TECHNICAL FACT SHEET

SOFTWARE SECURED

Penetration Testing for AI-Integrated Applications

WHO THIS IS FOR Engineering leads, CTOs, and security teams shipping products that use LLMs, RAG pipelines, AI agents, or MCP-connected tooling.

01 LLM API Integrations

TESTING APPROACH

We treat the LLM as an untrusted component in the application stack, not as a black box. Testing focuses on how user input reaches the model, how model output is handled downstream, and whether the boundary between the application and the LLM provider is properly enforced. We probe prompt construction logic, test system prompt confidentiality, and evaluate whether the application trusts model output in ways it shouldn't.

SAMPLE VULNERABILITIES

- System prompt extraction via indirect prompt injection in user-supplied content
- Model response passed to a downstream function (e.g., code executor, SQL builder) without sanitization
- API key exposure through verbose error handling when provider calls fail
- Unbounded token consumption leading to denial-of-service via inflated inference costs
- Sensitive data (PII, internal instructions) reflected back in model completions due to poor context management

02 Retrieval-Augmented Generation (RAG)

TESTING APPROACH

RAG systems introduce two attack surfaces that LLM-only integrations don't have: the document ingestion pipeline and the vector retrieval layer. We test both. On ingestion, we evaluate whether malicious content embedded in source documents can influence model behavior at query time. On retrieval, we test whether users can manipulate queries to surface documents they shouldn't have access to, and whether the retrieval results respect tenant or role boundaries. We also evaluate how the application handles conflicting or adversarially crafted retrieved chunks.

SAMPLE VULNERABILITIES

- Prompt injection embedded in ingested documents (PDFs, emails, knowledge base articles) that hijacks model behavior when retrieved
- Cross-tenant document retrieval: user in Tenant A retrieves chunks belonging to Tenant B through crafted semantic queries
- Retrieval manipulation that forces the model to ignore its knowledge base and respond from injected context
- Metadata leakage: document filenames, author fields, or internal IDs exposed in cited sources
- No re-ranking or relevance threshold, allowing low-quality injected content to dominate retrieved context

03 Model Context Protocol (MCP)

TESTING APPROACH

MCP dramatically expands what an AI system can do - and therefore what it can be made to do. We test MCP servers as network-exposed services with their own attack surface, evaluate tool definitions for overly permissive capability declarations, and assess whether the host application enforces meaningful human-in-the-loop controls before executing sensitive tools. We also test for prompt injection paths that could cause an LLM client to invoke MCP tools without user intent.

SAMPLE VULNERABILITIES

- MCP server exposes tools (file write, shell exec, API calls) without enforcing caller authentication
- Tool descriptions crafted or manipulated to mislead the LLM into invoking higher-privilege operations
- Prompt injection in external content (emails, documents) causes the LLM to silently invoke destructive MCP tools
- Tool results returned to the LLM contain adversarial instructions that redirect subsequent tool use
- No rate limiting or scope restriction on MCP tool execution, enabling automated abuse at scale

04 AI Agents

TESTING APPROACH

Agents operate with greater autonomy, longer task horizons, and more real-world side effects than single-turn LLM calls. Testing focuses on whether agents can be diverted from their intended task, whether they enforce scope boundaries across multi-step workflows, and whether their actions are reversible when something goes wrong. We simulate adversarial inputs at each step in multi-agent pipelines and evaluate whether orchestrator-to-subagent trust is properly scoped.

SAMPLE VULNERABILITIES

- Task hijacking via adversarial content in the environment (a webpage, file, or API response the agent reads mid-task)
- Privilege escalation: subagent granted broader permissions than the initiating user holds
- Persistent memory poisoning - injecting false context into agent memory that influences future sessions
- Unbounded action loops triggered by malformed tool results, consuming credentials or budget without user awareness
- Orchestrator blindly trusts subagent output, allowing a compromised subagent to direct orchestrator behavior

05 AI-Assisted Developer Tooling & Platform-Generated Code

TESTING APPROACH

AI-generated code introduces risk at the point of deployment, not at the point of generation. We review codebases and infrastructure where AI tooling (Copilot, Claude Code, Cursor, platform scaffolding) has contributed material functionality, focusing on the vulnerability patterns that appear most frequently in LLM-authored code. We also test platforms that generate executable output - workflow builders, low-code tools, AI-authored scripts - treating generated artifacts as first-class attack surface.

SAMPLE VULNERABILITIES

- Generated authentication code that omits token expiry, scope validation, or revocation logic
- AI-scaffolded API endpoints with missing authorization checks (the model implements the happy path, not the security path)
- Over-permissioned IAM roles and cloud policies produced by AI infrastructure tooling
- Generated SQL or shell commands built from user input without parameterization
- Platform-generated code deployed without review, containing hardcoded credentials or insecure default configurations
- XSS and unsafe HTML rendering from AI-generated UI components that embed dynamic content without sanitization

About Software Secured

Software Secured is a premium manual penetration testing firm serving Series A/B SaaS companies. Every finding we deliver is confirmed exploitable. Zero false positives. Reports accepted by SOC 2 auditors and enterprise security teams.

GET IN TOUCH
softwaresecured.com
sales@softwaresecured.com