# LUCIFER

## Autonomous AI Red Team Platform

Engineering Master Specification | v2.0 | 10-Team Parallel Build Plan

*CONFIDENTIAL — FOR ENGINEERING LEADERSHIP USE ONLY*

# 1. Executive Overview & Vision

Lucifer is a startup-grade, production-quality autonomous AI red-team platform. It simulates real-world adversarial behavior against explicitly authorized targets, produces legally defensible forensic evidence, and generates audit- and compliance-ready reports — not just lists of CVEs.

Unlike conventional scanners or manual pen-testing workflows, Lucifer orchestrates a multi-brain AI system where every agent has its own dedicated LLM brain that reasons, plans, and executes independently within its domain. Each brain has its own system prompt, tool schema, reasoning loop, and structured memory. The orchestrator coordinates these brains without micromanaging them.

## 1.1 Core Differentiators

- Every agent has its own dedicated LLM brain — independent reasoning, not just script execution
- Black-box first: starts with zero internal knowledge, mimicking a real external attacker
- Evidence over assumptions: every finding backed by captured HTTP transcripts, screenshots, and reproduction steps
- Persistent global run journal: prevents context loss across long agentic runs
- Human-in-the-loop risk gates: destructive actions require explicit approval
- Compliance-native output: SOC 2, PCI-DSS, HIPAA, ISO 27001 mapped automatically
- Pluggable everything: LLM providers, execution runners, knowledge sources, tool modules

## 1.2 Target Use Cases

| Use Case | Actor | Value Delivered |
|---|---|---|
| Authorized Web App Pen Test | Security Engineer | Full attack chain with evidence, reproduction steps |
| API Security Assessment | Security Architect | Business logic flaws, auth bypass, injection chains |
| Cloud Surface Audit | Cloud Security Team | Misconfiguration chains with exploitability proof |
| Pre-Audit Compliance Check | CISO / Auditor | Control gap mapping to SOC 2 / PCI-DSS / HIPAA |
| Continuous Red Team Automation | DevSecOps | Regression testing on every deploy or weekly schedule |
| Executive Risk Briefing | CTO / CEO | One-page business impact summary with severity scores |

# 2. System Architecture

Lucifer is designed as a distributed, event-driven platform with a clear separation between the orchestration brain, domain-specific agent brains, tool execution layer, evidence storage, knowledge management, and the reporting/compliance engine.

## 2.1 High-Level Component Map

| Layer | Component | Responsibility |
|---|---|---|
| Intelligence | Orchestrator Brain (LLM) | Goal decomposition, agent coordination, risk gate enforcement |
| Intelligence | 10 Individual Agent Brains (LLM, domain-specific) | Each agent has its own LLM, system prompt, tool schema, and memory |
| Persistence | Global Run Journal (SQLite) | Append-only, queryable log of all decisions, actions, evidence refs |
| Persistence | Evidence Store (filesystem + DB) | Immutable HTTP transcripts, screenshots, logs, artifacts |
| Persistence | Per-Agent Memory Store (vector DB) | Each agent has its own structured memory and lessons-learned store |
| Knowledge | Knowledge Base (vector DB + metadata) | Global + project-specific documents, chunked, embedded |
| Execution | Tool Runtime | HTTP client, headless browser, MITM proxy, OAST server |
| Execution | Replay Harness | Deterministic reproduction of any captured attack chain |
| Interface | Web Dashboard | Live monitoring, approval panel, journal viewer, KB management |
| Output | Report Engine | Narrative + technical PDF, compliance mapping, executive summary |

## 2.2 Orchestrator Brain

The Orchestrator is the primary coordinating brain. It receives the operator-defined scope and objective, decomposes the goal into a prioritized task graph, instantiates and coordinates agent brains, enforces approval gates for high-risk actions, and synthesizes findings into a coherent attack narrative.

**Implementation:** Stateful LangGraph graph with full context injection per node call (current run state, journal excerpt, pending agent reports, tool availability). The orchestrator never executes tools directly — it issues directives to agent brains and waits for their structured reports.

## 2.3 Individual Agent AI Brain Specifications

Every agent in Lucifer has its own dedicated AI brain. This means each agent has: its own LLM instance (via LiteLLM), a domain-specific system prompt, a curated set of callable tools, an independent ReAct reasoning loop, a structured output schema for its reports, and its own per-agent memory store. No agent is a dumb script. Every agent thinks.

The architecture below specifies each of the 10 agent brains in full. Implementing this correctly is the most critical engineering task in the project.

### Agent 0: Orchestrator Brain

| ORCHESTRATOR BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet (primary) / GPT-4o (fallback) — highest reasoning quality required |
| **Model Role** | Coordinator, planner, and risk enforcer. Never executes tools directly. |
| **System Prompt Core** | You are the Orchestrator of an authorized red-team AI platform. Your job is to decompose the operator's objective into a prioritized task graph, delegate tasks to specialized agent brains, synthesize their findings into a coherent attack narrative, and enforce approval gates for high-risk actions. You operate with full knowledge of the current run state and all previous agent reports. You never assume — you reason from evidence. |
| **Input Context** | scope_json, current_task_graph, journal_excerpt (last 50 entries), active_findings[], pending_agent_reports[], available_agents[], kb_chunks (top 5 relevant) |
| **Output Schema** | { next_directive: { agent_type, task_description, context_to_pass, priority }, risk_assessment: str, approval_required: bool, synthesis_notes: str } |
| **Reasoning Loop** | LangGraph stateful graph: PLAN → DELEGATE → WAIT_FOR_REPORT → [APPROVAL_GATE if high-risk] → ANALYZE → [COMPLETE or loop back to PLAN] |
| **Memory** | Full run journal (queryable), orchestrator-level notes store in PostgreSQL |
| **Tool Access** | None — issues directives only. Has read access to journal and findings DB. |
| **Token Budget** | 4,000 tokens per orchestrator call. Summarize journal if approaching limit. |

### Agent 1: Recon Agent Brain

| RECON AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Haiku (speed-optimized — recon is high-volume, low-complexity reasoning) |
| **Model Role** | Passive and active reconnaissance. Builds the attack surface map the entire run depends on. |
| **System Prompt Core** | You are the Recon Agent for an authorized red-team engagement. Your job is to build the most complete possible picture of the target's attack surface |

| | |
|---|---|
| | using only passive and light-active techniques. Start with zero assumptions. Query DNS, certificate transparency logs, crawl sitemaps, and fingerprint technologies. Every asset you discover must be logged with source evidence. Do not interact with any asset outside the approved scope. |
| **Input Context** | scope_json (domains, IPs, URL prefixes), orchestrator_directive, relevant_kb_chunks (recon techniques), previous_recon_notes from memory |
| **Output Schema** | { surface_map: { endpoints[], subdomains[], technologies[], open_ports[], interesting_paths[], auth_endpoints[] }, confidence_scores: {}, evidence_refs: [], recommended_next_agents: [] } |
| **Reasoning Loop** | ReAct: Thought → Action (tool call) → Observation → Thought... until surface map is complete or max_steps reached |
| **Memory** | Per-agent ChromaDB collection: stores discovered assets, technology fingerprints, previous recon notes for the same target. Retrieved on every run against same target. |
| **Tools** | dns_lookup, cert_transparency_search, subdomain_enum (subfinder), http_get (light crawl), technology_fingerprint (Wappalyzer signatures), sitemap_fetch, robots_fetch, wayback_machine_query, shodan_query |
| **Max Steps** | 50 ReAct steps before forced surface map output |
| **Token Budget** | 2,000 tokens per ReAct step. Summarize observations after every 10 steps. |

## Agent 2: Web Agent Brain

| WEB AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — complex reasoning required for authentication flow analysis and session logic |
| **Model Role** | Deep web application interaction: authentication, session management, CSRF, business logic flaws, client-side vulnerabilities. |
| **System Prompt Core** | You are the Web Agent for an authorized red-team engagement. You interact deeply with web application surfaces using a real HTTP client and headless browser. Your focus is authentication weaknesses, session management flaws, CSRF, clickjacking, open redirects, account enumeration, and business logic vulnerabilities. Think like an attacker who has just found the login page. Chain your findings — a low-severity bypass may enable a critical privilege escalation. |
| **Input Context** | surface_map (from Recon Agent), orchestrator_directive, auth_endpoints[], form_inventory[], session_tokens[], relevant_kb_chunks, agent_memory |
| **Output Schema** | { findings: [{ title, description, severity, vuln_category, reproduction_steps[], evidence_refs[], cvss_vector }], session_artifacts: { cookies, tokens, headers }, recommended_chains: [] } |
| **Reasoning Loop** | ReAct with chain-of-thought: explicitly reasons about what each HTTP response reveals before next action. Maintains session state across steps. |
| **Memory** | Stores successful auth credentials, session token patterns, discovered roles, form structures. Reused across re-runs for regression testing. |

| | |
|---|---|
| **Tools** | http_request (full session), browser_navigate, browser_interact (form fill, click), screenshot_capture, cookie_analyzer, token_entropy_check, header_security_check, form_discovery |
| **Approval Gate** | Any action that successfully authenticates as another user triggers APPROVAL_REQUEST before further exploitation |
| **Max Steps** | 80 ReAct steps |
| **Token Budget** | 2,500 tokens per step |

## Agent 3: Injection Agent Brain

| INJECTION AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — injection chain reasoning requires multi-step attack planning |
| **Model Role** | Detects and chains all injection vulnerability classes: SQLi, XSS, SSTI, CMDi, Path Traversal, XXE, SSRF, deserialization. |
| **System Prompt Core** | You are the Injection Agent for an authorized red-team engagement. You are an expert in all injection vulnerability classes. For every input vector discovered, you reason about what data reaches the backend, what parsers or interpreters process it, and what payloads are most likely to succeed given the observed technology stack. You prioritize chaining: a reflected XSS combined with a CSRF token leak is more valuable than either alone. Always capture evidence before claiming a finding is confirmed. |
| **Input Context** | surface_map, endpoint_inventory[], parameter_inventory[], technology_stack, relevant_kb_chunks (PayloadsAllTheThings, HackTricks injection chapters), agent_memory |
| **Output Schema** | { findings: [{ title, description, severity, vuln_class, payload_used, injection_point, evidence_refs[], reproduction_steps[], oast_callback_ref }], tested_vectors: [], false_positive_notes: [] } |
| **Reasoning Loop** | ReAct with payload hypothesis: Thought (what payload class is most likely given stack?) → Action (inject) → Observation (parse response, check OAST) → Thought (confirmed/not, next hypothesis) |
| **Memory** | Stores working payloads per technology stack, false-positive signatures, previously confirmed injection points for regression. |
| **Tools** | http_request, oast_payload_gen, oast_poll_callbacks, payload_mutator, sqlmap_targeted (single URL mode only), xss_polyglot_gen, ssti_probe_gen, response_analyzer |
| **Approval Gate** | Confirmed SQLi with database read capability triggers APPROVAL_REQUEST before data extraction |
| **Max Steps** | 100 ReAct steps (injection requires high iteration volume) |
| **Token Budget** | 2,000 tokens per step |

## Agent 4: Auth Agent Brain

| AUTH AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — OAuth/SAML flow analysis requires strong specification reasoning |
| **Model Role** | Authentication and authorization architecture analysis: OAuth 2.0, SAML, JWT, IDOR, BOLA, BFLA, privilege escalation, multi-tenant isolation. |
| **System Prompt Core** | You are the Auth Agent for an authorized red-team engagement. You are a specialist in broken authentication and authorization. You analyze OAuth 2.0 flows for state parameter bypass, redirect_uri manipulation, and token leakage. You test JWTs for algorithm confusion, weak secrets, and none-algorithm attacks. You systematically test every resource endpoint for IDOR by substituting IDs across roles. You think horizontally (same role, different user) and vertically (lower role accessing higher privilege). |
| **Input Context** | auth_endpoints[], oauth_config (if discovered), jwt_samples[], user_roles (if known), api_endpoint_inventory[], surface_map, relevant_kb_chunks, session_artifacts (from Web Agent) |
| **Output Schema** | { findings: [{ title, description, severity, auth_class, affected_roles[], evidence_refs[], reproduction_steps[] }], role_map: {}, privilege_chains: [], jwt_analysis: {} } |
| **Reasoning Loop** | ReAct with state machine reasoning: models the authorization state machine and systematically probes every edge for missing or incorrect enforcement |
| **Memory** | Stores discovered roles, user IDs, resource ownership patterns, JWT algorithm configurations. Critical for chaining findings across runs. |
| **Tools** | http_request (multi-session, role-switching), jwt_decoder, jwt_attack_gen (alg:none, key confusion, brute secret), oauth_flow_analyzer, idor_probe (sequential + random ID substitution), role_comparison_diff |
| **Approval Gate** | Any successful privilege escalation to admin/root role triggers APPROVAL_REQUEST before further access |
| **Max Steps** | 80 ReAct steps |
| **Token Budget** | 2,500 tokens per step |

## Agent 5: API Agent Brain

| API AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — GraphQL schema analysis and REST API logic requires strong reasoning |
| **Model Role** | REST and GraphQL API security: schema extraction, BOLA, BFLA, mass assignment, rate limit bypass, excessive data exposure, API versioning attacks. |
| **System Prompt Core** | You are the API Agent for an authorized red-team engagement. You specialize in API security. You discover and extract API schemas (OpenAPI, Swagger, GraphQL introspection), then systematically test every endpoint and operation for authorization flaws, mass assignment, rate limit bypass, and excessive data exposure. You understand that APIs often have |

| | |
|---|---|
| | versioned endpoints (/v1, /v2, /api/legacy) with inconsistent security enforcement. You probe all versions. You also test for GraphQL-specific attacks: deep query complexity abuse, field suggestion exploitation, and batching attacks. |
| **Input Context** | api_endpoints[], swagger_spec (if found), graphql_schema (if found), auth_tokens[], surface_map, relevant_kb_chunks (OWASP API Top 10), agent_memory |
| **Output Schema** | { findings: [{ title, description, severity, api_class, endpoint, method, evidence_refs[], reproduction_steps[] }], schema_map: {}, exposed_fields: [], versioned_endpoints: [] } |
| **Reasoning Loop** | ReAct with schema-driven testing: loads schema, generates test matrix, executes systematically, analyzes responses for anomalies |
| **Memory** | Stores API schema snapshots, discovered endpoints, known-vulnerable patterns per API design pattern. Enables regression on schema changes. |
| **Tools** | http_request, openapi_parser, graphql_introspect, graphql_query_gen, mass_assignment_probe, rate_limit_bypass_gen (IP rotation, header manipulation), api_version_discovery, response_field_analyzer |
| **Approval Gate** | Confirmed bulk data extraction endpoint triggers APPROVAL_REQUEST before downloading records |
| **Max Steps** | 90 ReAct steps |
| **Token Budget** | 2,500 tokens per step |

## Agent 6: Cloud Agent Brain

| CLOUD AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — cloud misconfiguration chain reasoning requires architecture knowledge |
| **Model Role** | Cloud surface security: AWS/GCP/Azure metadata endpoints, S3/blob storage misconfigs, IAM credential chains, cloud-native service exposure. |
| **System Prompt Core** | You are the Cloud Agent for an authorized red-team engagement. You specialize in cloud infrastructure security. From a web application foothold, you probe for cloud metadata endpoints (AWS 169.254.169.254, GCP 169.254.169.254/computeMetadata, Azure 169.254.169.254/metadata), enumerate exposed storage buckets, and analyze discovered credentials for IAM privilege chains. You understand that cloud misconfigurations often chain: SSRF leads to metadata, metadata leaks IAM keys, IAM keys lead to S3, S3 leads to secrets, secrets lead to RDS. Think in chains. |
| **Input Context** | surface_map, cloud_indicators (from Recon Agent tech fingerprint), ssrf_findings (from Injection Agent), relevant_kb_chunks (AWS/GCP/Azure security guides), agent_memory |
| **Output Schema** | { findings: [{ title, description, severity, cloud_provider, service, evidence_refs[], reproduction_steps[], chain_description }], discovered_credentials: [], storage_exposure: [], iam_analysis: {} } |

| | |
|---|---|
| **Reasoning Loop** | ReAct with attack chain planning: explicitly models the cloud attack kill chain before executing each step. Chains findings from other agents (e.g., uses SSRF findings from Injection Agent). |
| **Memory** | Stores cloud provider indicators, credential patterns, discovered IAM configurations, bucket names. Enables targeted re-testing. |
| **Tools** | http_request (for metadata endpoint probing via SSRF chains), aws_s3_enum, gcp_storage_enum, azure_blob_enum, iam_permission_analyzer, metadata_endpoint_probe, cloud_cred_validator (checks if discovered keys are valid — read-only check only, no destructive actions) |
| **Approval Gate** | ANY use of discovered cloud credentials beyond validation requires APPROVAL_REQUEST |
| **Max Steps** | 60 ReAct steps |
| **Token Budget** | 2,500 tokens per step |

## Agent 7: Network Agent Brain

| NETWORK AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Haiku — network scanning is largely mechanical; light reasoning sufficient |
| **Model Role** | Network-level surface: port scanning, service fingerprinting, exposed admin interfaces, default credentials on network services. |
| **System Prompt Core** | You are the Network Agent for an authorized red-team engagement. You scan the network surface of the target for exposed services, fingerprint their versions, identify admin interfaces that should not be externally accessible (Elasticsearch, Redis, MongoDB, Kubernetes API, Jenkins, Grafana), and test for default or weak credentials. You operate within the approved IP ranges only. You prioritize services that directly enable lateral movement or data access. |
| **Input Context** | scope_json (IP ranges, domains), surface_map (from Recon Agent), relevant_kb_chunks (default credentials list, known CVEs for discovered services), agent_memory |
| **Output Schema** | { findings: [{ title, description, severity, service, port, evidence_refs[], reproduction_steps[] }], port_map: {}, service_inventory: [], default_cred_hits: [] } |
| **Reasoning Loop** | ReAct: Scan → Fingerprint → Identify interesting services → Test each for exposure/misconfiguration → Report |
| **Memory** | Stores port/service maps per target IP range. Enables detection of new exposed services in regression runs. |
| **Tools** | nmap_scan (python-nmap, rate-limited), service_fingerprint, default_cred_test (against discovered services), elasticsearch_probe, redis_probe, mongodb_probe, k8s_api_probe, admin_panel_discovery |

| | |
|---|---|
| **Approval Gate** | Successful authentication to any admin interface triggers APPROVAL_REQUEST before further access |
| **Max Steps** | 40 ReAct steps |
| **Token Budget** | 1,500 tokens per step |

## Agent 8: Evidence Agent Brain

| EVIDENCE AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — false positive analysis and impact reasoning requires strong judgment |
| **Model Role** | Cross-validates ALL findings from all other agents. Nothing is marked CONFIRMED without Evidence Agent approval. Triggers OAST verification, performs state diffs, and rules out false positives. |
| **System Prompt Core** | You are the Evidence Agent for an authorized red-team engagement. You are the final quality gate for all findings. You receive unconfirmed findings from other agents and must independently verify each one. For injection findings: replay the request and check OAST callbacks. For auth bypass: confirm access to restricted resources with a fresh session. For privilege escalation: perform a before/after role comparison and document the exact resource access differential. You are rigorous and skeptical. A finding without reproducible evidence does not exist. |
| **Input Context** | unconfirmed_findings[] (from all agents), raw_evidence_refs[], oast_callbacks[], agent_memory, run_journal (for context on how each finding was discovered) |
| **Output Schema** | { confirmed_findings: [{ finding_id, confirmation_method, evidence_refs[], state_diff }], rejected_findings: [{ finding_id, rejection_reason }], needs_retest: [{ finding_id, retest_instructions }] } |
| **Reasoning Loop** | For each unconfirmed finding: Thought (what is the minimal reproducible proof?) → Action (replay/retest) → Observation (confirmed or not?) → Decision (CONFIRM / REJECT / RETEST) |
| **Memory** | Stores confirmed finding signatures for deduplication across runs. Stores false-positive patterns to prevent recurrence. |
| **Tools** | http_replay (deterministic replay harness), oast_poll_callbacks, oast_verify_finding, state_diff_compare (pre/post access comparison), screenshot_capture, role_access_matrix_compare, http_request (fresh session for auth verification) |
| **Approval Gate** | Does not trigger approvals — Evidence Agent only validates, never exploits |
| **Max Steps** | 30 ReAct steps per finding |
| **Token Budget** | 2,500 tokens per step |

## Agent 9: Knowledge Agent Brain

| KNOWLEDGE AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Haiku — retrieval and summarization; does not need top-tier reasoning |
| **Model Role** | Serves as the knowledge interface for all other agents. Retrieves, synthesizes, and cites relevant offensive techniques, writeups, and target intelligence from the KB. |
| **System Prompt Core** | You are the Knowledge Agent. You do not execute attacks. You serve all other agents by retrieving the most relevant knowledge from the knowledge base for their current task. When an agent is about to test GraphQL, you retrieve GraphQL attack techniques. When an agent encounters a specific technology, you retrieve known vulnerabilities and exploitation paths for it. You always cite your sources with document ID and chunk ID. You synthesize retrieved chunks into actionable intelligence — not raw text dumps. |
| **Input Context** | query (from requesting agent), requesting_agent_type, current_target_technology_stack, project_kb_scope, global_kb_scope |
| **Output Schema** | { relevant_chunks: [{ doc_id, chunk_id, content, relevance_score }], synthesis: str (actionable summary for the requesting agent), suggested_payloads: [], related_cves: [] } |
| **Reasoning Loop** | Query → Hybrid Search (semantic + BM25) → Re-rank → Synthesize → Return (no multi-step ReAct needed) |
| **Memory** | Query cache (Redis) for frequently requested topics. Per-run retrieval log for citation tracking. |
| **Tools** | kb_semantic_search, kb_keyword_search, kb_rerank, kb_get_chunk_by_id, kb_list_documents |
| **Approval Gate** | None — Knowledge Agent is read-only |
| **Max Steps** | Single-pass — no iterative loop required |
| **Token Budget** | 3,000 tokens for synthesis output |

## Agent 10: Report Agent Brain

| REPORT AGENT BRAIN — Full Specification | |
|---|---|
| **LLM Model** | Claude 3.5 Sonnet — narrative quality and executive communication require top-tier writing capability |
| **Model Role** | Assembles all confirmed findings, evidence, and context into the structured content that feeds the PDF report engine. Writes attack narratives, remediation advice, and executive summaries. |
| **System Prompt Core** | You are the Report Agent for an authorized red-team engagement. You write the final report content. You have access to all confirmed findings, evidence references, the full run journal, and the compliance mapping rules. For each finding you write: a clear technical description, step-by-step reproduction instructions, a business impact statement (not just technical severity — explain what an attacker could do and what data or systems are at risk), and |

| | |
|---|---|
| | specific, actionable remediation guidance. The executive summary must be readable by a non-technical CEO in 2 minutes and make the risk concrete and urgent without being alarmist. |
| **Input Context** | confirmed_findings[] (from Evidence Agent), compliance_mappings[], run_journal (for attack narrative), evidence_refs[], target_profile, engagement_metadata |
| **Output Schema** | { executive_summary: str, attack_narrative: str, findings_content: [{ finding_id, technical_description, business_impact, reproduction_steps[], remediation_guidance, compliance_references[] }], asset_inventory: [], remediation_roadmap: [] } |
| **Reasoning Loop** | Linear generation pass (no ReAct needed): load all data → generate each section → cross-reference evidence → return structured report content |
| **Memory** | Stores report templates and tone calibration notes per engagement type. Learns from previous report feedback if provided. |
| **Tools** | journal_query, finding_fetch, evidence_metadata_fetch, compliance_rules_lookup, cvss_score_lookup, kb_query (for remediation best practices) |
| **Approval Gate** | None — Report Agent only reads data |
| **Max Steps** | Single-pass generation per section |
| **Token Budget** | 8,000 tokens for full report content generation (largest budget in the system) |

## 2.4 Agent Brain Summary Comparison

| Agent | LLM Model | Reasoning Pattern | Has Memory | Approval Gate | Max Steps |
|---|---|---|---|---|---|
| Orchestrator | Claude 3.5 Sonnet | LangGraph stateful graph | Yes (run-level) | N/A (issues gates) | Unbounded |
| Recon Agent | Claude 3.5 Haiku | ReAct | Yes (asset map) | No | 50 |
| Web Agent | Claude 3.5 Sonnet | ReAct + chain-of-thought | Yes (sessions, roles) | Auth bypass | 80 |
| Injection Agent | Claude 3.5 Sonnet | ReAct + payload hypothesis | Yes (payloads, FPs) | SQLi data read | 100 |
| Auth Agent | Claude 3.5 Sonnet | ReAct + state machine | Yes (roles, IDs) | Admin escalation | 80 |
| API Agent | Claude 3.5 Sonnet | ReAct + schema-driven | Yes (schemas, endpoints) | Bulk data exfil | 90 |
| Cloud Agent | Claude 3.5 Sonnet | ReAct + kill chain planning | Yes (credentials, IAM) | Any cred usage | 60 |
| Network Agent | Claude 3.5 Haiku | ReAct (mechanical) | Yes (port map) | Admin auth success | 40 |
| Evidence Agent | Claude 3.5 Sonnet | Per-finding verification loop | Yes (FP patterns) | None (read/verify) | 30 per finding |
| Knowledge Agent | Claude 3.5 Haiku | Single-pass retrieval | Yes (query cache) | None (read-only) | 1 pass |
| Report Agent | Claude 3.5 Sonnet | Single-pass generation | Yes (tone, templates) | None (read-only) | 1 pass per section |

## 2.5 Agent Brain Lifecycle

Each agent brain follows this standard lifecycle managed by the orchestrator:

1. SPAWN: Orchestrator instantiates agent with directive, context payload, and memory snapshot
2. INIT: Agent loads its system prompt, injects context, retrieves top-k KB chunks via Knowledge Agent
3. REASON: Agent enters its ReAct loop (or single-pass for Knowledge/Report agents)
4. EXECUTE: Agent calls tools via the Tool Runtime layer; all calls pass scope guard
5. JOURNAL: Every tool call, observation, and intermediate thought is appended to the global run journal

6. APPROVAL CHECK: Before any high-risk action, agent emits APPROVAL_REQUEST and blocks until decision arrives
7. REPORT: Agent produces structured output in its defined output schema
8. MEMORY UPDATE: Agent writes lessons learned and key artifacts to its per-agent memory store
9. TERMINATE: Agent signals completion; orchestrator reads report and updates task graph

## 2.6 Inter-Agent Context Passing

Agents do not call each other directly. The orchestrator mediates all inter-agent communication. When Agent A produces output that Agent B needs, the orchestrator:

10. Extracts the relevant fields from Agent A’s structured report
11. Adds them to Agent B’s context payload at spawn time
12. Records the dependency in the run journal

Example: Injection Agent discovers an SSRF endpoint. Orchestrator extracts the SSRF finding and passes it as ssrf_findings to the Cloud Agent’s context payload, enabling the Cloud Agent to probe metadata endpoints via the SSRF vector without needing to rediscover it.

## 2.7 Global Run Journal

Every action, decision, tool call, agent report, approval event, and evidence reference is appended to the global run journal. It is never overwritten. All agents read from and append to this journal.

```
journal_entry: { id, timestamp, run_id, agent_id, entry_type, payload,
evidence_refs[], parent_id }
entry_type: DECISION | TOOL_CALL | TOOL_RESULT | FINDING | APPROVAL_REQUEST |
APPROVAL_RESULT | MEMORY_UPDATE | ERROR | NOTE
```

## 2.8 Data Flow

13. Operator defines scope, target, mode, and objective via UI
14. Orchestrator Brain loads scope, retrieves relevant KB chunks, initializes task graph
15. Orchestrator spawns Recon Agent Brain with scope + KB context
16. Recon Agent Brain reasons, executes tools, builds surface map, writes to journal, reports back
17. Orchestrator analyzes surface map, spawns specialist agent brains per discovered surface
18. Each specialist agent brain reasons independently, uses tools, captures evidence, writes findings
19. High-risk actions trigger approval request; operator approves/rejects via dashboard
20. Evidence Agent Brain validates all findings independently with OAST and replay
21. Knowledge Agent Brain serves context to any requesting agent on demand
22. Report Agent Brain assembles all confirmed findings into structured report content
23. Report Engine renders final PDF with executive summary, technical findings, and evidence appendix

# 3. Technology Stack — Maximum Advantage Selection

Every technology choice below is made to maximize development velocity, production reliability, LLM integration quality, and long-term maintainability. Rationale is provided for every decision.

## 3.1 Backend Runtime

| Component | Technology | Rationale |
|---|---|---|
| Primary Language | Python 3.12+ | Best LLM SDK ecosystem (OpenAI, Anthropic, LangChain, LlamaIndex), rich security tooling (scapy, requests, httpx), mature async support |
| Async Framework | FastAPI + asyncio | High-performance async API, automatic OpenAPI docs, native Pydantic validation, WebSocket support for live dashboard |
| Task Queue | Celery + Redis | Distributed agent brain execution, priority queues, retry policies, result backend |
| Process Isolation | Docker + subprocess | Each agent brain runs in an isolated container to prevent cross-contamination |
| Config Management | Pydantic-Settings + .env | Type-safe config, environment variable injection, secrets management |

## 3.2 AI / LLM Layer

| Component | Technology | Rationale |
|---|---|---|
| LLM Abstraction | LiteLLM | Single interface for OpenAI, Anthropic, Ollama, Azure; enables per-agent model selection and fallback routing |
| High-Reasoning Model | Claude 3.5 Sonnet / GPT-4o | Used for Orchestrator, Web, Injection, Auth, API, Cloud, Evidence, Report agents — best tool-calling accuracy and long context |
| Fast/Cheap Model | Claude 3.5 Haiku | Used for Recon, Network, Knowledge agents — high-volume, lower-complexity reasoning |
| Local Model Fallback | Ollama + Llama 3.1 70B | Air-gapped runs, cost control, sensitive target data stays on-prem |
| Agent Framework | LangGraph | Stateful agent graphs, conditional edges, human-in-the-loop nodes, native streaming |
| RAG Pipeline | LlamaIndex + ChromaDB | Document ingestion, chunking, embedding, hybrid search (semantic + keyword) |

| Component | Technology | Rationale |
|---|---|---|
| Embeddings | text-embedding-3-small (OpenAI) / nomic-embed-text (local) | Fast, cheap, high-quality; nomic for air-gapped setups |
| Prompt Management | LangSmith / Promptfoo | Prompt versioning, evaluation, A/B testing of system prompts per agent |

## 3.3 Database & Storage

| Component | Technology | Rationale |
|---|---|---|
| Relational DB | PostgreSQL 16 | Run metadata, finding records, approval events, user management; ACID guarantees |
| Run Journal | SQLite (per-run file) | Portable, append-only, embedded — each run gets its own journal file |
| Vector Store (shared) | ChromaDB (local) / Qdrant (production) | KB embeddings; ChromaDB for local dev, Qdrant for production performance |
| Per-Agent Memory | ChromaDB collections (one per agent type per target) | Each agent brain has its own memory namespace; prevents cross-contamination |
| Evidence Store | Local filesystem + S3-compatible (MinIO/S3) | Immutable artifact storage; local for dev, MinIO or S3 for production |
| Cache / Queue | Redis 7 | Celery broker, result backend, rate-limiting, Knowledge Agent query cache |
| ORM | SQLAlchemy 2.0 + Alembic | Async ORM, migration management, schema versioning |

## 3.4 Tool Execution Layer

| Tool Module | Technology | Purpose |
|---|---|---|
| HTTP Client | HTTPX (async) + requests | Replayable, observable HTTP with full request/response capture, cookie jar management |
| Headless Browser | Playwright (async Python) | JS-heavy app interaction, screenshot capture, DOM inspection, auth flows |
| MITM Proxy | mitmproxy (Python API) | Transparent request/response interception, modification, and recording |
| Port Scanner | nmap (python-nmap) + masscan | Fast TCP/UDP scanning; nmap for service version detection |
| OAST Server | Interactsh (self-hosted) or canarytokens | Out-of-band callback verification for blind injection, SSRF, XXE |

| Tool Module | Technology | Purpose |
|---|---|---|
| DNS Recon | dnspython + subfinder + amass | Subdomain enumeration, DNS record extraction, zone transfer attempts |
| TLS Analysis | sslyze + cryptography | TLS version, cipher suite, certificate chain analysis |
| Fuzzer | Boofuzz + custom mutators | Protocol-level fuzzing for network services |
| Wordlists | SecLists (bundled) | Discovery wordlists for dirs, files, params, subdomains, passwords |
| Web Crawler | scrapy + Playwright | Spidering for endpoint discovery, form extraction, link analysis |
| JWT Toolkit | PyJWT + python-jose | JWT decode, algorithm confusion, key confusion attacks |
| GraphQL Tools | graphql-core + custom | Schema introspection, query complexity attacks, injection vectors |

## 3.5 Frontend / Dashboard

| Component | Technology | Rationale |
|---|---|---|
| Framework | React 18 + TypeScript | Industry standard, rich ecosystem, type safety |
| UI Components | shadcn/ui + Tailwind CSS | Accessible, customizable, dark-mode ready |
| Real-time Updates | WebSocket (FastAPI native) | Live journal streaming, finding pop-ups, agent status with model name |
| State Management | Zustand | Lightweight, no boilerplate, perfect for real-time agent status state |
| Charts & Viz | Recharts + D3 | Finding severity trends, attack surface graphs, agent activity timeline |
| Build Tool | Vite | Sub-second HMR, fast production builds |
| API Client | TanStack Query + Axios | Caching, background refetch, optimistic updates |

## 3.6 Reporting Engine

| Component | Technology | Rationale |
|---|---|---|
| PDF Generation | WeasyPrint + Jinja2 | HTML/CSS to PDF, full styling control, no headless browser dependency |
| Report Templates | Jinja2 HTML templates | Separate content from presentation; easy to theme per client |

| Component | Technology | Rationale |
|---|---|---|
| Compliance Mapping | Custom Python engine + YAML rules | Maps findings to SOC 2, PCI-DSS, HIPAA, ISO 27001 via configurable rules |
| CVSS Scoring | cvss (Python lib) | Automated CVSS 3.1 base score calculation from finding attributes |
| Narrative Generation | Report Agent Brain (Claude 3.5 Sonnet) | Attack narratives, executive summaries, and remediation guidance written by the Report Agent |

## 3.7 DevOps & Infrastructure

| Component | Technology | Rationale |
|---|---|---|
| Containerization | Docker + Docker Compose | Reproducible local dev, agent isolation, easy cloud migration |
| CI/CD | GitHub Actions | Automated test, lint, build, and container publish pipeline |
| Secret Management | HashiCorp Vault (prod) / .env (dev) | Centralized secrets rotation; .env for local development only |
| Monitoring | Prometheus + Grafana | Run metrics, per-agent LLM token usage, tool call rates, error rates |
| Logging | structlog + Loki | Structured JSON logs, queryable in Grafana |
| Testing | pytest + pytest-asyncio + Playwright | Unit, integration, and E2E test coverage |
| Code Quality | ruff + mypy + pre-commit | Fast linting, type checking, enforced on every commit |

# 4. Detailed Tool Module Specifications

## 4.1 HTTP Client Engine

The HTTP client is the core execution primitive. Every request must be replayable and observable.

- Library: HTTPX async with custom transport layer
- Features: full request/response capture (headers, body, timing), automatic cookie jar persistence, redirect chain tracking, TLS certificate capture, HTTP/2 support
- Evidence format: HAR (HTTP Archive) files per session, stored immutably in evidence store
- Replay: any captured HAR can be replayed deterministically via the replay harness
- Rate limiting: configurable per-target RPS to avoid DoS and IDS triggering

```
class HttpEngine:     async def request(self, method, url, **kwargs) -> HttpEvidence     async def replay(self, har_entry_id) -> HttpEvidence     async def session(self, target) -> HttpSession  # persistent cookie/auth session
```

## 4.2 Headless Browser Engine

Required for JavaScript-heavy SPAs, complex auth flows, and screenshot capture.

- Library: Playwright async Python
- Features: full page screenshots, network request interception, form interaction, OAuth/SSO flow automation, DOM snapshot extraction
- Context isolation: each agent brain run gets its own browser context
- Evidence: screenshots stored as PNG in evidence store with timestamp and URL metadata

```
class BrowserEngine:     async def navigate(self, url) -> PageSnapshot     async def interact(self, actions: List[BrowserAction]) -> PageSnapshot     async def screenshot(self, url) -> EvidenceRef
```

## 4.3 MITM Proxy / Recorder

Transparent proxy that sits between the tool layer and target, recording all traffic.

- Library: mitmproxy Python API (inline script mode)
- Features: request/response modification hooks, SSL interception with custom CA, WebSocket capture, automatic flow serialization

```
class MITMRecorder:     def start(self, port=8080) -> None     def get_flows(self, filter=None) -> List[Flow]     def export_har(self, session_id) -> HARFile
```

## 4.4 OAST / Callback Server

Out-of-band application security testing server for blind injection, SSRF, XXE, and DNS rebinding detection.

- Deploy self-hosted Interactsh server in the Lucifer infrastructure

- Each agent brain run gets a unique subdomain: {run_id}.{agent_id}.{oast_domain}
- Server records DNS, HTTP, SMTP, and SMB callbacks with full request context

```
class OASTServer:     def get_payload(self, run_id, finding_id) -> str  # unique callback URL     def poll_callbacks(self, run_id) -> List[OASTCallback]     def confirm_finding(self, finding_id) -> EvidenceRef
```

## 4.5 Knowledge Base Pipeline

24. Ingest: PDF, TXT, MD, URL → text extraction (pdfminer, requests, markdownify)
25. Chunk: recursive text splitter, 512 tokens per chunk, 50-token overlap
26. Embed: text-embedding-3-small or nomic-embed-text for local
27. Store: ChromaDB with metadata (source, scope: global/project, doc_id, chunk_id)
28. Retrieve: hybrid search (semantic cosine similarity + BM25 keyword), top-k=8
29. Cite: every KB usage logs doc_id + chunk_id in journal entry

# 5. Internal API Design

## 5.1 Core REST Endpoints

| Endpoint | Method | Description |
|---|---|---|
| /api/v1/runs | POST | Create new red team run with scope definition |
| /api/v1/runs/{id} | GET | Get run status, progress, active agents, and LLM cost summary |
| /api/v1/runs/{id}/journal | GET | Query run journal with filters (agent, type, time range) |
| /api/v1/runs/{id}/findings | GET | List all findings with severity, status, evidence refs |
| /api/v1/runs/{id}/approve | POST | Approve or reject a pending high-risk action |
| /api/v1/runs/{id}/agents | GET | List all agent brains for a run with status and model used |
| /api/v1/runs/{id}/report | POST | Trigger Report Agent and report engine for completed run |
| /api/v1/kb/documents | POST | Upload document to knowledge base |
| /api/v1/kb/documents | GET | List KB documents (global or project scope) |
| /api/v1/kb/documents/{id} | DELETE | Delete KB document and all associated chunks/embeddings |
| /api/v1/evidence/{ref_id} | GET | Retrieve specific evidence artifact |
| /api/v1/targets | POST/GET/DELETE | Manage authorized targets and scope definitions |
| /api/v1/reports/{run_id} | GET | Download generated PDF report |

## 5.2 WebSocket Endpoints

- /ws/runs/{id}/journal — streaming live journal entries, including agent reasoning steps
- /ws/runs/{id}/findings — real-time finding notifications with severity badge
- /ws/runs/{id}/approvals — approval request/response bidirectional channel
- /ws/runs/{id}/agent-status — live agent brain heartbeat, current ReAct step, token usage

# 6. Security, Legal & Compliance Architecture

## 6.1 Authorization Enforcement

- Scope definition: operator provides explicit CIDR ranges, domains, and URL prefixes at run creation
- Pre-execution guard: every tool call validates the target against approved scope before execution — no exceptions. This guard is called by the Tool Runtime layer, not by individual agent brains.
- Out-of-scope detection: if an agent discovers a redirect to an out-of-scope asset, it records it as an observation but does NOT follow or interact with it
- Audit trail: every scope check is logged in the journal

## 6.2 Human-in-the-Loop Approval Gates

| Action Category | Examples | Default Policy |
|---|---|---|
| Data Exfiltration Test | Downloading DB dump, exporting PII samples | REQUIRE APPROVAL |
| Auth Bypass Exploitation | Logging in as another user, admin takeover | REQUIRE APPROVAL |
| Destructive Testing | SQL DROP, file deletion, account lockout | REQUIRE APPROVAL + DOUBLE CONFIRM |
| High-Rate Fuzzing | >100 req/sec to single endpoint | REQUIRE APPROVAL |
| Cloud Credential Usage | Using found AWS keys to access S3/IAM | REQUIRE APPROVAL |
| Passive Recon | DNS lookup, certificate transparency | AUTOMATIC |
| Read-only HTTP requests | GET requests to discovered endpoints | AUTOMATIC |
| Screenshot capture | Visual evidence of finding | AUTOMATIC |

## 6.3 Compliance Mapping Engine

```
compliance_rules.yaml:   BROKEN_AUTH:     SOC2: [CC6.1, CC6.2, CC6.3]     PCI_DSS:
[Req 8.2, Req 8.3, Req 8.5]     HIPAA: [164.312(d), 164.312(a)(2)(i)]
ISO_27001: [A.9.2, A.9.4]
```

# 7. Database Schema

## 7.1 Core Tables (PostgreSQL)

```
runs: id, name, target_id, scope_json, mode, status, llm_cost_usd, created_at,
started_at, completed_at, operator_id
agents: id, run_id, agent_type, llm_model, status, started_at, completed_at,
step_count, token_usage, journal_entry_count
findings: id, run_id, title, description, severity, status, vuln_category,
cvss_score, evidence_refs[], compliance_mappings[], discovered_by_agent,
confirmed_by_agent, created_at
approval_events: id, run_id, finding_id, action_description, risk_level,
requested_by_agent, requested_at, decided_at, decision, operator_id, notes
targets: id, name, base_urls[], ip_ranges[], scope_domains[], owner_id,
authorization_doc_ref
evidence_artifacts: id, run_id, finding_id, artifact_type, storage_path, sha256,
metadata_json, created_at
kb_documents: id, name, scope, project_id, file_path, ingested_at, chunk_count,
metadata_json
agent_memory: id, agent_type, target_id, memory_type, content_json, created_at,
updated_at
```

## 7.2 Run Journal Schema (SQLite, per-run)

```
journal_entries: id, timestamp, agent_id, agent_type, entry_type, payload_json,
evidence_refs_json, parent_entry_id, tool_name, tool_duration_ms, llm_tokens_used
```

# 8. Knowledge Base Strategy

## 8.1 Recommended Seed Content

- OWASP Testing Guide v4.2 (full PDF)
- OWASP API Security Top 10
- HackTricks Web Pentesting (exported markdown)
- PortSwigger Web Security Academy writeups
- PayloadsAllTheThings (GitHub markdown export)
- Bug bounty writeups from HackerOne Hacktivity (curated selection)
- MITRE ATT&CK for Enterprise (web surfaces)
- AWS/GCP/Azure Security Best Practices
- GraphQL security research papers
- JWT attack vector catalog

## 8.2 Project-Specific Knowledge

- Technology stack documentation
- Previous pen test reports (for regression context)
- Architecture diagrams (gray-box or white-box engagements)
- API documentation (Swagger/OpenAPI spec, Postman collections)
- Known user accounts and roles (if provided by client)

# 9. Report Generation Specification

## 9.1 Report Structure

| Section | Audience | Content |
|---|---|---|
| Cover Page | All | Target name, engagement date, classification, operator name |
| Executive Summary | C-Suite, CISO | 2-page narrative: what was found, business impact, overall risk rating, top 3 priorities. Written by Report Agent Brain. |
| Attack Narrative | Security Team, Management | Chronological story of the attack chain in attacker-perspective prose. Written by Report Agent Brain from run journal. |
| Technical Findings | Engineers, Auditors | Full details per finding: description, reproduction steps, evidence, CVSS score, discovering agent, confirming agent, remediation |
| Evidence Appendix | Engineers, Auditors | Embedded or linked HTTP transcripts, screenshots, OAST callback logs |
| Asset Inventory | Security Team | All discovered assets within scope with classification |
| Compliance Mapping | Auditors, CISO | Control-by-control mapping with pass/fail/partial status and evidence references |
| Remediation Roadmap | Engineering, Management | Prioritized remediation plan with effort estimates and quick wins |

## 9.2 Severity Classification

| Severity | CVSS Range | Business Impact | SLA to Remediate |
|---|---|---|---|
| Critical | 9.0 - 10.0 | Full system compromise, data breach, complete business impact | 24 hours |
| High | 7.0 - 8.9 | Significant data exposure, partial compromise, auth bypass | 7 days |
| Medium | 4.0 - 6.9 | Limited data exposure, requires user interaction, defense-in-depth gap | 30 days |
| Low | 0.1 - 3.9 | Minimal direct impact, informational value, hardening opportunity | 90 days |
| Informational | 0.0 | No direct risk, best practice deviation, observation | Next sprint |

# 10. Project Execution Phases — 10 Parallel Engineering Teams

The project is decomposed into 6 phases. Teams are designed so their work streams are maximally independent, enabling true parallel execution with minimal blocking dependencies.

## Team Assignments Overview

| Team | Name | Domain | Size |
|---|---|---|---|
| Team 1 | Platform Core | Infrastructure, DB, shared services, auth, CI/CD, scope enforcement | 3-4 engineers |
| Team 2 | Orchestrator & Agent Framework | LangGraph orchestrator, agent brain lifecycle, message bus, LiteLLM abstraction | 3-4 engineers |
| Team 3 | Tool Execution Engine | HTTP client, browser, MITM proxy, OAST, replay harness, scope guard integration | 3-4 engineers |
| Team 4 | Web, Injection & API Agent Brains | Web Agent, Injection Agent, API Agent — full brain implementation | 3-4 engineers |
| Team 5 | Auth, Cloud & Network Agent Brains | Auth Agent, Cloud Agent, Network Agent — full brain implementation | 2-3 engineers |
| Team 6 | Knowledge & Memory Layer | KB ingestion pipeline, vector store, per-agent memory, Knowledge Agent Brain | 2-3 engineers |
| Team 7 | Evidence Agent & Compliance Engine | Evidence Agent Brain, evidence store, OAST integration, compliance mapping, CVSS | 2-3 engineers |
| Team 8 | Report Agent & Reporting Engine | Report Agent Brain, PDF generation, templates, narrative generation | 2-3 engineers |
| Team 9 | Web Dashboard & UI | React dashboard, WebSocket integration, approval panel, agent status panel, KB UI | 3-4 engineers |
| Team 10 | QA, Security & DevOps | Testing, CI/CD, containerization, monitoring, security hardening, operator manual | 2-3 engineers |

## Phase 1: Foundation (Weeks 1–3)

Establish the shared infrastructure that all other teams depend on.

### Team 1 — Platform Core (Lead)

- Set up monorepo: backend/, frontend/, agents/, tools/, reports/, infra/
- Docker Compose: PostgreSQL, Redis, ChromaDB, MinIO, Interactsh
- Implement database schema with Alembic migrations for all core tables including agents and agent_memory
- FastAPI skeleton with auth (JWT + API key), CORS, error handling middleware
- Implement scope enforcement guard as a standalone callable module — this is shared by ALL tool modules
- CRUD endpoints for Run, Target, Finding, Evidence, KBDocument, Agent (stub responses)
- Celery task queue structure; Prometheus metrics middleware

### Team 2 — Orchestrator Framework (Start Early)

- Define AgentBrain base class interface: system_prompt, input_schema, output_schema, tools[], memory_namespace, max_steps, token_budget
- Set up LiteLLM provider abstraction with per-agent model configuration in .env
- Define AgentDirective and AgentReport Pydantic models

### Team 10 — DevOps

- GitHub Actions: lint (ruff), type check (mypy), unit test, Docker build
- Pre-commit hooks; base Docker images for backend and agent containers
- Test fixture factories for all DB models

### Phase 1 Exit Criteria

- All services start cleanly with docker-compose up
- Scope enforcement guard unit-tested at 100%
- AgentBrain base class interface agreed upon and documented by Team 2
- CI pipeline passes on main branch

---

## Phase 2: Core Execution Engine + Agent Framework (Weeks 2–5)

Build the tool execution primitives and the agent brain framework. Teams 2, 3, and 6 work in parallel.

### Team 2 — Orchestrator & Agent Framework

- Implement LangGraph orchestrator graph: PLAN → DELEGATE → WAIT_FOR_REPORT → APPROVAL_GATE → ANALYZE → COMPLETE
- Build full agent brain lifecycle: spawn (with context injection), ReAct loop engine, journal write integration, memory load/save, graceful termination
- Implement agent registry and factory pattern
- Approval gate node: blocks graph, emits WebSocket event, waits for operator decision via polling
- Orchestrator context manager: assembles full context payload for each agent brain spawn
- Token budget enforcement per agent brain: hard stop at budget, forced output generation
- Integration tests with mock LLM responses for all graph paths

### Team 3 — Tool Execution Engine

- HttpEngine: HTTPX async, HAR capture, cookie jar, rate limiting — all calls route through scope guard
- BrowserEngine: Playwright navigate, interact, screenshot, network interception
- MITMRecorder: mitmproxy inline, flow capture, HAR export
- OASTClient: unique payload per agent/finding, poll, return EvidenceRef
- Replay harness: load HAR entry, replay deterministically, compare response
- Tool registry: all tools registered with metadata (name, description, input schema) for LLM tool-calling

### Team 6 — Knowledge & Memory Layer

- Document ingestion pipeline: PDF, TXT, MD, URL
- ChromaDB setup: global_kb, per-project collections, per-agent memory collections
- Hybrid retriever: semantic + BM25, RRF merge, citation output
- Knowledge Agent Brain: system prompt, single-pass retrieval loop, structured output schema
- Per-agent memory interface: read/write lessons learned, asset maps, credential patterns
- Seed global KB with OWASP Testing Guide and PayloadsAllTheThings

### Phase 2 Exit Criteria

- HttpEngine executes full auth session and exports valid HAR
- Agent brain lifecycle: spawn → ReAct loop (mock tools) → structured report → memory save → terminate
- Knowledge Agent Brain returns cited KB chunks for a test query
- OAST server receives a test DNS callback

## Phase 3: All Domain Agent Brains (Weeks 4–8)

Implement all 8 domain agent brains. Teams 4, 5, 7, and 8 work in parallel.

### Team 4 — Web, Injection & API Agent Brains

- Recon Agent Brain: full system prompt, ReAct loop, all recon tools, surface map output schema, memory integration
- Web Agent Brain: full system prompt, session-aware ReAct loop, authentication testing logic, approval gate integration for auth bypass
- Injection Agent Brain: full system prompt, payload hypothesis ReAct pattern, OAST integration, all injection classes, approval gate for SQLi data read
- API Agent Brain: full system prompt, schema-driven testing loop, GraphQL + REST coverage, approval gate for bulk data
- Each brain: tested end-to-end against DVWA or Juice Shop endpoints in isolation

### Team 5 — Auth, Cloud & Network Agent Brains

- Auth Agent Brain: full system prompt, state machine ReAct pattern, OAuth/SAML/JWT tools, IDOR probe, approval gate for admin escalation
- Cloud Agent Brain: full system prompt, kill chain planning ReAct, metadata probe, S3/IAM tools, approval gate for credential use
- Network Agent Brain: full system prompt, mechanical scan ReAct, nmap integration, default cred tests, approval gate for admin auth
- Evidence Agent Brain: full system prompt, per-finding verification loop, replay harness, OAST verify, state diff compare, CONFIRM/REJECT/RETEST output

### Team 7 — Evidence & Compliance Engine

- Evidence Store abstraction: local + S3, content-addressed (SHA256), immutable writes
- Artifact indexing: by run_id, finding_id, agent_id, artifact_type
- Compliance mapping engine: YAML rules for SOC 2, PCI-DSS, HIPAA, ISO 27001
- CVSS 3.1 auto-scorer: maps finding attributes to CVSS vector string and score
- Finding deduplication: semantic similarity + URL fingerprinting

### Team 8 — Report Agent Brain & Reporting Engine

- Report Agent Brain: full system prompt, linear generation pass, all section outputs, evidence cross-referencing
- Jinja2 HTML report templates: all sections with severity color coding and branding
- WeasyPrint PDF renderer with custom CSS
- Evidence embedding: inline screenshots, truncated HTTP transcripts with full links
- Test report generation with synthetic finding set covering all severities

### Phase 3 Exit Criteria

- All 10 agent brains (including Orchestrator) pass unit tests with mock LLM and mock tools
- Recon Agent Brain produces surface map on DVWA
- Injection Agent Brain detects SQLi and XSS on DVWA with Evidence Agent confirmation
- Evidence Agent Brain confirms a finding via OAST callback and state diff
- Report Agent Brain produces all report sections for a synthetic finding set
- PDF renders correctly with embedded evidence

---

## Phase 4: Orchestrated End-to-End Runs (Weeks 7–10)

Wire all agent brains into the orchestrator. Teams 2, 4, 5, and 9 converge.

### Team 2 — Full Orchestration Integration

- Integrate all 10 agent brains into orchestrator graph as callable nodes
- Dynamic task graph: orchestrator LLM call produces prioritized agent spawn plan from surface map
- Conflict resolution: contradictory evidence from two agents escalated to orchestrator for reasoning
- Run state persistence: full state serializable to PostgreSQL for crash recovery
- Token budget and LLM cost tracking: per-agent, per-run, reported in dashboard and final report
- Full E2E test on DVWA: recon → agent dispatch → findings → approval gate → evidence → report

### Team 9 — Web Dashboard

- React app with all routes: Dashboard, Runs, Run Detail, Findings, KB Management, Settings
- WebSocket: live journal stream (with agent name and model), finding notifications, agent status with step count and token usage
- Approval panel: pending card with agent name, action description, risk level, Approve/Reject + notes
- Findings browser: sortable table, finding detail modal with evidence viewer (HAR, screenshots)
- Agent status panel: shows all agent brains for active run with status, model, step count, token budget bar
- KB management: upload, list, delete, search preview, scope selector

### Phase 4 Exit Criteria

- Full E2E run on DVWA completes autonomously with all agent brains executing
- All approval gates fire and block correctly via UI
- Dashboard shows live agent brain status, step counts, and token usage
- PDF report passes review by a certified penetration tester

---

## Phase 5: Hardening & Production Readiness (Weeks 9–12)

### Team 1 — Platform Hardening

- Full RBAC: admin, analyst, viewer roles with per-target permissions
- API rate limiting and request validation hardening
- Run resource limits: max token budget, max duration, max concurrent agent brains

### Team 2 — Orchestrator Hardening

- Agent self-correction: 3x error on same agent → escalate to human or skip with documented rationale
- Context window management: intelligent journal summarization when approaching limit
- LLM cost report: per-run cost breakdown by agent brain in final report appendix

### Teams 4 & 5 — Agent Brain Hardening

- Per-agent timeout and error recovery: brain failure does not abort entire run
- False-positive filtering: second-pass LLM validation call in each agent before writing finding
- Expand payload libraries and wordlists for all agent brains

### Team 10 — QA & DevOps

- 80%+ unit test coverage across backend, agent brains, and tool modules
- Integration tests for all critical paths: full run lifecycle, approval gate, report generation
- E2E tests with Playwright: create run, live journal, approve action, download report
- Prometheus + Grafana: run duration, per-agent token usage, tool call rate, error rate
- Internal security review: hardcoded secrets, IDOR in run access, SQL injection in own API
- Production deployment guide and operator manual

---

## Phase 6: Cloud Scale & Advanced Features (Weeks 12+)

Post-MVP features prioritized by customer feedback.

- Cloud Runner: AWS ECS/Fargate or GCP Cloud Run for agent brain execution
- Scheduled Runs: cron-based continuous red teaming with delta reports
- Integrations: Jira/Linear finding export, Slack notifications, Splunk/Elastic log forwarding
- Custom Agent Brain Plugins: operator-defined brains via plugin SDK with defined interface
- Multi-target Campaigns: orchestrate runs across multiple targets simultaneously
- AI-driven Scope Expansion: orchestrator suggests additional targets from infrastructure discoveries

# 11. Critical Path & Dependency Map

| Deliverable | Produced By | Required By | Phase |
|---|---|---|---|
| Docker Compose + DB schema + scope guard | Team 1 | ALL teams | 1 |
| AgentBrain base class interface | Team 2 | Teams 4, 5, 6, 7, 8 | 1 |
| LiteLLM provider abstraction | Team 2 | ALL agent brain teams | 2 |
| HttpEngine + BrowserEngine + MITM | Team 3 | Teams 4, 5 | 2 |
| OAST server + OASTClient | Team 3 | Teams 4, 5, 7 | 2 |
| LangGraph orchestrator base + lifecycle | Team 2 | Teams 4, 5 (integration) | 2 |
| KB ingestion + ChromaDB + retriever | Team 6 | Teams 4, 5 (via Knowledge Agent) | 2 |
| Per-agent memory interface | Team 6 | Teams 4, 5, 7, 8 | 2 |
| Knowledge Agent Brain | Team 6 | ALL domain agent brains | 2 |
| All domain agent brains (Teams 4 & 5) | Teams 4, 5 | Team 2 (orchestration) | 3 |
| Evidence Agent Brain | Team 5 | Teams 2, 7 | 3 |
| Evidence store + compliance engine | Team 7 | Teams 8, 9 | 3 |
| Report Agent Brain + PDF engine | Team 8 | Team 9 (download), E2E tests | 4 |
| Full orchestration integration | Team 2 | E2E tests, Phase 4 | 4 |
| React dashboard + WebSocket + approval UI | Team 9 | E2E tests, operator review | 4 |
| Full test coverage + production hardening | All + Team 10 | Phase 5 release | 5 |

# 12. Engineering Risk Register

| Risk | Probability | Impact | Mitigation |
|---|---|---|---|
| LLM latency/cost makes runs too slow/expensive | High | High | Token budgets per agent, Haiku for low-complexity agents, Ollama fallback, cache Knowledge Agent queries |
| LLM hallucinations produce false findings | High | High | Mandatory Evidence Agent brain validates ALL findings; nothing confirmed without OAST or replay proof |
| Agent brain gets stuck in infinite ReAct loop | Medium | High | Hard max_steps limit per brain, dead-letter queue for stuck runs, orchestrator timeout watchdog |
| Scope enforcement bypass | Low | Critical | Scope guard in Tool Runtime layer (not in agent brains) — agents cannot bypass it. Defense-in-depth + audit. |
| Context window exhaustion on long runs | Medium | High | Journal summarization, per-brain context budgets, sliding window. Orchestrator manages context actively. |
| Per-agent memory store corruption/mixing | Low | High | Namespaced ChromaDB collections per agent_type+target_id. No shared write access between brains. |
| Agent brain produces output that does not match output schema | Medium | Medium | Pydantic output validation on every agent report. Retry with correction prompt up to 3 times. |
| LLM provider outage (OpenAI/Anthropic) | Low | High | LiteLLM fallback routing: Sonnet → GPT-4o → Ollama. Run pauses gracefully, not aborts. |
| Report Agent Brain produces inaccurate remediation | Medium | Medium | Human reviewer sign-off before report delivery. Report marked as AI-assisted. |

# 13. Definition of Done — MVP

## 13.1 Functional Completeness

- Operator creates target, defines scope, starts run via UI in under 5 minutes
- All 10 agent brains spawn, execute, write to journal, and terminate cleanly in a full run
- Platform autonomously finds all intentional vulnerabilities in DVWA (Damn Vulnerable Web App)
- Platform autonomously finds top 10 vulnerabilities in OWASP Juice Shop
- Evidence Agent Brain confirms at least 80% of findings with OAST or replay evidence
- Report Agent Brain produces a full PDF report that passes review by a certified penetration tester
- Compliance mapping correctly maps all finding categories to SOC 2, PCI-DSS, HIPAA, ISO 27001
- Approval gate reliably blocks and resumes runs on operator decision
- Knowledge base measurably improves agent finding quality vs. no-KB baseline

## 13.2 Quality & Security

- Zero known security vulnerabilities in Lucifer's own codebase
- Scope enforcement guard has 100% pass rate including adversarial inputs
- All API endpoints require authentication; no unauthenticated access to run data
- All agent brain output schemas validated with Pydantic on every response
- 80%+ unit and integration test coverage

## 13.3 Operational Readiness

- Platform runs stably for 8+ hours with all agent brains active, no memory leaks or crashes
- LLM cost per full run documented and within acceptable threshold
- Operator manual covers all core workflows including KB management and approval handling
- Docker Compose deployment reproducible from clean machine in under 15 minutes
- Monitoring dashboards show per-agent token usage, error rates, and run health

**END OF DOCUMENT**

*Lucifer Engineering Master Spec v2.0 | Classification: CONFIDENTIAL*