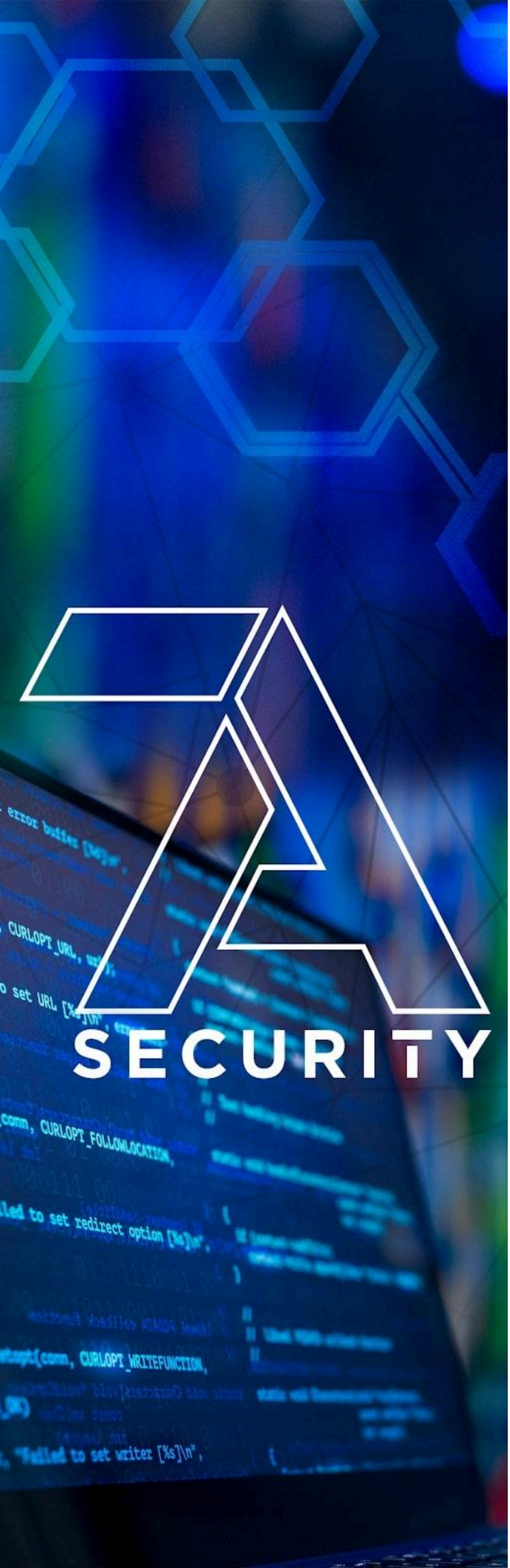

PLATINUM

ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

AICPA
SOC 2
Formerly SAS 70 Reports

Lightweight Threat Model

Client:
Super Tanks Maintainers
KNDW Shelter Solutions AS

Super Tanks Test Targets:

Core Security Architecture
Published Threat Model
OWASP Agentic Mapping

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Noelia Aranguren

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

"The maintainer design goal for Super Tanks is to prevent unsafe AI-agent actions before execution rather than detect them after the fact. This report treats that design goal as defense in depth: individual layers can fail, so security depends on safe composition, fail-closed behavior, auditability, and containment."

From Super Tanks Maintainers

This document outlines the results of a lightweight threat model conducted against the Super Tanks platform. The project was solicited by Super Tanks and executed by 7ASecurity in June 2026. The audit team dedicated 1.35 working days to complete this assignment.

The methodology implemented was *whitebox*: 7ASecurity was provided with access to documentation and source code. A team of 2 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by June 2026, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email. The Super Tanks team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

Moving forward, the scope section elaborates on the items under review, while the threat model section documents the identified security mechanisms in place and provides ideas for hardening the security posture.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Super Tanks applications.

Scope

The following list outlines the items in scope for this project:

- **WP1: Super Tanks Lightweight Threat Model**
  - Super Tanks v3.2, commit *c99a908*: https://github.com/KNDW-AS/super-tanks/…ba9
  - NOTES:
    - This is not a penetration test.
    - The engagement used 1.35 person-days, whitebox access to documentation and source code, and limited source-code confirmation.
    - No running exploit validation was performed.

Executive Summary

Super Tanks implements a credible defense-in-depth architecture for AI-agent governance, with meaningful controls around gateway enforcement, identity, approval, memory, code quarantine, integrity, modes, and auditability. The highest-value hardening areas are per-execution sandboxing, external integrity anchors, audit key separation, trust and approval evidence chaining, and continued documentation alignment with implemented controls.

WP1: Super Tanks Lightweight Threat Model

Introduction

Super Tanks is an open-source governance layer for autonomous AI agents deployed on a Python, Docker, SQLite, Ollama, and Telegram-oriented local stack. The project presents a defense-in-depth control model that mediates agent actions before they reach tools, models, memory, code, smart-home integrations, or the outside world[1].

The threat model analysis in this document identifies security threats and vulnerabilities to enable early mitigation. The document, together with the related attack scenarios, establishes a baseline to encourage all team members to adopt a threat-led mindset toward everything designed and implemented, focusing on security from the outset to address risks before they evolve into exploitable vulnerabilities. A lightweight STRIDE-based approach was applied using project documentation, the provided source tree, and limited source-code confirmation to assess Super Tanks.

This section classifies attack scenarios, outlines potential weaknesses, and proposes mitigations. The analysis focuses on Super Tanks security architecture, published threat model claims, OWASP Agentic Top 10 mapping, gateway control points, agent identity, human approval, memory, code quarantine, auditability, and integrity mechanisms[2][3][4][5][6].

[1] https://github.com/kndw-as/super-tanks
[2] https://github.com/kndw-as/super-tanks/blob/main/README.md
[3] https://github.com/kndw-as/super-tanks/blob/main/SECURITY.md
[4] https://github.com/kndw-as/super-tanks/blob/main/SYSTEM_CARD.md
[5] https://github.com/kndw-as/super-tanks/blob/main/docs/RISK_REGISTER.md
[6] https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

Relevant assets and threat actors

Key Assets:

- Super Tanks source code, release artifacts, install scripts, Docker configuration, and Python package metadata
- Agent identities and HMAC signing key material, including data/.identity_key and issued identity tokens
- Soul files, soul_integrity.json, DIQ contract files, and DIQ_CHECKSUMS.json integrity state
- Gateway, DIQ registry, per-agent allowlists, tool-zone boundaries, and allowed_agents skill isolation
- GO-Gate approval requests, request IDs, argument hashes, Telegram approval channel, and administrator decisions
- Memory stores, shadow proposals, tripwires, memory audit database, dispatch audit database, trust score database, and audit-chain state
- ZEF prompt-injection patterns, LLM classifier path, adversarial corpus, and tool-output provenance metadata
- Code quarantine queue, AST scanner policy, proposed patches, approved changes, and live source tree
- Home Assistant entities, locks, sensors, files, web browsing outputs, cloud model tokens, Ollama models, and other controlled tools

Relevant Threat Actors:

- External attacker supplying malicious prompts, web content, files, memory entries, or A2A messages
- Compromised or prompt-injected agent attempting to exceed the intended READ, CHAT, WRITE, EXEC, or ADMIN surface
- Malicious or compromised local user with partial deployment access but without full host compromise
- Malicious or compromised maintainer, dependency, model, MCP server, or tool provider influencing the agentic supply chain
- LAN attacker or smart-home attacker attempting to manipulate local service calls or approval flows
- Advanced persistent threat targeting the agent governance layer, operator trust, or long-lived memory context

System context and trust boundaries

Super Tanks is centered on a local deployment where agents receive untrusted user, tool, file, memory, web, and inter-agent content, then route actions through ZEF filtering, identity verification, DIQ role checks, per-agent allowlists, GO-Gate approval, mode and trust controls, memory RBAC, code quarantine, and audit logging before reaching tools or external effects. The main trust boundaries are between untrusted content and agent context, agents and the gateway, the gateway and high-impact tools, the administrator and GO-Gate, runtime code and the source tree, and local host controls outside the Super Tanks process.

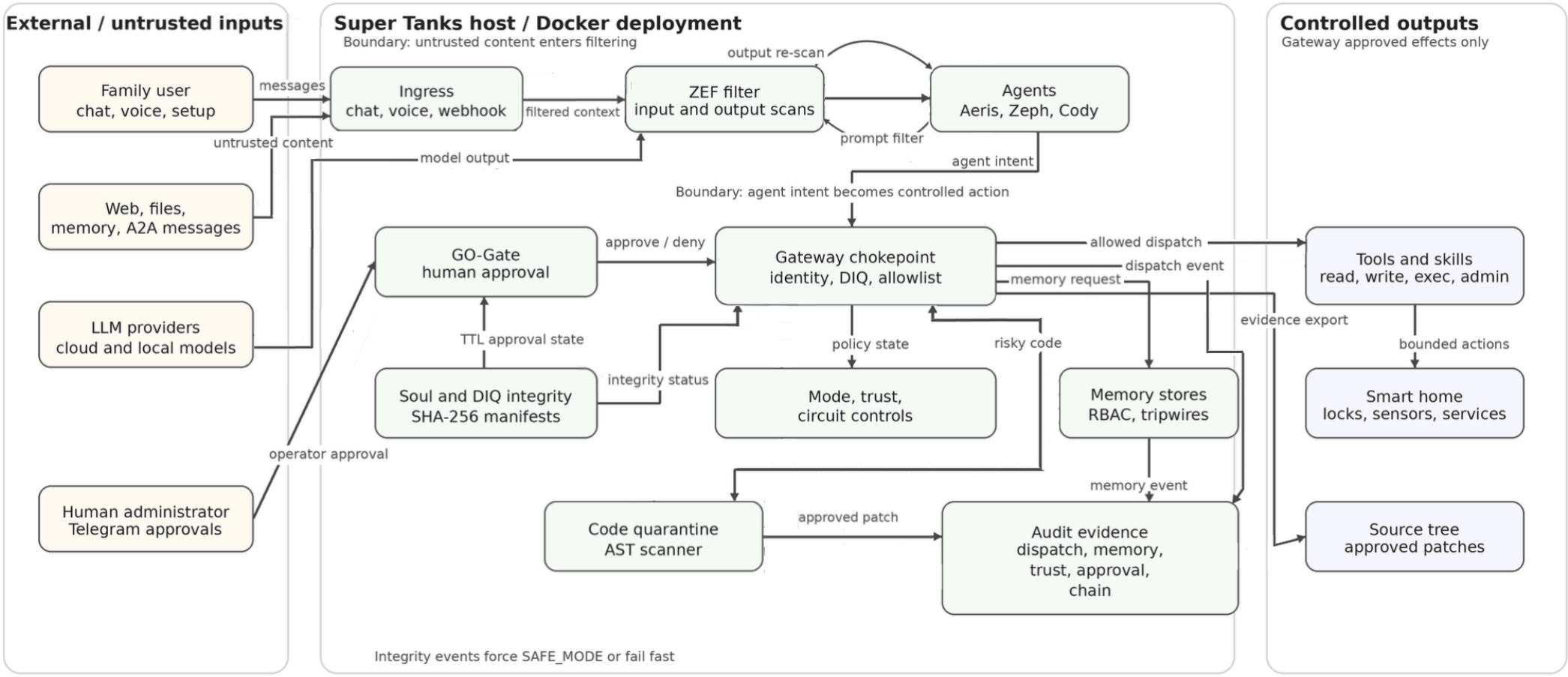

Fig.: Simplified Super Tanks system context and trust boundaries

Attack surface

The attack surface includes all potential entry points that an attacker may exploit to compromise the system, manipulate agent decisions, access sensitive data, bypass controls, or disrupt availability. For Super Tanks, the most relevant surfaces are untrusted natural-language input, tool outputs reintroduced into model context, A2A messaging, identity tokens, DIQ contracts, per-agent allowlists, GO-Gate requests, Telegram approvals, memory paths, shadow proposals, code-quarantine inputs, model-provider outputs, smart-home service arguments, install scripts, and local host files.

The provided source tree materially narrows uncertainty compared with a documentation-only review because it contains implementations for gateway dispatch audit, output re-scanning through ZEF, A2A verification in escalation rules, HMAC identity tokens, DIQ gateway chokepoint enforcement, GO-Gate TTL checks, soul and DIQ integrity checks, AST-based quarantine scanning, audit-chain support, and extensive tests. Remaining uncertainty is concentrated in out-of-tree runtime entry points, tool implementations outside this open-source release, host-level controls, live deployment configuration, and whether public documentation has been updated to reflect the latest implemented controls.

Countermeasures

The following practices were identified based on available documentation and source-code review:

- HMAC identity verification before DIQ lookup, reducing tool-surface disclosure to unauthenticated callers[7][8].
- DIQ role checks, final gateway-enforced execute path, per-agent allowlists, and fail-closed handling when the allowlist subsystem is unavailable.
- ZEF filtering for direct input, high-risk channels, and gateway-level tool-output re-scan with untrusted_content provenance tagging for warning-level content.
- GO-Gate approval records with single-use request IDs, full SHA-256 argument hashes, approval TTL, and atomic status transitions.
- LOCKDOWN and AUTONOMOUS modes, night mode, trust score changes, tripwires, token budgets, and circuit-breaker style controls.
- Memory RBAC, tripwire honeypots, shadow review workflow, audit logging, dispatch correlation IDs, and HMAC audit-chain tamper checks across dispatch, memory-access, and threats tables.
- DIQ tamper checks that hard-fail startup and soul mismatch checks that enter SAFE_MODE and alert administrators when protected content differs from the expected manifest.

[7] https://github.com/kndw-as/super-tanks/blob/main/core/gateway.py
[8] https://github.com/kndw-as/super-tanks/blob/main/core/security/agent_identity.py

- AST-based code quarantine scanner covering imports, attribute access, dynamic execution, getattr obfuscation, dunder probes, alias rebinding, and delayed execution patterns[9].
- Import-time invariant tests for forbidden tools covering the three current agents, including Cody, with extension planned for future agents and new tool profiles.
- trust_score.record_event capability gating through _require_authority(), providing an in-process guard around trust mutations.
- Published security policy, system card, risk register, and incident-response runbooks documenting expected alerts, triage steps, and residual risks[10][11][12][13].

[9] https://github.com/kndw-as/super-tanks/blob/main/core/zeph_quarantine_ast.py
[10] https://github.com/kndw-as/super-tanks/blob/main/SECURITY.md
[11] https://github.com/kndw-as/super-tanks/blob/main/SYSTEM_CARD.md
[12] https://github.com/kndw-as/super-tanks/blob/main/docs/RISK_REGISTER.md
[13] https://github.com/kndw-as/super-tanks/blob/main/docs/INCIDENT_RESPONSE.md

Threat 01: Indirect Prompt Injection and Agent Goal Hijack *(High)*

Risk: High (Likelihood: Medium; Impact: High).
Residual risk: semantic, encoded, transformed, or memory-persisted payloads can still reach agent context after preventive filtering.
OWASP Agentic Mapping:

- ASI01 Agent Goal Hijack (page 9)[14]
- ASI06 Memory & Context Poisoning (page 24)[15]

Super Tanks treats user prompts, tool outputs, web pages, files, memory entries, and A2A content as untrusted inputs. A prompt-injected agent could attempt to reinterpret attacker-controlled content as instructions, alter goals, influence tool selection, or persist malicious context through memory. ZEF and gateway re-scanning reduce this risk, but encoded, semantic, or model-specific payloads remain plausible when content is reintroduced into agent context.

Attack Scenarios

- An attacker places instructions in a web page, file, or memory object that tells Aeris or Zeph to ignore policy, call a tool, disclose data, or create a harmful task after the content is summarized.
- A low-confidence payload trips WARN rather than BLOCK, receives untrusted_content metadata, but a downstream agent loop or prompt template fails to treat provenance metadata as binding.
- A malicious A2A message or tool result combines benign text with delayed instructions that bypass regex patterns and rely on model compliance with hidden task framing.
- A memory-poisoning attempt stores apparently useful facts in an agent-private namespace so later retrieval changes decisions, tool arguments, or trust in external content.

Recommendations

- It is recommended to make provenance metadata a hard input contract in every agent prompt template and tool adapter. This prevents WARN-level external content from being interpreted as trusted instructions.
- It is recommended to expand adversarial testing beyond the current high-signal corpus to cover encodings, multilingual payloads, staged payloads, HTML/Markdown tricks, and tool-output transformation chains. This improves confidence in ZEF and tool-output re-scan behavior.
- It is recommended to require manual review for memory writes derived from external content unless confidence, namespace, and source provenance conditions are all satisfied. This limits long-lived context poisoning.

[14] https://genai.owasp.org/download/52117/
[15] https://genai.owasp.org/download/52117/

- It is recommended to log blocked, warned, and passed tool-output scans with correlation IDs and representative sanitized snippets. This improves incident reconstruction and false-negative triage.

Threat 02: Tool Misuse and Gateway Chokepoint Bypass *(High)*

Risk: High (Likelihood: Medium; Impact: High).
Residual risk: future tools, wrappers, or fallback paths can drift away from the gateway invariant if tests and registration rules are not kept current.
OWASP Agentic Mapping:

- ASI02 Tool Misuse and Exploitation (page 12)[16]

The gateway is the central boundary between agent decisions and tool effects. Identity verification, DIQ role checks, per-agent allowlists, and the DIQTool gateway context are intended to prevent a compromised or prompt-injected agent from invoking tools beyond its role. Any alternate invocation path, missing wrapper, fallback behavior, or registry drift could undermine this boundary.

Attack Scenarios

- A compromised agent attempts to call a tool directly, instantiate a DIQTool subclass, or exploit a plugin fallback path where no DIQ wrapper exists.
- A new tool is registered with an overly permissive required_role or copied into the wrong allowlist, allowing Aeris, Cody, or an unknown agent to gain WRITE, EXEC, or ADMIN reach.
- An attacker influences tool parameters for home_assistant, file_write, shell_exec, python_exec, memory_delete, or code_edit after the high-level tool choice appears legitimate.
- A failure in the allowlist module, DIQ registry, or identity subsystem is treated as availability pressure and accidentally becomes a bypass rather than a denial.

Recommendations

- It is recommended to require every production tool to have a DIQ wrapper and to fail closed when no wrapper exists for high-impact tool names. This prevents silent fallback around the gateway.
- It is recommended to extend existing import-time invariant tests for forbidden tools to future agents and new tool profiles. This prevents allowlist drift during feature additions.
- It is recommended to validate high-impact tool parameters against strict schemas and deployment-specific allow rules after tool selection and before execution. This reduces abuse through legitimate tools.
- It is recommended to monitor denied_identity, denied_role, denied_allowlist, denied_subsystem, and no_wrapper events as security signals. This turns bypass attempts into actionable detection.

[16] https://genai.owasp.org/download/52117/

Threat 03: Agent Identity, A2A Spoofing, and Privilege Boundary Abuse *(Medium)*

Risk: Medium (Likelihood: Medium; Impact: Medium).
Residual risk: host-level key theft or out-of-tree runtime components that trust free-form agent IDs can weaken otherwise sound identity checks.
OWASP Agentic Mapping:

- ASI03 Identity and Privilege Abuse (page 15)[17]
- ASI07 Insecure Inter-Agent Communication (page 27)[18]

Agent identity and inter-agent communication are critical because a forged sender, stolen HMAC token, or confused deputy path could cause one agent to inherit another agent surface. The source tree contains identity signing and A2A verification support, but the full runtime entry point and some tool implementations are outside the open-source release, so production wiring remains a key control point.

Attack Scenarios

- A malicious input claims to be a trusted sender and sends an A2A message that asks Zeph to perform a higher-privilege action on behalf of Aeris or a user.
- An attacker obtains data/.identity_key from the host filesystem and issues valid identity tokens for arbitrary agent IDs.
- A tool or out-of-tree runtime component accepts an agent_id string as authoritative without passing through verify_identity or verify_a2a_message.
- A compromised agent uses ambiguity between user access levels, agent roles, allowed_agents, and DIQ roles to create a confused-deputy action chain.

Recommendations

- It is recommended to make A2A signature verification mandatory at every receive path and keep tests that fail when an unsigned or incorrectly signed message is accepted. This prevents sender spoofing.
- It is recommended to protect identity key material with OS-level controls, file permissions, disk encryption, backup discipline, and separate deployment accounts. This anchors the identity model outside the agent process.
- It is recommended to pass verified identity context, not free-form agent_id strings, across runtime boundaries. This reduces identity confusion between in-tree and out-of-tree components.
- It is recommended to document the precedence between user access levels, agent roles, DIQ required roles, and allowed_agents in the public security model. This reduces ambiguity during future review.

[17] https://genai.owasp.org/download/52117/
[18] https://genai.owasp.org/download/52117/

Threat 04: Code Proposal Quarantine and Runtime Execution Escape *(High)*

Risk: High (Likelihood: Medium; Impact: High).
Residual risk: accepted code and dependency upgrades still execute with deployment privileges unless a per-execution sandbox contains them.
OWASP Agentic Mapping:

- ASI05 Unexpected Code Execution (RCE) (page 21)[19]
- ASI04 Agentic Supply Chain Vulnerabilities (page 18)[20]

Super Tanks uses a code-quarantine flow and AST scanner to prevent dangerous code from reaching the live tree. This is a strong preventive control for known patterns, but it is static. Once a proposal is approved or auto-merged under an allowed mode, accepted code can execute with the runtime privileges of the deployment unless a separate per-execution runtime sandbox contains it. The dependency-upgrade path also runs pip install through subprocess without separate containment.

Attack Scenarios

- A prompt-injected agent proposes a patch that avoids current AST patterns but performs a dangerous runtime action through an allowed library or call sequence.
- A malicious proposal splits behavior across files or staged changes so each individual patch appears harmless but the combined effect creates execution, exfiltration, or persistence.
- A compromised Zeph path or mode transition causes PASS proposals to be approved without adequate human review in AUTONOMOUS mode.
- An attacker targets install scripts, dependency declarations, subprocess-based pip install behavior, or configuration files rather than Python code paths covered by the AST scanner.

Recommendations

- It is recommended to run approved code in a runtime sandbox such as Docker user isolation, nsjail, firejail, seccomp, AppArmor, or equivalent least-privilege containment. This limits damage from scanner false negatives.
- It is recommended to make code-quarantine decisions patch-set aware and evaluate cumulative behavior across all modified files. This reduces staged bypasses.
- It is recommended to require GO-Gate approval for any code-affecting action in LOCKDOWN and for high-risk categories in AUTONOMOUS mode. This preserves human control over executable changes.
- It is recommended to extend scanner and review coverage to install scripts, dependency manifests, subprocess-based upgrade paths, workflow files, and configuration that can influence execution. This improves supply-chain resilience.

[19] https://genai.owasp.org/download/52117/
[20] https://genai.owasp.org/download/52117/

Threat 05: Soul and DIQ Integrity Manifest Tampering *(Medium)*

Risk: Medium (Likelihood: Low; Impact: High).
Residual risk: deployment-specific manifests in a writable tree can be modified together with protected files unless an external trust anchor exists.
OWASP Agentic Mapping:

- ASI04 Agentic Supply Chain Vulnerabilities (page 18)[21]
- ASI10 Rogue Agents (page 36)[22]

Soul files and DIQ contracts define agent identity, capability boundaries, and tool interface invariants. In the reviewed implementation, DIQ tampering hard-fails startup, while soul mismatches enter SAFE_MODE and trigger administrator alerting. The main residual risk arises if a deployment-specific manifest resides in the writable tree alongside the files it protects, allowing a local write-capable attacker to update protected files and manifests together unless an external trust anchor exists.

Attack Scenarios

- A malicious local process modifies a soul file and soul_integrity.json in the same filesystem transaction to preserve matching hashes when the manifest resides in the writable deployment tree.
- A compromised dependency or install step updates DIQ contracts and DIQ_CHECKSUMS.json before startup, making unauthorized capabilities appear expected.
- An attacker causes operators to approve a soul or DIQ mismatch as a maintenance event without sufficient provenance review.
- A backup, restore, or deployment sync process rolls back code and manifests to an older but valid state with weaker controls.

Recommendations

- It is recommended to sign soul and DIQ manifests with an offline key, TPM, HSM, or release-signing key stored outside the writable runtime tree. This prevents same-filesystem manifest forgery.
- It is recommended to include manifest version, commit, build timestamp, and release provenance in integrity checks. This prevents valid but stale rollback states.
- It is recommended to require a documented operator checklist before exiting SAFE_MODE after an integrity alert. This reduces social-engineering and maintenance confusion.
- It is recommended to alert on both mismatches and unexpected manifest regeneration events. This detects tampering attempts that try to hide behind refreshed checksums.

[21] https://genai.owasp.org/download/52117/
[22] https://genai.owasp.org/download/52117/

Threat 06: Audit, Trust, and Approval Evidence Integrity Gaps *(Medium)*

Risk: Medium (Likelihood: Medium; Impact: Medium).
Residual risk: trust and approval evidence remain less tamper-evident than chained stores, and shared key material links identity compromise to audit compromise.
OWASP Agentic Mapping:

- ASI09 Human-Agent Trust Exploitation (page 33)[23]
- Evidence integrity is treated as a cross-cutting agentic control concern.

Super Tanks relies on auditability to support incident response, human oversight, and non-repudiation. The source tree includes dispatch audit, memory audit, trust state, approval requests, HMAC audit-chain support, and a partial in-process authority gate for trust mutations. The remaining threat is coverage, key separation, correlation, and documentation drift: if audit rows are incomplete, tampered, or described inconsistently in public materials, maintainers may draw incorrect conclusions after a security event.

Attack Scenarios

- An attacker with filesystem write access edits SQLite rows to hide an action, approval, trust mutation, or memory operation.
- An attacker who steals data/.identity_key can issue identity tokens and weaken audit-chain evidence integrity because the audit-chain key is derived from the identity key.
- A compromised in-process component calls trust_score.record_event or related mutation functions without a verified caller capability.
- Public SECURITY, SYSTEM_CARD, and RISK_REGISTER content becomes inconsistent with implemented controls, leading deployers to overestimate or underestimate residual risk.

Recommendations

- It is recommended to extend the existing chained HMAC audit model from dispatch, memory-access, and threats tables to the trust and approval databases. This protects evidence integrity across all human and agent control points.
- It is recommended to separate identity-token signing keys from audit-chain HMAC keys. This prevents one key compromise from affecting both authentication and evidence integrity.
- It is recommended to make correlation IDs mandatory across every side-effecting subsystem and fail tests when an event cannot be joined back to a dispatch. This improves incident reconstruction.
- It is recommended to strengthen the existing in-process trust mutation authority gate with tests and boundaries that prevent untrusted code from opening the authority window. This reduces trust inflation from compromised components.

[23] https://genai.owasp.org/download/52117/

- It is recommended to update public security documentation whenever a control is implemented, narrowed, or superseded. This keeps the published threat model accurate.

Threat 07: Operational Mode, Sandbox, and Availability Control Failures *(Medium)*

Risk: Medium (Likelihood: Medium; Impact: Medium).
Residual risk: availability pressure, in-flight actions, and host-level state manipulation can still exceed process-level controls.
OWASP Agentic Mapping:

- ASI08 Cascading Failures (page 30)[24]
- ASI10 Rogue Agents (page 36)[25]

Mode transitions, GO-Gate, token budgets, circuit breakers, night mode, and safe-mode behavior are designed to fail closed. However, availability and containment remain important because an attacker can try to exhaust budgets, trigger repeated approvals, keep long-running calls alive after LOCKDOWN, or abuse local deployment assumptions outside the Super Tanks process.

Attack Scenarios

- A malicious input triggers repeated LLM calls, scan attempts, approval prompts, or tool retries to exhaust token budgets or operator attention.
- A long-running tool call begins in AUTONOMOUS mode and continues after LOCKDOWN is activated because the control only affects new dispatches.
- An attacker manipulates system time, timezone configuration, or state files to influence night mode, scheduled health checks, or AUTONOMOUS timeout behavior.
- A host-level compromise reads identity keys, rewrites databases, modifies source, or disables containers, bypassing Super Tanks process-level controls entirely.

Recommendations

- It is recommended to implement cooperative cancellation and process-level kill controls for in-flight high-impact tool calls when LOCKDOWN or SAFE_MODE activates. This reduces residual blast radius.
- It is recommended to rate-limit approval prompts, failed scans, denied dispatches, and repeated model calls per agent and per user. This mitigates denial-of-service and operator fatigue.
- It is recommended to protect mode state, schedule configuration, and key databases with least-privilege filesystem permissions and integrity monitoring. This prevents local state manipulation.
- It is recommended to document host-level hardening prerequisites, including file permissions, separate service users, encrypted disks, backup protection, and Docker isolation settings. This clarifies the trust boundary outside Super Tanks.

[24] https://genai.owasp.org/download/52117/
[25] https://genai.owasp.org/download/52117/

Conclusion

The Super Tanks solution will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Super Tanks application provided a number of positive impressions during this assignment that must be mentioned here:

- The documented architecture maps cleanly to STRIDE and the OWASP Agentic Top 10 ASI categories, with threat boundaries reflected in implemented controls rather than only design prose.
- Gateway chokepoint enforcement, fail-closed DIQ handling, A2A verification, GO-Gate atomic transitions, ZEF re-scanning, code quarantine, SAFE_MODE behavior, and audit-chain support are positive controls for an AI-agent governance layer.
- The project maintains substantial automated coverage, with 1398 tests reported as passing.

The security of the Super Tanks solution will improve with a focus on the following areas:

- Add per-execution runtime sandboxing for approved code and the dependency-upgrade path.
- Separate identity-token and audit-chain key material, and extend tamper-evident audit chaining to trust and approval stores.
- Externally anchor or sign soul and DIQ integrity manifests, and document deployment hardening prerequisites for identity keys, manifests, databases, and mode state.
- Keep public security documentation, ASI mapping, and residual-risk language synchronized with implementation changes.

Once the threat model weaknesses have been addressed with additional security controls, a more thorough review, ideally including a full source code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Super Tanks resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank William Park and the rest of the Super Tanks team, for their exemplary assistance and support throughout this audit.