. Hash-based change detection ensures only intentional modifications trigger synchronization, and the database is authoritative for conflict resolution.

5. Compliance and Privacy

AMS security controls align with major regulatory frameworks. The General Data Protection Regulation (GDPR) requires organizations to protect personal data of EU citizens. AMS supports GDPR through personally identifiable information (PII) detection and redaction in Layer 3, data export capabilities for subject access requests, soft delete functionality for right-to-erasure requests, and comprehensive audit logging for accountability demonstrations.

The California Consumer Privacy Act (CCPA) grants California residents rights over their personal information. AMS supports CCPA through data inventory capabilities via memory listing APIs, deletion APIs for consumer deletion requests, and access logging that tracks who accessed what data and when. SOC 2 Type II certification requires demonstrated security controls over an extended period. AMS supports SOC 2 through granular access controls, comprehensive audit trails, change management via the supersession system, and availability monitoring capabilities.

The Health Insurance Portability and Accountability Act (HIPAA) protects patient health information. AMS supports HIPAA through PHI detection in Layer 3, access controls, and audit logging. Full HIPAA compliance requires additional Business Associate Agreements and infrastructure controls beyond what AMS provides out of the box. The Open Web Application Security Project (OWASP) maintains a list of the most critical web application security risks. AMS addresses injection attacks through Layers 1 and 2, broken access control through Layer 6, security misconfiguration through hardened defaults, and cryptographic failures through TLS encryption on all connections.

5.1 Privacy-by-Design Principles

Privacy protections are built into the architecture, not bolted on later:

- **Data Minimization:** AMS stores only the data necessary for operation. The embedding cache expires automatically after 24 hours, temporary execution environments are

destroyed immediately after use, and the Curator service can be configured to archive or delete inactive memories based on retention policies.
- **Purpose Limitation:** Memories are tagged with their intended use through the metadata system. Agents can filter retrieved memories based on context, ensuring that information collected for one purpose is not inadvertently used for another.
- **Storage Limitation:** The Curator service runs nightly to identify and archive inactive memories based on configurable retention policies. Organizations can set different retention periods for different memory types.
- **Transparency:** The Obsidian sync feature provides a human-readable view of all stored memories as markdown files. All operations are logged to the audit trail, allowing organizations to demonstrate exactly what data was accessed, when, and by whom.

6. Memory Integrity and Access Control

AMS stores memories as a graph of interconnected knowledge, not isolated records. Powerful traversal capabilities come with their own security considerations.

Six relationship types (prerequisite, references, troubleshoots, extends, related_to, and part_of) have defined semantics and traversal rules. Relationships include confidence scores from 0.0 to 1.0 that affect traversal priority and prevent weak associations from propagating sensitive data. Graph queries enforce maximum depth and result count to prevent resource exhaustion.

6.1 Version Control and Supersession

Memory evolution is tracked through a supersession system that maintains integrity while allowing knowledge updates. When a memory is updated, the old version is marked as superseded rather than deleted, preserving audit history and preventing information loss. The set_canonical API designates authoritative memories for specific topics, ensuring agents retrieve verified information. Deleted memories are archived rather than permanently removed, allowing recovery and forensic analysis.

6.2 Access Pattern Monitoring

AMS tracks memory access patterns to detect anomalies and support security analysis:
- **Access Counting:** Each memory record tracks access_count and last_accessed timestamp.
- **Session Tracking:** Memory accesses are associated with agent sessions, enabling cross-session analysis.
- **Importance Decay:** Infrequently accessed memories have reduced importance scores, limiting their influence on agent behavior.

7. Future Development

The security roadmap for 2026 focuses on enterprise readiness. An observability dashboard will provide real-time visualization of embedding drift, access patterns, and anomaly detection. Multi-tenant isolation through row-level security in PostgreSQL will support enterprise deployments with multiple customers. Enhanced role-based access control will offer granular permissions for memory creation, execution, and administration. SOC 2 certification through formal audit is planned for enterprise customers.

Looking further ahead to 2027-2028, the roadmap includes trusted execution environments using hardware-backed isolation with Intel SGX or AMD SEV for highest-security deployments. Differential privacy will provide mathematical privacy guarantees for embedding operations in regulated industries. Federated learning will enable cross-organization automaton improvement without sharing raw data.

8. Conclusion

This whitepaper has examined the security architecture of the Automaton Memory System across seven key areas. These include the threat landscape facing AI agent memory systems, the seven-layer validation pipeline, sandboxed execution, operational security, compliance frameworks, memory integrity controls, and the development roadmap.

Most AI memory solutions bolt security on after the fact. AMS took the opposite approach, designing every capability with protection as the starting point, not a feature to add later.

AMS does not replace instruction-based skill systems. It provides the secure execution layer that turns those instructions into action. Organizations can adopt AMS alongside their existing agent infrastructure, gaining learned procedures and Bayesian optimization without abandoning current investments.

For more information about AMS security or to schedule a review, contact Dead Reckoning Foundry at security@deadreckoningfoundry.com

Appendix A: Security Validation Checklist

Organizations evaluating AMS can use this checklist to verify security controls:

Security Control	Implemented	Verified
7-Layer Input Validation Pipeline	✓	☐
Docker Sandbox for Automaton Execution	✓	☐
Restricted Python Fallback	✓	☐
PII Detection and Redaction	✓	☐
Path Traversal Protection	✓	☐
PostgreSQL TLS Encryption	✓	☐
Redis Password Authentication	✓	☐
API Token Authentication	✓	☐
Audit Logging Enabled	✓	☐
Memory Supersession Tracking	✓	☐
Bayesian Success Tracking	✓	☐