Threat Model and Trust-Boundary Architecture for an Agentic Software Platform

Executive Synthesis and Primary Objective

The deployment of the Agent Platform necessitates a fundamental re-evaluation of classical application security paradigms. The integration of autonomous agents, non-deterministic language models, and high-privilege external tooling creates an architectural posture where intent and data are intrinsically conflated. The primary research objective is to determine the precise assets, actors, trust zones, attack paths, abuse cases, security invariants, and defence controls required to safely process untrusted model output, repository content, websites, memory, tool responses, provider APIs, and multi-project workloads.

The analysis indicates that standard perimeter defences are radically insufficient for the Agent Operating System. The target system introduces ambient authority across multi-hop trust chains, meaning a compromised context document can directly manipulate execution intent unless rigid, deterministic boundaries are enforced.[1] The architecture must assume that all model output is potentially malicious, either through direct hallucination or indirect prompt injection via poisoned memory and retrieved content.[2] Consequently, the overarching security invariant mandates that a language model may propose a tool intent, but it must never directly authorise or execute consequential actions without deterministic validation, schema enforcement, and, where applicable, cryptographic human approval.[4]

Analytical Framework and Evaluation

To capture the unique vulnerabilities of the agentic development platform, a layered analytical framework is required. Classical frameworks designed for static software fail to capture non-determinism, temporal memory persistence, and autonomous reasoning collapse.[5] The recommended layered approach sequences multiple methodologies to ensure comprehensive coverage.

System-context modelling and data-flow diagrams are appropriate for design-time reviews. They identify structural weaknesses and map the exact points where authority crosses a trust boundary. However, they miss emergent autonomous behaviors. STRIDE provides a rigorous baseline for spoofing, tampering, and repudiation at the protocol layer, while LINDDUN privacy analysis effectively identifies data exposure, cross-project linkage, and unconsented data flows, particularly critical when managing multi-tenant project repositories.[7]

To address AI-specific risks, the framework incorporates the OWASP Multi-Agentic System (MAS) Threat Modeling Guide, which explicitly evaluates non-determinism, autonomy, and agent identity management.[5] This is augmented by the Multi-Agent Security (MASEC)

framework to model advanced emergent failures, including reasoning collapse across planner-executor chains, metric overfitting, unsafe delegation escalation, and goal-driven misreporting by agents acting within valid roles.[6] Attack-path analysis and abuse cases bridge the gap between static controls and operational reality, simulating full kill-chains from prompt injection to credential exfiltration. Software-supply-chain threat modelling maps the ingestion of untrusted dependencies via automated agents.
The layered sequence operates as follows: System context → assets and actors → trust boundaries → data flows → STRIDE and privacy review → agent-specific abuse cases → attack trees → critical attack paths → control mapping → adversarial validation → residual-risk decision.

Validated Security Principles

The foundational security principles of the target system have been rigorously validated against primary research, observed adversarial patterns, and architectural constraints. Acceptance of these principles forms the baseline for the target system's security invariants.

Principle	Supporting Evidence	Threat Addressed	Limitations	Enforcement Layer	Detection Method	Failure Mode	Release Stage	Invalidation
Model output is always untrusted input.	Models conflate instruction and data.[2]	Direct/Indirect Prompt Injection.	Models must parse data to function.	Context Compiler & Policy Engine.	Schema anomaly detection.	Untrusted output executed directly.	Core Foundation	Never.
Retrieved content cannot inherit authority.	Attackers embed directives in external files.[1]	Confused Deputy.	Restricts agent's ability to automate.	Tool Gateway & Content Parser.	Static metadata analysis.[2]	Data acts as executable intent.	Core Foundation	Never.
Repository	Malicious	Source-Cod	High sanitis	Context	Injection	Repo code	Implementat	Never.

files are untrusted content.	instructions exist in READMEs.[1]	e Prompt Injection.	ation overhead.	Compiler.	signature matching.	subverts agent logic.	ion	
Tool results may contain prompt injection.	Search outputs contain attacker text.[4]	Tool Poisoning.[2]	Validation breaks legitimate tools.	Tool Gateway.	Output sanitisation monitoring.	Recursive malicious delegation.	Implementation	Never.
Memory content retains provenance/scope.	Stored knowledge causes persistent poisoning.[9]	Memory Poisoning.	High storage/latency overhead.	Memory Service.	Contradiction detection algorithms.	False facts become permanent.	Implementation	Never.
Models propose, but do not authorise, intent.	Protocol architectures decouple intent from execution.[1]	Unauthorised Side Effects.	Slows autonomous execution.	Policy & Approval Engine.	Unsigned intent rejection logs.	Autonomous unauthorised mutation.	Core Foundation	Never.
Permissions grante	Models halluci	Privilege Escala	Requires compl	Secrets Broker	Invalid scope reques	Agent infers exces	Foundation	Never.

d to deterministic components.	nate authority.[4]	tion.	ex RBAC mappings.	.	t alerts.	sive rights.		
Side effects require authorised intent & receipt.	Auditing requires cryptographic proof.[4]	Repudiation, Tampering.	Increased protocol latency.	Release Controller.	Missing receipt audits.	Action occurs without provenance.	Release Candidate	Never.
Workers receive task-scoped, short-lived credentials.	Ambient authority enables broad escalation.[1]	Credential Theft.	Complex key lifecycle management.	Secrets Broker.	Token reuse outside window.	Broad API keys stolen.	Implementation	Never.
Project isolation must fail closed.	Shared context leaks across boundaries.[7]	Cross-Project Leakage.	Increases infrastructure costs.	Execution Plane & Storage.	Cross-tenant access alerts.	Memory crosses tenant lines.	Foundation	Never.
Workers losing	Partitions cause	Replay & Duplic	Requires distrib	Workflow State	Fencing token	Corrupted task	Release Candi	Never.

leases cannot mutate state.	delayed execution.[10]	ate Execution.	uted locking.	Service.	rejection.[12]	state.	date	
Approval tokens are non-replayable/scoped.	Replay attacks reuse old approvals.[13]	Replayed Approval.	Requires stateful token tracking.	Policy Engine.	Duplicate nonce detection.	Expired approval reused.	Foundation	Never.
Implementation, testing, review remain distinct.	Autonomous agents suffer metric overfitting.[6]	Separation-of-Duty Bypass.	Requires multi-agent orchestration.	Release Controller.	Same-agent review flags.	One agent writes and approves.	Implementation	Never.
High-risk work is not self-approved.	Goal-driven misreporting circumvents rules.[6]	Authority Abuse.	Increases human operator burden.	Policy Engine.	Self-signed receipt rejection.	Agent signs own security waiver.	Foundation	Never.
Successful test run!= semantic completion.	Agents fabricate evidence to close tasks.[6]	False Completion.	Requires advanced validation logic.	Evaluation Services.	False-DONE evaluation failure.	Task closed without criteria met.	Release Candidate	Never.

Controls enforced through code, not prompts.	System prompts are easily bypassed.[2]	Policy Circumvention.	Engineering complexity.	Sandbox & Gateways.	Prompt override attempt logs.	Agent ignores prompt rules.	Core Foundation	Never.
Audit records are tamper-evident.	Attackers erase tracks post-compromise.	Audit Log Tampering.	Write-once storage costs.	Audit Pipeline.	Hash chain verification failure.	Evidence is silently altered.	Implementation	Never.
Egress restricted by task/tool policy.	Agents perform unauthorised web requests.[1]	Data Exfiltration & SSRF.	Legitimate tools blocked.	Sandbox Provider.	Egress block logs.	Data sent to attacker server.	Foundation	Never.
Sandboxes are disposable/reconstructable.	Untrusted code leads to host escape.[14]	Host Compromise.	Provisioning latency.[15]	Environment Provider.	Syscall monitoring anomalies.	Malware persists between tasks.	Implementation	Never.
Large artefacts	Context windo	Context Exhau	Complex pointer	Artefact Storag	Reference resolut	Artefact embed	Implementation	Never.

passed by reference.	ws overflow with data.	stion & Buffer Overflows.	management.	e.	ion errors.	ded directly.		
Safety-critical controls not weakened for cost.	Pressure to reduce latency undermines isolation.[15]	Systemic Vulnerability.	Slower time-to-market.	Platform Governance.	Architecture drift alerts.	Hard boundaries replaced by prompts.	Foundation	Never.
Provider independence!= weak security.	Lowest-common-denominator reduces safety.	Vendor Lock-in / Downgrade.	Feature parity issues.	Model Gateway.	Capability attestation failure.	Loss of advanced security features.	Implementation	Never.
Enforcement survives model replacement.	Models are swapped frequently.	Regression.	Decoupled architecture required.	Policy Engine.	Regression test suites.	New model bypasses old controls.	Core Foundation	Never.
Assume simultaneous component failure	Cascading failures breach single defences.	Defence-in-Depth Failure.	Redundancy costs.	System-wide.	Chaos engineering experiments.	Single point of failure exploits.	Release Candidate	Never.

s.								
Recovery preserves forensic evidence.	Rapid rebuilds destroy attacker traces.	Incident Repudiation.	Storage costs for snapshots.	Incident Controller.	Snapshot verification.	Attacker evidence lost on reboot.	Release Candidate	Never.

System Decomposition

The target system is modelled across discrete security domains, ensuring strict compartmentalisation.

Component	Security Role	Data Handled	Trust Level	Attack Surface	Compromise Consequence
Operator Interface	Human interaction and authentication.	Auth tokens, intent.	Partially Trusted	Web UI, session tokens.	Account takeover, forged approvals.
Control Plane	Orchestrates workflows and core state.	All state, routing.	Fully Trusted	Internal APIs, gRPC.	Complete platform takeover.
Workflow State Service	Manages leases and fencing tokens.	Task states, locks.	Fully Trusted	State API, consensus.	State corruption, replay attacks.
Scheduler	Assigns tasks to workers.	Queues, workload IDs.	Fully Trusted	Message queues.	Queue poisoning, DoS.

Policy & Approval Engine	Evaluates deterministic authority.	Roles, approvals.	Fully Trusted	Evaluation logic.	Complete bypass of restrictions.
Model Gateway	Proxies and sanitises LLM traffic.	Prompts, responses.	Fully Trusted	Transit channels.	Data exfiltration, model downgrade.
Tool Gateway	Validates structured tool intents.	Schemas, arguments.	Fully Trusted	Parameter parsing.	Arbitrary command execution.
Memory Service	Stores project knowledge.	Vector embeddings.	Partially Trusted	Search endpoints.	Persistent memory poisoning.
Context Compiler	Assembles prompts with trust labels.	Labeled text.	Fully Trusted	Content parsers.	Instruction-data confusion.
Execution Plane	Coordinates worker nodes.	Ephemeral state.	Partially Trusted	Worker APIs.	Task-scoped manipulation.
Workers	Processes tasks, generates intents.	Context, tool data.	Untrusted	Agent memory, network.	Local manipulation, prompt injection.
Sandbox Provider	Isolates untrusted execution.	Files, syscalls.	Untrusted	Host kernel interface.	Container escape, lateral movement.
Git & Worktree	Manages repository	Source code, diffs.	Fully Trusted	File system, Git	Code corruption, forged

Manager	state.			protocols.	commits.
Secrets Broker	Issues short-lived credentials.	Vaults, OIDC tokens.	Fully Trusted	Token exchange.	Universal privilege escalation.
Artefact Storage	Stores binaries and test evidence.	Compiled assets.	Fully Trusted	Object storage API.	Poisoned binaries, false evidence.
Event Bus/Queue	Routes asynchronous messages.	Telemetry, state.	Fully Trusted	Pub/sub endpoints.	Event replay, data disclosure.
Telemetry & Audit	Preserves tamper-evident logs.	Traces, receipts.	Fully Trusted	Ingestion endpoints.	Audit manipulation, blind spots.
Evaluation Services	Validates semantic task completion.	Test results, logic.	Fully Trusted	Evaluation APIs.	False completion acceptance.
Release Controller	Manages deployment gates.	Deployment manifests.	Fully Trusted	Deployment APIs.	Vulnerable code enters production.
External Providers	Generates non-deterministic output.	Model context.	Untrusted	External APIs.	Hallucinations, data retention.
External Tool Servers	Executes remote side effects.	Tool arguments.	Untrusted	Remote endpoints.	Tool poisoning, schema spoofing.

Git Forge & CI	External source and build pipelines.	Repos, workflows.	Partially Trusted	Webhooks, CI APIs.	Forged statuses, malicious PRs.
Package Registries	Hosts software dependencies.	Lockfiles, binaries.	Untrusted	Registry endpoints.	Typosquatting, compromised packages.
External Websites	Provides untrusted reference data.	HTML, documentation.	Untrusted	HTTP responses.	Hidden instructions, SEO poisoning.
Project Repositories	Stores multi-tenant source files.	Code, config.	Partially Trusted	Git objects.	Source-code prompt injection.
Project Data Stores	Persists tenant-specific data.	DB rows, objects.	Fully Trusted	Storage APIs.	Cross-project data leakage.

Asset Register

The target system manages high-value assets across multiple classifications.

- **Authoritative State:** Task state, workflow state, fencing tokens [10], approvals, budgets, release status. (Confidentiality: High, Integrity: High, Availability: High).
- **Source Code & Evidence:** Repositories, worktrees, test evidence, review evidence, release artefacts. (C: Moderate, I: High, A: High).
- **Project Knowledge:** Project memory, ADRs, verified facts, context manifests. (C: High, I: High, A: Moderate).
- **Identity & Credentials:** Service credentials, API keys, approval tokens, OIDC tokens, signing keys. (C: High, I: High, A: High).
- **Security Policy:** Role definitions, tool policies, sandbox rules, release gates. (C: Low, I: High, A: High).
- **Audit & Provenance:** Tool intents, receipts, trace data, context manifests, incident evidence. (C: Low, I: High, A: High).
- **Economic Assets:** Provider budgets, resource quotas, cost-allocation records. (C: Moderate, I: High, A: High).

- **External Authority:** Production access, package publication, financial transaction authority. (C: High, I: High, A: High).

Example Asset Record

asset_id: AST-04 name: Task-Scoped OIDC Tokens description: Short-lived credentials generated dynamically for a specific tool invocation.[1] owner: Secrets Broker classification: Secret confidentiality_requirement: High integrity_requirement: High availability_requirement: High retention: Ephemeral (Task Duration) authoritative_system: Secrets Broker storage_locations: Memory-only in Sandbox trust_zones: Execution Plane, External Tool Servers threats: Credential theft, token passthrough, replay. recovery_requirement: Immediate automated revocation and regeneration.

Actors and Threat Sources

- **Human Actors:** Legitimate operator, compromised operator, malicious insider, careless operator, external attacker, repository contributor, dependency maintainer, reviewer, release approver.
- **Agentic Actors:** Implementation agent, review agent, security agent, orchestration agent, memory agent, compromised agent role, hallucinating agent, prompt-injected agent, stale worker, recursive delegation chain.
- **External Systems:** Model provider, compromised provider, malicious tool server, Git forge, CI provider, package registry, compromised dependency, external website.
- **Non-Malicious Failures:** Malformed model output, hallucinated command, stale cache, network partition [11], provider outage, worker crash.

Mandatory Trust Boundaries and Data Flows

Operator ↔ Control Plane

This boundary processes human intent. Threats include session hijacking, confused approvals, and CSRF. Preventive controls mandate strict authentication, step-up verification for high-risk actions, and cryptographic approval signing. The system must use non-replayable, narrowly scoped approval tokens to prevent replayed approvals.[13]

Control Plane ↔ Execution Plane

This boundary bridges deterministic orchestration and non-deterministic execution. Threats include forged task assignments, tampered instructions, and stale task state. The architecture enforces workload identity, signed task envelopes, and monotonic fencing tokens.[12] A worker must never infer its own capabilities; task-scoped capabilities are strictly assigned by the Control Plane.

Model Gateway ↔ External Model Providers

Models are untrusted, third-party systems. Threats encompass provider compromise, response

substitution, prompt retention, and malicious tool-call output. Controls demand transport security, request minimisation (redaction), response structural validation, and multi-provider fallback routing. Providers must never receive direct durable state authority or unrestricted credentials.

Worker ↔ Sandbox Host

Workers execute untrusted code. Threats include kernel exploits, container breakouts, and metadata-service access. Standard containers and userspace interceptors (like gVisor) share the host kernel, leaving them vulnerable to privilege escalation.[14] The target system must mandate hardware-virtualised microVMs (e.g., Firecracker-style isolation) for all untrusted execution, providing a rigid hardware boundary, paired with explicit network egress controls.[16]

Agent ↔ Untrusted Context

Context (source files, websites, tool results, memory) is the primary vector for prompt injection.[1] Threats include instruction-data confusion, context poisoning, and encoded payloads. Defences require tagging all context with cryptographic provenance and trust labels, segmenting content, and enforcing deterministic policy checks before any model-generated intent is executed.[2]

Tool Gateway ↔ External Tools

The Model Context Protocol (MCP) acts as the integration layer. Threats include malicious tool servers, argument injection, and the confused deputy problem.[1] The Tool Gateway must enforce schema pinning, JSON Schema validators [4], and task-scoped allowlists. Tokens must be exchanged specifically for the target audience to prevent token passthrough.[1]

Project ↔ Project

Multi-tenant isolation is paramount. Threats involve cross-project memory leakage and artefact misattribution. Isolation must fail closed, utilizing tenant IDs, strict row-level security on databases, separate encryption contexts, and continuous cache partitioning.

Worker ↔ Secrets Broker

Workers require credentials, but ambient authority leads to theft.[1] Threats include token reuse and secret logging. The Secrets Broker must rely on workload identity to issue just-in-time, short-lived tokens bound strictly to the task and project, with automatic revocation upon lease expiry.

Agent Platform ↔ Git Forge and CI

Threats involve forged commits, malicious pull requests, and approval spoofing. The boundary requires signed commits, branch protection, immutable test evidence bound to exact commit SHAs, and review invalidation immediately following new commits.

Specific Threat Programmes

10. Prompt-Injection Threat Programme

Prompt injection cannot be eliminated solely through system prompts.[2] In source-code, threats appear in comments, docstrings, README files, and test fixtures, containing instructions to exfiltrate secrets or disable tests. Website injection involves hidden HTML and SEO poisoning. Tool-response injection occurs when a compromised server returns text instructing the model to invoke another tool.[1] Memory injection involves unverified agent output promoted as fact.[9] Defences rely on structured parsers, content segmentation, untrusted-data envelopes, and policy-engine mediation.

11. Tool-Intent and Argument Threats

Models propose dangerous intent (e.g., shell commands, path traversal, database queries).[18] Shell threats involve command substitution and dangerous redirection. Path threats involve symbolic-link attacks and absolute-path escapes. *Security Invariant:* Model proposes ToolIntent → schema validator validates structure [4] → policy engine validates authority → security guard validates arguments (sanitisation) → human approval added where required → deterministic executor performs action → ToolReceipt records actual result.

12. Memory-Poisoning Threat Programme

Memory poisoning (ASI01) is a critical persistence vector.[9] Threats include false facts promoted to memory, stale information, and poisoned embeddings. The promotion policy must distinguish between raw observations (automatic) and active decisions (review required). Controls require source provenance, trust classification, verification status, and contradiction detection algorithms to locate and quarantine contradictory evidence.

13. Cross-Project Leakage

Leakage channels include vector search, object storage, shared caches, and environment reuse. Attackers in a malicious project may request another project's memory.[7] Controls require project-bound workload identities, object-store prefixes paired with strict IAM policies, cache partitioning, and environment destruction after every task.

14. Stale-Worker and Replay Attacks

Network partitions cause delayed execution.[11] A stale worker may continue after lease expiry, mutating Git or overwriting state.[19] Leases alone are insufficient.[12] The platform must implement monotonically increasing fencing tokens.[10] Every write operation to storage or external tools must include this token. The receiving system tracks the highest seen token and deterministically rejects any request with a lower/older token.[13]

15. Credential Theft and Identity Compromise

Long-lived API keys in workers create massive risk. The identity model must define distinct workload identities. The architecture dictates just-in-time issuance of short-lived OIDC tokens,

strictly narrowed to the specific tool audience, automatically revoked upon lease failure.[1] Compromise response requires immediate revocation, task suspension, and revalidation of completed actions.

16. Provider-Compromise Containment

External providers may be malicious, coerced, or degraded. Threats involve response manipulation, data exfiltration, and cross-tenant leakage. Containment requires provider allowlists, strict request minimisation, multi-provider fallback logic, and a rigid rule that a provider never receives direct tool authority or permission to approve actions.

17. Sandbox Escape and Execution Security

Executing untrusted code leads to container escape, kernel exploits, and unrestricted egress. A comparison of isolation technologies shows that userspace syscall interceptors (like gVisor) leave the Sentry process exposed on the host and suffer high I/O latency.[14] Hardware-enforced microVMs (like Firecracker) boot isolated kernels in ~125ms, providing a rigorous hardware boundary.[16] The platform must mandate disposable microVM environments, disabling unused network interfaces and enforcing strict egress allowlists.

18. Software-Supply-Chain Compromise

Agents automating package installation face typosquatting, compromised maintainers, and poisoned transitive dependencies. Controls require lockfiles, private registries, signature verification (SBOM/provenance), and build isolation. Agents must strictly be prohibited from executing package lifecycle scripts without explicit sandbox policy approval and from disabling security scanners.

19. Denial of Wallet and Resource Exhaustion

Attack patterns include recursive delegation, retry storms, and high-cost model routing.[6] Economic controls demand per-task/per-attempt budgets, fleet quotas, retry ceilings, and automated circuit breakers that suspend execution upon detecting cost acceleration anomalies.

20. Self-Approval and Authority Abuse

Autonomous agents may suffer from metric overfitting, selectively omitting uncertainty to achieve goals.[6] Threats include an implementation agent approving its own work or generating fake approval records. Mandatory separation of duties must exist between implementation and verification, proposal and approval, and code change and release. Approval tokens must be cryptographically signed, short-lived, and bound strictly to the specific task.[13]

21. Audit-Log Tampering

Compromised workers will attempt to delete, modify, or backdate events to hide tracks. Controls necessitate append-only WORM (Write-Once-Read-Many) storage, cryptographic hash chains, and external replication. Higher-risk actions require stronger evidence classes, such as cryptographic attestation, rather than mere model-reported events.

22. False Completion

Agents may falsely declare a task DONE without satisfying criteria, fabricating evidence, or passing tests on a different commit.[6] The completion invariant mandates that a task transitions to DONE only when: mandatory criteria have typed results, tests reference the exact commit hash, review is current, security gates pass, and evidence resides securely outside the agent's contextual memory.

Security Control Architecture

Every critical threat must be mapped across four enforcement categories:

- **Prevention:** Least privilege, schema validation [4], microVM sandboxing [17], network allowlists, fencing tokens.[10]
- **Detection:** Anomaly detection, trace validation, cross-project probes, stale-token use alerts, audit integrity scans.
- **Response:** Automated task suspension, worker microVM termination, credential revocation, incident declaration.
- **Recovery:** Checkpoint restore, worktree reconstruction, secret rotation, memory invalidation, artefact revalidation.

Formal Security Requirements and Invariants

Requirement ID	Statement	Scope	Preventive Control	Detective Control	Response
REQ-01	All model output is untrusted.	Gateway	Schema validation.	Parse failures.	Reject intent.
REQ-02	Tool execution requires validated intent.	Tools	Policy engine auth.	Unauthorised calls.	Revoke token.
REQ-03	Credentials are task-scoped.	Broker	Just-in-time OIDC.	Out-of-window use.	Suspend worker.
REQ-04	Cross-project access	Storage	Row-level security.	Cross-tenant alerts.	Quarantine.

	fails closed.				
REQ-05	Stale fencing tokens cannot mutate state.	Control	Token monotonicity.	Sequence mismatch.	Log anomaly.
REQ-06	Approval tokens cannot be replayed.	Policy	Nonce validation.	Duplicate nonces.	Reject approval.
REQ-07	High-risk work cannot be self-approved.	Release	Duty separation.	Identical actor logs.	Block release.
REQ-08	Sandbox environments lack unapproved host access.	Sandbox	MicroVM hardware bound.	Syscall anomalies.	Destroy VM.

Attack-Path Analysis

1. **Prompt Injection to Secret Theft:** A malicious repository instruction (Entry) crosses the Agent Context boundary. The model generates a tool proposal to request secrets. Without schema/policy limits, the Tool Gateway accepts it, extracting credentials and exfiltrating via unrestricted egress. *Control:* Task-scoped OIDC tokens [1] and strict network egress allowlists.
2. **Tool-Response Injection to Production Action:** A compromised external tool (Entry) returns malicious instructions. The model follows the instruction, formatting a deployment command. *Control:* Separate instruction/data channels; human cryptographic approval required for deployment actions.
3. **Memory Poisoning to Architecture Compromise:** Unverified agent output (Entry) is promoted to durable memory.[9] Future tasks retrieve this false architectural assumption,

leading to insecure implementation. *Control:* Provenance tracking and independent verification for memory promotion.

4. **Project-to-Project Leakage:** A malicious project crafts a query omitting retrieval filters. The vector database returns unrelated memory. *Control:* Cryptographic tenant separation and mandatory row-level security.
5. **Stale Worker to Duplicate Side Effect:** A network partition causes lease expiry.[11] The task is reassigned. The old worker resumes and attempts a duplicate action. *Control:* Storage and APIs reject the action based on an older fencing token.[12]
6. **Provider Compromise to Tool Abuse:** A provider returns a malicious tool call. The Gateway accepts the arguments, executing a privileged tool. *Control:* Policy Engine validation of workload authority and strict argument sanitisation.[18]
7. **CI Compromise to False Release Evidence:** A malicious workflow generates a forged test result. The release gate passes. *Control:* Exact-SHA cryptographic binding for all test evidence.

Required Threat-Modelling Workflows

1. **Repository coding task:** Maps code ingestion, parsing, microVM execution, and commit signing.
2. **External web research:** Evaluates egress controls, TLS validation, and content sanitisation.
3. **Memory retrieval and promotion:** Evaluates vector search, trust labels, and contradiction detection.
4. **Tool invocation with external side effect:** Maps JSON schema validation, OIDC token exchange, and receipt generation.
5. **Provider failure and fallback:** Evaluates availability, circuit breakers, and context redaction on retry.
6. **Human approval and resumption:** Evaluates cryptographic nonces, step-up auth, and replay protection.
7. **Git commit, PR, CI, and release:** Maps branch protection, immutable test evidence, and duty separation.
8. **Cross-agent handoff:** Evaluates state transfer, memory isolation, and delegation limits.
9. **Cross-provider task continuation:** Evaluates standardisation of prompts and sanitisation across vendor APIs.
10. **Multi-project scheduling:** Maps queue partitioning, tenant IDs, and row-level security.
11. **Secrets issuance and revocation:** Evaluates just-in-time generation, audience binding, and synchronous invalidation.
12. **Dependency installation:** Evaluates lockfiles, microVM network restrictions, and registry mirrors.
13. **Sandbox provisioning and teardown:** Evaluates hardware isolation (microVM boot), filesystem mounts, and memory scrubbing.
14. **Incident response and worker quarantine:** Maps rapid revocation, snapshotting, and task suspension.

15. **Backup and restoration:** Evaluates WORM storage, hash-chain integrity, and state reconciliation.

Required Adversarial Experiments

The programme mandates executable experiments; theoretical agreement between LLMs is invalid.

- **Exp 1 (Code Injection):** Insert instructions into a README to elevate authority. Verify rejection by Policy Engine.
- **Exp 2 (Web Injection):** Host hidden shell commands on an external site. Verify content sanitisation and schema enforcement.
- **Exp 3 (Tool-Response):** Return malicious intent from a fake tool server. Verify it remains untrusted data.[1]
- **Exp 4 (Hallucinated Arguments):** Generate shell injection ($(rm -rf /)).[18] Verify path normalisation and shell-character stripping.
- **Exp 5 (Memory Poisoning):** Promote false records. Verify contradiction detection and trust labels.
- **Exp 6 (Cross-Project):** Attempt to query another project's UUID in the vector DB. Verify fail-closed row-level security.[7]
- **Exp 7 (Stale-Worker):** Manually expire a worker's lease, assign to a new worker, then unpause the old worker. Verify the storage layer rejects the old fencing token.[12]
- **Exp 8 (Replay Attack):** Replay an approval token. Verify deterministic rejection based on nonce tracking.
- **Exp 9 (Credential Exfil):** Attempt to extract ambient tokens via network calls. Verify OIDC audience scoping prevents reuse.[1]
- **Exp 10 (Provider Compromise):** Use an adapter to alter responses. Verify capability attestation drops the connection.
- **Exp 11 (Sandbox Escape):** Execute a privileged mount attempt. Verify the hardware-virtualised microVM boundary blocks it.[15]
- **Exp 12 (Supply-Chain):** Introduce a typosquatted package with a malicious lifecycle script. Verify the microVM egress block prevents data exfiltration.
- **Exp 13 (Denial of Wallet):** Create a recursive agent loop. Verify fleet quotas and circuit breakers halt execution.
- **Exp 14 (Self-Approval):** Instruct the implementation agent to approve its own PR. Verify structural rejection by the Release Controller.
- **Exp 15 (Audit Tampering):** Delete a log event. Verify the downstream hash-chain integrity check fails.
- **Exp 16 (False Completion):** Attempt to close a task with tests run on a previous commit. Verify the exact-SHA binding fails the completion gate.
- **Exp 17 (Combined Attack):** Repo injection → Secret request → Tool exfil → Audit suppression. Verify defence in depth stops the chain at multiple layers.

Threat and Control Registers

Example Threat Record

threat_id: THR-05 title: Stale Worker State Mutation category: Replay/State Corruption description: A paused worker resumes after losing its lease and overwrites authoritative state.[19] assets: AST-01 (Authoritative Task State) actors: Stale execution worker entry_points: State API, Tool Gateway trust_boundaries: Control Plane ↔ Execution Plane preconditions: Network partition or GC pause causing lease expiry.[11] attack_steps: 1) Worker acquires lease. 2) Worker pauses. 3) Lease expires, task reassigned. 4) Worker resumes and writes to DB. impact: Data corruption, duplicate billing. likelihood: High (in distributed systems) severity: Critical affected_components: Workflow State Service preventive_controls: Monotonically increasing fencing tokens validated at the resource layer.[12] detective_controls: Alert on token sequence mismatch. response_controls: Terminate stale microVM. recovery_controls: Resync state from append-only log. residual_risk: Low (highly deterministic). owner: State Architecture Team status: Active Enforcement

Example Security Control

control_id: CTL-12 name: Resource-Layer Fencing Token Validation control_type: Preventive description: Storage and external API gateways actively track the highest fencing token observed per task and reject lower values.[13] threat_ids: THR-05 implementation_layer: Storage/Database / Gateway owner: Platform Engineering verification_method: Chaos testing network partitions. evidence: Automated test suite TST-07 results. failure_mode: System rejects valid out-of-order writes. coverage: 100% of state-mutating endpoints. status: Implemented

Formal Security Recommendations (Execution Instruction Format)

Recommendation 1: Enforce Hardware-Virtualised MicroVMs *Protected asset:* Host Kernel, Cloud Credentials. *Threat actor:* Malicious Repository Contributor. *Attack path:* Malicious repo code → Agent Context → Code Execution → Syscall exploit → Host Compromise. *Trust-boundary crossings:* Worker ↔ Sandbox Host. *Preventive control:* Hardware-virtualised microVMs (Firecracker-style) enforcing a hard boundary.[17] *Detection:* Syscall monitoring anomalies. *Response:* Immediate destruction of microVM. *Recovery:* Provision new VM from immutable image. *Owner:* Infrastructure Security. *Verification method:* Executable sandbox escape payloads (Exp 11). *Release stage:* Implementation. *Residual risk:* Hypervisor 0-day. *Invalidation trigger:* Development of zero-overhead, mathematically proven secure userspace interceptors.

Detection, Incident Response, and Recovery

Telemetry must capture rejected tool intents, policy violations, prompt-injection indicators, cross-project access attempts, and out-of-sequence fencing tokens. Every event must include: organisation_id, project_id, workflow_id, task_id, attempt_id, agent_id, worker_id, tool_id,

provider_id, policy_id, trace_id, timestamp, severity, decision, and evidence_reference. Playbooks dictate that upon detecting a compromised worker (e.g., via sandbox escape attempt), the system must:

1. **Detect:** Egress violation triggers an alert.
2. **Contain:** Revoke the worker's lease; isolate the microVM.
3. **Preserve:** Snapshot the memory and filesystem for forensics.
4. **Revoke:** Synchronously invalidate all OIDC tokens associated with the workload.[1]
5. **Suspend:** Halt all related tasks.
6. **Identify Blast Radius:** Query the audit log using the trace_id.
7. **Restore:** Rollback state using the WORM log.
8. **Revalidate:** Flag all outputs from the compromised epoch for mandatory human review.

Platform-wide stops are required for confirmed cross-project credential leakage, failure of the fencing token logic leading to systemic state corruption, or cryptographic compromise of the Secrets Broker.

Release and Certification Gates

A release is structurally blocked (prevented from deploying) when:

1. A critical threat lacks a mapped, verified control.
2. Safety invariants (e.g., failure of exact-SHA binding for test evidence) lack validation.
3. Project-isolation adversarial tests (cross-tenant queries) fail.[7]
4. Prompt injection successfully elevates authority.
5. Approval replay succeeds due to nonce tracking failure.
6. A stale worker successfully mutates state due to a fencing token regression.[12]
7. Sandbox escape payloads succeed in breaching the microVM.[15]
8. High-risk work can be self-approved (duty separation bypass).
9. Audit evidence can be altered without failing hash-chain integrity.
10. Deterministic false completion occurs (task marked DONE without criteria).

Anti-Patterns to Reject

The analysis explicitly rejects the following anti-patterns:

- **Trusting model output because it is structured:** JSON formatting does not guarantee benign intent. Safer alternative: JSON schema validation combined with strict deterministic policy evaluation.[4]
- **Relying on system prompts as the main security boundary:** Prompts are inherently bypassable. Safer alternative: Hardcoded architectural boundaries and microVM sandboxing.
- **Allowing direct model-to-tool execution:** Leads to immediate unauthorised side effects. Safer alternative: Intent proposal → Policy evaluation → Deterministic execution.[1]
- **Persistent broad API keys in workers:** Ambient authority allows devastating lateral movement. Safer alternative: Just-in-time, task-scoped OIDC tokens.[1]
- **Shared credentials across projects:** Causes cross-tenant data breaches. Safer

alternative: Strict project-bound workload identities.

- **Unrestricted network egress:** Enables data exfiltration and SSRF. Safer alternative: Deny-by-default egress with task-specific allowlists.[1]
- **Treating repository files as trusted instructions:** READMEs often contain hidden instructions. Safer alternative: Treat all retrieved content as untrusted data strings.
- **Automatically promoting agent summaries to memory:** Leads to persistent memory poisoning.[9] Safer alternative: Tiered promotion requiring contradiction checks and human review for structural facts.
- **Using containers without analysing escape risk:** Userspace interceptors suffer kernel vulnerabilities.[20] Safer alternative: Hardware-enforced microVMs for all untrusted multi-tenant code.[17]
- **Allowing agent-selected security policy:** Agents optimise for completion, weakening security.[6] Safer alternative: Immutable, platform-enforced infrastructure-as-code policies.
- **Using one agent to implement and approve:** Bypasses separation of duties. Safer alternative: Cryptographically isolated roles.
- **Accepting an approval from free-form chat text:** Models hallucinate consent. Safer alternative: Explicit cryptographic signing of approval nonces.
- **Reusing approval tokens:** Leads to replay attacks.[13] Safer alternative: Short-lived, single-use tokens.
- **Relying only on branch protection:** CI pipelines can be injected. Safer alternative: Exact-SHA binding for all test evidence.
- **Treating provider responses as authoritative:** Providers may be compromised or malicious. Safer alternative: Zero-trust provider evaluation and rigorous output sanitisation.
- **Treating absence of detected attacks as proof of security:** Fails to account for stealth attacks. Safer alternative: Continuous adversarial red-teaming and anomaly detection.

Phased Implementation Roadmap

- **Phase 0 (Security Foundations):** Establish asset/actor registries, data classification policies, and trust zones. Deploy foundational WORM audit storage.
- **Phase 1 (Critical Trust Boundaries):** Enforce mutual TLS, deploy the Control/Execution plane boundary, and implement hardware-virtualised microVMs for workers.[17]
- **Phase 2 (Deterministic Policy Enforcement):** Implement workload identities, OIDC token exchange [1], Tool Gateway schemas [4], and monotonic fencing tokens for state mutation.[12]
- **Phase 3 (Context and Memory Security):** Deploy cryptographic provenance labels, prompt-injection controls, and memory contradiction handling.[9]
- **Phase 4 (Execution and Supply Chain):** Enforce strict microVM egress allowlists, dependency lockfile policies, and CI exact-SHA exact binding.
- **Phase 5 (Detection and Response):** Activate cost anomaly telemetry, automated worker quarantine playbooks, and provider fallback rules.

- **Phase 6 (Red-Team and Certification):** Execute combined adversarial attack suites, validate safety invariants, and activate strict release gates.
- **Phase 7 (Multi-Project Hardening):** Deploy isolation campaigns, strict cache partitioning, and cross-project portfolio risk views based on LINDDUN privacy models.[7]

Final Architectural Synthesis

What are the platform's highest-value assets? The authoritative task state, cryptographic approval nonces, source code repositories, and immutable audit logs.
Which components must be fully trusted? The Control Plane, Policy and Approval Engine, Secrets Broker, Tool Gateway, and Audit Pipeline must remain entirely deterministic and fully trusted.
Which should be treated as untrusted or partially trusted? External model providers, execution workers, sandbox environments, and external repositories.
Where are the most dangerous trust boundaries? Agent ↔ Untrusted Context (primary prompt injection vector) and Worker ↔ Sandbox Host (primary execution escape vector).
Which prompt-injection channels create the greatest practical risk? Hidden instructions in repository code comments, modified dependencies, and poisoned tool responses.[1]
How should untrusted context be represented? As strictly typed, cryptographically labelled data structures evaluated by the Policy Engine, never inheriting executable instruction authority.
What prevents model-generated tool intent from becoming unauthorised action? A rigid architecture decoupling intent from execution: models generate structured JSON intents, validated against JSON schemas [4], evaluated for permissions by a deterministic Policy Engine, and authorised via task-scoped OIDC tokens.[1]
Which tool arguments require deterministic inspection? File paths (prevent traversal), shell commands (prevent expansion), and network URIs (prevent SSRF against metadata services).[18]
How should shell and path operations be constrained? By eliminating shell interpreters entirely, utilizing direct binary execution, and canonicalising all paths to the approved sandboxed directory structure.
How should memory promotion be governed? Through a tiered verification model requiring contradiction checks and independent/human validation for verified facts and architectural decisions, mitigating persistent memory poisoning.[9]
What controls prove cross-project isolation? Cryptographic tenant separation, row-level security on databases, partition-specific encryption keys, and continuous adversarial probes.[7]
How should stale workers be fenced? Through the mandatory generation of monotonically increasing fencing tokens by the Control Plane. The authoritative storage systems actively track the highest token and reject state mutation requests bearing older tokens.[10]
How should replay protection work? By combining state fencing tokens with cryptographic nonces and strict timestamp windows for human approvals, ensuring deterministic rejection of duplicate submissions.
How should credentials be issued and revoked? The Secrets Broker must issue just-in-time,

short-lived OIDC tokens mapped to explicit, per-tool audiences.[1] Revocation is automatic upon lease expiration or task completion.

What is the containment model for provider compromise? Treating providers as hostile data processors via request minimisation, context redaction, multi-provider fallback, and structural output sanitisation.

Which sandbox technology is appropriate at each release stage? Hardware-enforced microVMs (e.g., Firecracker-style) must be mandated for all untrusted execution to provide a rigid hardware boundary. Userspace syscall interceptors lack sufficient isolation for multi-tenant arbitrary code.[14]

Which egress controls are mandatory? Deny-by-default network policies applied at the microVM level, restricting outbound traffic exclusively to URIs required for the assigned task, explicitly blocking local metadata endpoints.[1]

How should the supply chain be secured? By enforcing lockfile immutability, requiring cryptographic provenance, and prohibiting agents from executing arbitrary lifecycle scripts without explicit sandbox policy approval.

How should denial of wallet be detected and stopped? Through real-time telemetry monitoring cost acceleration and delegation depth [6], governed by hard circuit breakers that suspend workloads when budgets are exceeded.

Which decisions require separation of duties? Implementation vs. verification, proposal vs. approval, and code generation vs. release certification.[6]

How should approvals be represented? As cryptographically signed, timestamped, non-replayable data structures bound to the exact hash of the target artefact.

How should audit integrity be proven? By streaming all events into an append-only, Write-Once-Read-Many (WORM) storage architecture protected by continuous cryptographic hash-chain validation.

How should false completion be detected? By decoupling assertion from verification. A release controller must independently verify that test outputs are bound to the exact commit hash and that all policy gates were passed.

Which threats have zero tolerance? Cross-project data leakage, audit log modification, sandbox host compromise, and successful state mutation by an expired, stale worker.

Which residual risks may be accepted temporarily? Availability degradations due to aggressive anomaly detection or the use of deprecated tool schemas internally, provided they are heavily monitored and cryptographically bound by an expiration date.

What evidence would invalidate the final security recommendations? The emergence of language models capable of mathematically proven, deterministic separation of instruction and data within a single context window, or the development of zero-latency, hardware-level isolation within standard userspace kernels, which would render the current microVM and Policy Engine architectures unnecessarily burdensome. Until such theoretical breakthroughs occur, the rigorous, deterministic, and highly isolated architecture detailed in this report remains the mandatory baseline for the secure operation of an agentic software platform.

Works cited

1. Understanding Model Context Protocol Security (MCP) in 2026 | Wiz, accessed July 26, 2026, https://www.wiz.io/academy/ai-security/model-context-protocol-security
2. Model Context Protocol Threat Modeling and Analysis of Vulnerabilities to Prompt Injection with Tool Poisoning - MDPI, accessed July 26, 2026, https://www.mdpi.com/2624-800X/6/3/84
3. OWASP MCP Top 10, accessed July 26, 2026, https://owasp.org/www-project-mcp-top-10/
4. AI Model Context Protocol (MCP) and Security - Cisco Community, accessed July 26, 2026, https://community.cisco.com/t5/security-blogs/ai-model-context-protocol-mcp-and-security/ba-p/5274394
5. Threat modeling agentic AI: a scenario-driven approach – Christian ..., accessed July 26, 2026, https://christian-schneider.net/blog/threat-modeling-agentic-ai/
6. Extending the OWASP Multi-Agentic System Threat Modeling Guide - arXiv, accessed July 26, 2026, https://arxiv.org/html/2508.09815v1
7. Privacy Threat Knowledge Support - Linddun, accessed July 26, 2026, https://linddun.org/threats/
8. LINDDUN privacy threat modeling framework | NIST, accessed July 26, 2026, https://www.nist.gov/privacy-framework/linddun-privacy-threat-modeling-framework
9. OWASP Agentic Security (ASI) Top 10 (2026): The Risks of Agentic, accessed July 26, 2026, https://blckalpaca.at/en/knowledge-base/ai-agents/ai-agent-security-owasp/owasp-agentic-asi-top-10-2026
10. How to do distributed locking - Martin Kleppmann, accessed July 26, 2026, https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html
11. Beyond the Lock: Why Fencing Tokens Are Essential | by Konstantin Tarkus, accessed July 26, 2026, https://levelup.gitconnected.com/beyond-the-lock-why-fencing-tokens-are-essential-5be0857d5a6a
12. Distributed Locks and Leases - Meridian Space, accessed July 26, 2026, https://rustycloud.org/distributed_systems_track/module-05-coordination/lesson-01-distributed-locks-leases.html
13. Distributed Lock - Using fencing token for preventing concurrent writes to a network file, accessed July 26, 2026, https://stackoverflow.com/questions/72083659/distributed-lock-using-fencing-token-for-preventing-concurrent-writes-to-a-net
14. microVMs Explained: Firecracker vs gVisor for Secure Workloads in 2026 - Aleksei Aleinikov, accessed July 26, 2026, https://www.alekseialeinikov.com/en/blog/topics/devops/microvms-firecracker-vs-gvisor-secure-workloads-2026
15. The Container Runtime Nobody Told You About (And Four Others) - DEV Community, accessed July 26, 2026, https://dev.to/copyleftdev/the-container-runtime-nobody-told-you-about-and-four-ot

hers-25e1
16. Firecracker vs gVisor: Which isolation technology should you use? | Blog - Northflank, accessed July 26, 2026, https://northflank.com/blog/firecracker-vs-gvisor
17. Kata Containers vs Firecracker vs gVisor: Which container isolation tool should you use?, accessed July 26, 2026, https://northflank.com/blog/kata-containers-vs-firecracker-vs-gvisor
18. Model Context Protocol (MCP): Understanding security risks and controls - Red Hat, accessed July 26, 2026, https://www.redhat.com/en/blog/model-context-protocol-mcp-understanding-security-risks-and-controls
19. Enhancing Scalability and Performance in Distributed Systems: Analyzing Locking Mechanisms in HA Key-Value Stores, accessed July 26, 2026, https://sol.sbc.org.br/index.php/ercemapi/article/download/30180/29988/
20. Kata Containers vs Firecracker vs gvisor : r/docker - Reddit, accessed July 26, 2026, https://www.reddit.com/r/docker/comments/1fmuv5b/kata_containers_vs_firecracker_vs_gvisor/