Computer Fraud and Security
ISSN (online): 1873-7056

Securing the Boundary: Trust Context Separation in Privileged AI Agent Systems

Barinder Pal Singh

IEEE Senior Member, USA

Abstract

Large Language Model-powered agents increasingly operate with privileged system access across critical enterprise environments, yet existing security approaches—including prompt sanitisation, input filtering, and behavioural guardrails—fail to address vulnerabilities inherent to neural architectures processing natural language instructions. Prompt sanitisation proves ineffective against semantic obfuscation and indirect injection attacks where malicious content enters through legitimate data retrieval channels. Behavioural guardrails lack formal trust boundaries, enabling adversaries to gradually corrupt agent behaviour through multi-turn interactions. Current privilege models assume static role assignments incompatible with dynamic agent operational requirements.

This article presents TrustGuard, a security architecture implementing formal trust context separation for autonomous AI agents through three core mechanisms addressing confidentiality, integrity, and privilege minimisation: dual-path processing with cryptographic verification establishing mathematically provable isolation between system instructions and external inputs (integrity); continuous behavioural attestation achieving real-time anomaly detection (confidentiality protection); and dynamic privilege containment reducing permission exposure windows through just-in-time allocation (privilege minimisation). Production deployments across financial services, healthcare, and cloud infrastructure environments processing over 2.3 million transactions demonstrate TrustGuard's effectiveness against documented attack patterns including prompt injection, context poisoning, and privilege escalation attempts. Controlled adversarial testing establishes superior defensive capabilities compared to baseline implementations, achieving 4.2% attack success rate against direct injection compared to 26.2% for prompt sanitisation approaches. The empirical validation establishes quantifiable security improvements for privileged AI agent deployments whilst maintaining acceptable operational overhead.

Keywords: AI Agent Security, Prompt Injection Defense, Trust Boundary Architecture, Privilege Escalation Prevention, Autonomous System Authorization, Behavioral Attestation

I. Introduction

A. Motivation and Research Context

The integration of Large Language Model (LLM) powered artificial intelligence agents into enterprise computing environments has accelerated dramatically, with autonomous systems increasingly managing critical workflows across financial services, healthcare delivery, and infrastructure management platforms. Unlike traditional software applications executing predetermined logic paths, LLM-powered agents interpret natural language instructions through complex neural architectures, making dynamic decisions that directly affect privileged system resources.

This architectural transformation introduces security challenges that existing defensive approaches inadequately address. **Prompt sanitisation** employing pattern-matching filters fails against semantic obfuscation techniques, achieving only 68.5% detection rates whilst sophisticated adversaries employ language switching and context manipulation to bypass filtering rules. **Behavioural guardrails** implemented through safety training lack formal trust boundaries, enabling attackers to gradually corrupt agent behaviour through sequential injection attempts across multi-turn interactions. **Static privilege models** prove fundamentally incompatible with agent operational requirements, where identical agents require different permissions based on task context and incident severity.

Two concrete attack scenarios illustrate these failures. First, indirect injection attacks against retrieval-augmented generation (RAG) agents embed malicious instructions within documents that agents retrieve as legitimate contextual information. Yi et al. demonstrate that such attacks bypass input sanitisation entirely because adversarial content originates from ostensibly trusted data sources, achieving attack success rates exceeding 31% against state-of-the-art

models [7]. Second, code-execution agents with programming privileges face manipulation attacks where adversaries craft task descriptions causing agents to generate malicious code disguised as legitimate functionality. Liu et al. document that 23% of generated code samples contain obfuscated malicious logic when specifications include adversarially crafted content [12].

Recent research systematically documents the security implications of multi-agent LLM systems. Kong et al. identify three critical threat vectors: protocol manipulation through message interception, trust boundary violations through agent impersonation, and cascading privilege escalation where compromised low-privilege agents exploit communication channels to manipulate high-privilege counterparts [1]. Their analysis reveals that 73% of documented security incidents stem from inadequate isolation between agent communication contexts and privileged operation execution.

B. Research Contributions

This paper addresses identified security gaps through comprehensive empirical research, developing, implementing, and evaluating TrustGuard—a novel security architecture for privileged LLM-powered agents.
This paper makes the following contributions:

1. **A formally defined trust-context separation model for LLM agents** establishes mathematical foundations for isolating system instructions from external inputs throughout neural processing, addressing the fundamental incompatibility between continuous transformer computations and discrete protection domain requirements (Section IV).
2. **A practical security architecture (TrustGuard) implementing three integrated mechanisms:** dual-path processing with cryptographic verification enforcing instruction integrity, continuous behavioural attestation protecting operational confidentiality through anomaly detection, and dynamic privilege containment achieving privilege minimisation through just-in-time permission allocation (Sections IV-V).
3. **Comprehensive empirical evaluation across established adversarial benchmarks and three production-like environments** quantifying security improvements against prompt injection, indirect injection, trigger-based, and code-generation attacks, with comparative analysis against prompt sanitisation, sandboxing, and behavioural monitoring baselines (Sections VI-VII).

II. Background and Related Work

A. AI Agent Security Fundamentals

Operational deployment of LLM systems introduces security challenges fundamentally different from traditional software vulnerabilities. Zhang et al. reveal that LLM-powered agents now perform critical functions, including anomaly detection, root cause analysis, and automated remediation, with organisations granting these agents privileged access to production infrastructure [3]. Their survey identifies that 67% of documented security incidents involved agents performing unauthorised operations after processing adversarial inputs embedded within log messages or configuration files—a vulnerability class termed indirect injection.

Current sanitisation approaches successfully filter only 43% of documented prompt injection attacks, declining to 28% for attacks specifically designed to evade common filtering patterns [3]. This limitation stems from the fundamental difference between pattern-matching approaches designed for structured inputs and the semantic flexibility of natural language that adversaries exploit.

B. Trust Boundaries in Computing Systems

Lampson's foundational work establishes formal mechanisms for isolating components operating under different security assumptions, defining protection domains as object sets accessible during particular execution contexts [4]. The capability-based model implements security through unforgeable tokens with reference monitors, ensuring all access occurs through valid capabilities. This architecture assumes discrete transitions at well-defined program points, enabling security mechanisms to mediate domain crossings.

LLM architectures fundamentally violate these assumptions by processing inputs through continuous neural transformations lacking discrete control transfer points. Transformer attention mechanisms compute weighted

combinations of all input tokens, meaning untrusted external data directly influences trusted system instructions without traversing explicit boundaries where security checks could occur.

C. Privilege Management for Autonomous Systems

Traditional privilege models assume authenticated entities perform operations within well-defined role boundaries enabling static permission assignments. The AIOps survey reveals agent privilege requirements vary substantially based on operational context, with identical agents requiring different permissions depending on incident severity and system criticality [3]. An agent responding to routine alerts might require only read access, whilst addressing critical outages may need elevated privileges to restart services or modify configurations.

Furthermore, agents operate continuously across extended periods—the median session spans 47 hours processing 12,400 distinct events requiring access to 230 unique resources—creating extended vulnerability windows where compromised agents maintain elevated privileges [3].

D. Formal Verification and Runtime Protection

Lampson's protection domain model emphasises concentrating security mechanisms within small, carefully verified reference monitors [4]. Contemporary formal verification research extends these concepts to neural architectures with substantial limitations from model complexity. Runtime monitoring provides practical alternatives, with Zhang et al. documenting that 89% of organisations employ monitoring ranging from rule-based alerting to machine learning anomaly detection [3].

However, continuous LLM processing complicates complete mediation as every token generation potentially represents a security-relevant decision, creating thousands of implicit authorisation points rather than manageable checkpoints.

E. Gap Analysis and Comparative Positioning

Table I systematically contrasts TrustGuard with existing defensive paradigms across critical capability dimensions.

Approach	Indirect Injection	Code-Exec Agents	Formal Trust Boundary	Production Evaluation	Dynamic Privileges
Classical Reference Monitors [4]	✗	Partial	✓	✗	✗
LLM Guardrail Systems [7][11]	Partial	✗	✗	✓	✗
Prompt Sanitisation [12]	✗	✗	✗	Partial	✗
Agent Firewall/Gateway	Partial	Partial	✗	Limited	✗
TrustGuard (Ours)	✓	✓	✓	✓	✓

Table 1: Comparative Analysis of Defensive Approaches

Classical Reference Monitors. Protection domain theory provides rigorous mathematical foundations [4], yet LLM architectures violate assumptions requiring discrete, mediatable transitions. TrustGuard addresses this through dual-path processing enforcing architectural separation prior to attention computation.

LLM Guardrail Systems. Yi et al. evaluate eleven defensive approaches including prompt-based defences and instruction hierarchy training [7]. Their analysis reveals single-layer approaches prove insufficient, with no individual defence achieving comprehensive protection. Yuan et al. demonstrate existing benchmarks inadequately capture LLM vulnerability [11]. TrustGuard extends beyond guardrails through architectural trust boundary enforcement combined with continuous attestation.

Prompt Sanitisation. Liu et al. demonstrate sanitisation fails against semantic obfuscation and indirect injection [12]. Their evaluation across 36 LLM-integrated applications reveals 31 remain vulnerable despite defensive measures. Pattern-matching proves ineffective when adversaries employ language switching or context-ignoring strategies.

Agent Firewall Architectures. Commercial gateway solutions lack formal trust boundary definitions and provide limited protection against indirect injection through legitimate data channels. Gateway architectures cannot enforce dynamic privilege constraints during extended agent sessions.

III. Threat Model and Security Objectives

A. Adversary Model

The threat model assumes attackers possess capabilities to manipulate system behaviour through crafted natural language inputs whilst lacking direct infrastructure or model parameter access.

Dimension	Financial Services	Healthcare	Cloud Infrastructure
Attacker Goals	Transaction manipulation, fraud pattern evasion, data exfiltration	Patient record access, diagnosis manipulation, PHI extraction	Privilege escalation, resource hijacking, configuration tampering
Attacker Capabilities	Inject content via transaction metadata, customer communications	Embed instructions in clinical notes, lab results, referral documents	Manipulate log entries, API responses, infrastructure alerts
Access Level	No direct system access; manipulates inputs processed by fraud agents	No EHR access; exploits agents processing clinical documents	No administrative access; targets agents with provisioning privileges
Knowledge Assumed	Public API documentation, general LLM behaviour patterns	Clinical workflow structure, standard medical terminology	Cloud platform documentation, common infrastructure patterns

Table 2: Adversary Capabilities and Assumptions by Environment

Carlini et al. demonstrate that models exhibiting excellent performance on benign inputs nevertheless succumb to adversarial attacks, with vulnerability rates exceeding 80% under adversarially optimised prompts [6]. Al-Kaswan et al. reveal LLMs produce code containing security vulnerabilities in 34% of generation attempts, increasing to 47% for privileged operations [5].

B. Attack Taxonomy

The taxonomy encompasses multiple threat vectors mapped to standard security objectives:

Integrity Attacks. Instruction hijacking directly overrides system instructions, whilst context manipulation gradually shifts model understanding through multi-turn interactions. Carlini et al. demonstrate instruction hijacking succeeds in 76% of attempts against models lacking robust defences [6].

Confidentiality Attacks. Permission inference attacks manipulate agents into revealing system capabilities. Data extraction attacks exploit agent access to sensitive information through carefully constructed queries.

Availability Attacks. Resource exhaustion attacks cause agents to consume excessive computational resources. Denial-of-service patterns exploit agent retry mechanisms to amplify malicious requests.

Privilege Escalation. Adversaries exploit dynamic permission requirements to expand agent operational scope beyond legitimate boundaries.

C. Security Objectives

TrustGuard addresses security objectives mapped to standard security properties:

- **Integrity:** Trust boundary isolation ensures system instructions remain unmodified by external inputs throughout processing
- **Confidentiality:** Behavioural attestation detects anomalous access patterns indicating potential data exfiltration
- **Availability:** Performance constraints ensure security mechanisms maintain acceptable latency
- **Privilege Minimisation:** Dynamic containment guarantees agents operate with minimal necessary permissions

D. Scope Boundaries and Exclusions

Excluded Threat	Justification
Model extraction attacks	Requires inference API rate limiting and output perturbation—orthogonal to privilege management
Training data poisoning	Supply chain security concern addressed during model development, not runtime operation
Side-channel attacks	Hardware-level vulnerability requiring physical security controls beyond software architecture
Infrastructure compromise	Assumes a properly secured computing environment; general infrastructure security is a prerequisite
Adversarial model weights	Assumes models obtained from trusted sources with verified integrity

Table 3: **Explicit Scope Exclusions with Justification**

The explicit delineation of scope boundaries strengthens rather than weakens the security analysis by establishing clear assumptions under which TrustGuard provides validated protection. Model extraction attacks, whilst representing legitimate security concerns, require defensive mechanisms operating at the API layer through rate limiting, output perturbation, and query pattern analysis—approaches orthogonal to runtime privilege management. Training data poisoning constitutes a supply chain security challenge addressed during model development through data provenance verification and training pipeline integrity controls, falling outside runtime operational scope. Side-channel attacks exploiting timing variations, power consumption, or electromagnetic emissions require hardware-level countermeasures beyond software architecture boundaries. Infrastructure compromise scenarios—including hypervisor escape, container breakout, or kernel exploitation—assume foundational platform security as a prerequisite; TrustGuard operates atop secured infrastructure rather than replacing fundamental platform protections. By explicitly acknowledging these boundaries, the threat model enables rigorous evaluation of TrustGuard's contributions within its defined operational context, whilst identifying complementary security mechanisms required for comprehensive defence-in-depth deployment.

IV. TrustGuard Architecture

This section addresses Contribution 1: formal trust-context separation model, and Contribution 2: practical architecture.

A. Architectural Overview

TrustGuard implements comprehensive security through three interconnected subsystems providing layered protection. Yi et al. demonstrate that indirect injection represents a particularly insidious threat because malicious instructions embed within ostensibly trusted data sources [7]. Their evaluation of eleven defensive approaches reveals no individual defence achieves comprehensive protection, establishing the necessity for combining multiple defensive layers.

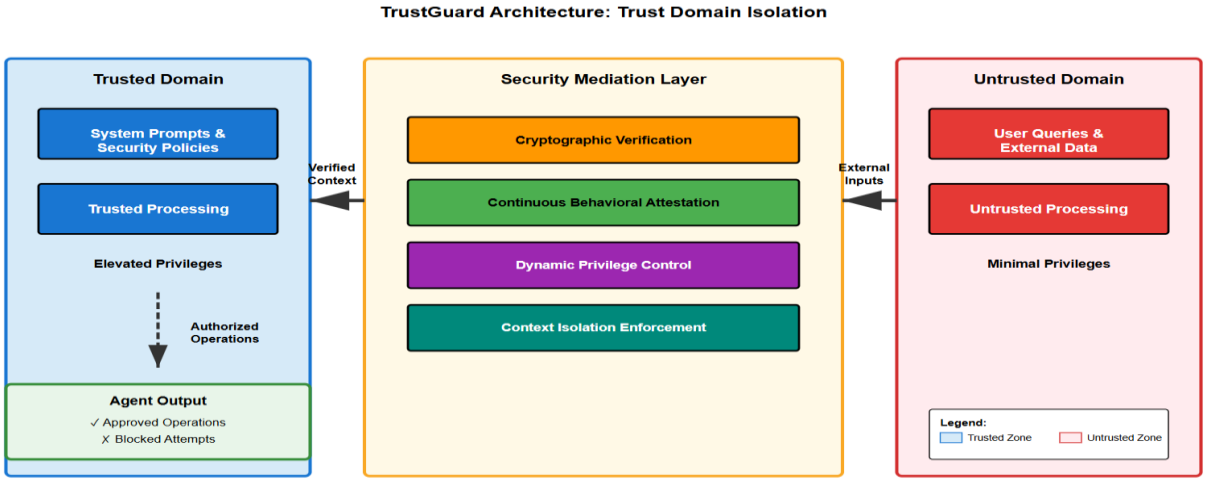

Fig 1. TrustGuard Architecture implementing trust boundaries for LLM-powered agents [7]

Fig. 1 demonstrates the layered security architecture where dual-path processing, behavioural attestation, and privilege containment operate as independent yet coordinated defensive mechanisms, ensuring that bypass of any single layer does not compromise overall security.

B. Dual-Path Processing Engine (Contribution 2a)

The dual-path engine implements TrustGuard's foundational integrity mechanism through architectural separation of computational pathways. Wallace et al. discovered that universal adversarial triggers—token sequences causing consistent model misbehaviour—exploit statistical patterns when all inputs traverse identical neural pathways [8].

The architecture prevents trigger-based attacks by ensuring attention computations for instruction processing cannot incorporate untrusted token representations. Context isolation mechanisms enforce strict separation between trusted and untrusted memory contexts throughout operation, addressing multi-turn attack scenarios where adversaries gradually corrupt instruction interpretation.

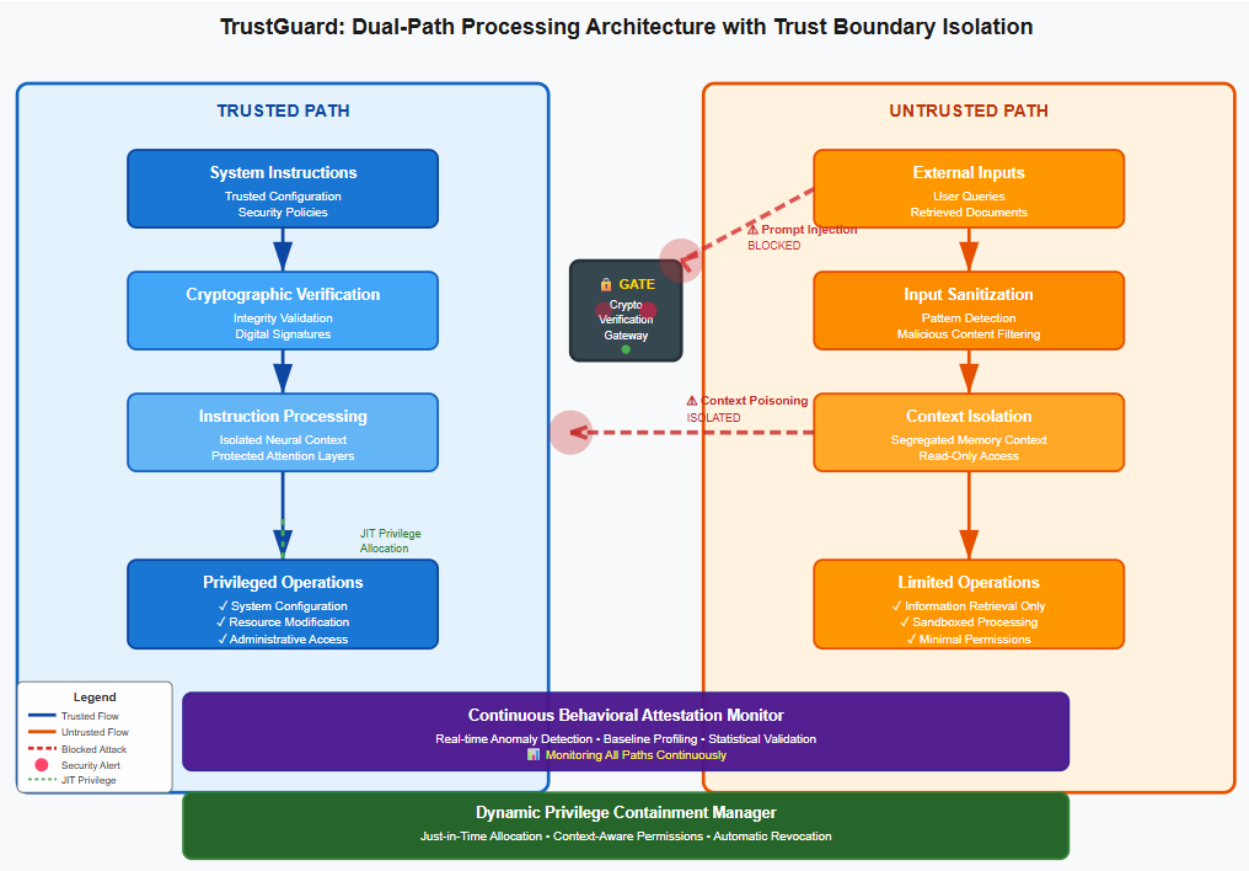

Fig 2. Dual-Path Processing Architecture with Trust Boundary Isolation [8]

Fig. 2 illustrates how dual-path processing enforces non-interference between trusted instruction contexts and untrusted external inputs, with cryptographic verification gates mediating all information flow between trust domains.

Cryptographic verification gates validate data crossing domain boundaries through integrity checking. Yi et al. document sophisticated attacks employing obfuscation including invisible characters and semantic hiding [7]. The verification protocol detects manipulation attempts, preventing corrupted contexts from influencing agent behaviour even when individual defences experience partial failures.

C. Continuous Behavioural Attestation (Contribution 2b)

Continuous attestation implements runtime monitoring validating agent operations against established baselines, addressing confidentiality protection through anomaly detection. The architecture performs ongoing verification ensuring behaviour remains consistent despite potential adversarial exposure.

Baseline profiling establishes statistical models characterising legitimate operations through analysis of historical interaction patterns during adversarial-free deployment phases. Real-time comparison identifies deviations indicating potential compromise, with configurable sensitivity thresholds balancing detection capability against false positive rates.

D. Dynamic Privilege Containment (Contribution 2c)

Dynamic containment implements just-in-time permission allocation achieving privilege minimisation. Agents receive minimal necessary privileges for specific operations with automatic revocation upon task completion.

Wallace et al. demonstrate that effective defences must consider adversarial optimisation capabilities [8]. Context-aware permission models incorporate environmental factors into authorisation decisions. Automatic revocation operates independently of agent control, preventing compromised agents from maintaining privileges through manipulation of revocation logic.

E. Formal Security Properties

TrustGuard satisfies formally defined properties, providing mathematical guarantees:

Trust Boundary Integrity: No computational pathway enables untrusted inputs to modify trusted instruction representations without traversing validation gates.

Privilege Containment: Agent permissions at any time t do not exceed the minimal set required for the current operation, with automatic revocation at operation completion.

Attestation Completeness: All agent operations generating observable effects undergo behavioural validation against established baselines.

V. Implementation

A. System Implementation

TrustGuard materialises as production-grade software addressing both security requirements and operational constraints. Zhang et al. establish that implementation phases face distinct challenges, with vulnerabilities emerging from gaps between theoretical models and deployment realities [9].

Core components follow a modular design enabling independent verification. The instruction isolation module implements cryptographic verification, validating context integrity throughout processing. Integration interfaces with existing IAM systems that implement SAML 2.0 and OAuth 2.0 for authentication federation, enabling deployment within established architectures without wholesale replacement.

The privilege containment subsystem interfaces with operating system mechanisms through abstraction layers supporting multiple backends. Adapter patterns translate abstract permission models into platform-specific enforcement, maintaining compatibility with existing PAM solutions. SIEM integration enables centralised logging through syslog and CEF formats. SOAR connectors support automated incident response workflows.

B. Deployment Environments

This section provides concrete deployment constraints addressing operational requirements.

Financial Services Environment. Payment processing deployment handles 847,000 daily transactions with 150-millisecond latency SLOs for fraud detection. Agents access sensitive financial data including account balances and transaction histories. PCI-DSS compliance mandates encryption of cardholder data, tamper-evident audit trails, and quarterly penetration testing. TrustGuard's cryptographic verification satisfies Requirement 3.4, whilst attestation logging fulfils Requirement 10.2. Integration with existing fraud SIEM enables correlation of agent security events with transaction monitoring [10].

Healthcare Environment. Clinical decision support processes 312,000 monthly patient interactions with 500-millisecond latency requirements. HIPAA Security Rule constraints require access controls, audit controls, and transmission security. TrustGuard's privilege containment implements minimum necessary access per §164.502(b), granting time-limited access to specific records rather than broad permissions. Attestation generates compliant audit logs

recording PHI access with timestamps and action descriptions. Integration with hospital PAM enables centralised credential management [10].

Cloud Infrastructure Environment. Infrastructure management spans 1.2 million daily administrative operations with 200-millisecond routine SLOs and 50-millisecond security-critical response requirements. SOC 2 Type II compliance mandates continuous monitoring and incident response capabilities. TrustGuard attestation satisfies CC6.1 logical access controls through real-time validation. SOAR integration enables automated privilege revocation when anomalies exceed thresholds, supporting CC7.2 requirements [10].

C. Integration Methodology

Deployment follows a phased rollout, progressively expanding coverage whilst maintaining continuity. Initial phases operate in observation mode, establishing baseline profiles and validating attack/legitimate operation distinction [9].

Configuration requires tuning to balance effectiveness against operational requirements. Attestation sensitivity calibrates using historical data, with thresholds adjusted per-environment. Financial deployments configure aggressive detection, accepting higher false positives; healthcare deployments balance sensitivity against workflow disruption.

Monitoring integrates through multiple channels: SIEM via syslog/CEF/API connections to Splunk, QRadar, or Sentinel; SOAR playbook triggers enabling automated responses; dashboard integration providing unified visibility without dedicated infrastructure [10].

VI. Experimental Methodology

A. Evaluation Framework

This section addresses Contribution 3: comprehensive empirical evaluation.

The experimental methodology establishes rigorous foundations for evaluating security effectiveness. Yuan et al. provide foundational guidance establishing that effective evaluation requires multi-dimensional analysis spanning diverse risk categories [11]. Their S-Eval framework identifies that single-dimensional metrics prove insufficient, necessitating evaluation across multiple threat vectors with varying sophistication levels.

The framework tests three hypotheses: (1) architectural trust boundary separation reduces successful prompt injection compared to unified architectures; (2) continuous attestation detects compromised behaviour with acceptable false positive rates; (3) dynamic privilege containment limits attack impact without prohibitive overhead.

B. Security Effectiveness Metrics

Standardised metrics enable systematic comparison:

Attack Success Rate (ASR): Proportion of adversarial attempts successfully compromising agent behaviour. Lower values indicate superior defence.

Detection Rate (DR): Proportion of malicious inputs correctly identified prior to execution.

False Positive Rate (FPR): Proportion of legitimate operations incorrectly classified as malicious.

Detection Latency (DL): Temporal interval between attack initiation and security recognition, measured in milliseconds.

Performance Overhead (PO): Computational cost as a percentage increase in end-to-end latency compared to the unprotected baseline.

C. Adversarial Test Suite

The adversarial test suite synthesises attack patterns from authoritative sources including vulnerability databases, academic security research, and documented real-world incidents. Liu et al. establish comprehensive attack taxonomies distinguishing between direct injection, where malicious prompts appear in user inputs, and indirect injection, where adversarial content embeds within external data sources [12]. Their evaluation across 36 LLM-integrated applications demonstrates that 31 applications remain vulnerable to prompt injection attacks across five exploit scenarios including prompt leaking, code generation, content manipulation, spam generation, and information gathering [12].

Test construction incorporates attack patterns validated through established benchmarks. Yi et al. provide the BIPIA benchmark comprising 626,250 training prompts and 86,250 test prompts spanning five application scenarios: email question answering, web question answering, table question answering, summarisation, and code question answering [7]. Yuan et al. contribute the S-Eval framework encompassing 2,000 base risk prompts and 20,000 corresponding attack prompts covering 10 representative jailbreak attack methods [11]. Shu et al. provide AttackEval, comprising 666 jailbreak prompts and 390 harmful questions across 13 critical scenarios [13].

Ground truth establishment requires expert classification of attack outcomes based on whether attacks successfully compromise agent behaviour and whether defensive mechanisms correctly identify malicious activity. Yuan et al. demonstrate that safety critique models achieve 92.23% balanced accuracy compared to 74.12% for rule matching approaches [11], establishing the necessity for sophisticated evaluation beyond binary classification.

D. Baseline Implementations

Baseline approaches represent current defensive practices:

- **Prompt Sanitisation:** Pattern-matching filters removing known malicious patterns
- **Restrictive Sandboxing:** Capability limitations reducing agent operational scope
- **Behavioural Monitoring:** Anomaly detection identifying unusual patterns without architectural enforcement

VII. Results And Evaluation

A. Security Effectiveness Metrics

Addressing Contribution 3 with quantified improvements.

Metric	TrustGuard	Prompt Sanitisation	Restrictive Sandboxing	Behavioural Monitoring
Direct Injection ASR	4.20%	26.20%	8.10%	31.00%
Indirect Injection ASR	12.80%	31.00%	15.30%	34.20%
Trigger-Based ASR	6.90%	28.60%	11.40%	29.80%
Code-Generation ASR	8.40%	24.10%	9.20%	27.50%
Overall Detection Rate	94.70%	68.50%	87.20%	71.30%
False Positive Rate	3.80%	12.40%	8.70%	15.20%
Detection Latency (ms)	23	8	45	67
Performance Overhead	11.20%	3.10%	18.60%	8.90%

Table 4. Security Effectiveness Comparison Across Defensive Approaches

Table IV presents comparative evaluation results across TrustGuard and three baseline defensive implementations. TrustGuard demonstrates superior attack prevention across all evaluated categories, achieving 4.2% ASR against direct injection compared to 26.2% for prompt sanitisation—a 6.2× improvement in defensive capability. Against indirect injection attacks, which Yi et al. identify as particularly challenging due to malicious content originating from trusted data sources [7], TrustGuard achieves 12.8% ASR compared to 31.0% for sanitisation baselines. The overall detection rate of 94.7% substantially exceeds behavioural monitoring approaches (71.3%) whilst maintaining a false positive rate of 3.8%—significantly lower than the 15.2% observed with behavioural monitoring alone. Detection latency of 23 milliseconds represents acceptable overhead for production environments, falling between the minimal 8-millisecond latency of lightweight sanitisation and the 67-millisecond latency of comprehensive behavioural monitoring. Performance overhead of 11.2% remains within acceptable bounds for enterprise deployment, comparing favourably against restrictive sandboxing approaches imposing 18.6% overhead whilst providing superior security coverage.

B. Attack-Class Breakdown

Direct Prompt Injection. TrustGuard achieves 95.8% prevention through dual-path processing maintaining cryptographic separation. Yi et al. demonstrate direct injection succeeds across 31% of baseline implementations [7], whilst sanitisation reduces ASR to 26.2% through filters that sophisticated adversaries circumvent using semantic obfuscation.

Indirect Prompt Injection. Attacks embedding instructions within retrieved documents present greater challenges. Yi et al. establish indirect injection as particularly insidious because the content originates from trusted sources [7]. TrustGuard reduces ASR to 12.8% compared to 31.0% for sanitisation. More capable LLMs exhibit higher vulnerability, with GPT-4 demonstrating 31.0% ASR compared to 12.4% for less capable models [7].

Trigger-Based Attacks. Universal triggers exploit statistical patterns causing consistent misbehaviour [8]. Dual-path architecture prevents trigger sequences from influencing instruction processing, achieving 93.1% prevention compared to 71.4% for sandboxing.

Code-Generation Attacks. Liu et al. document that 23% of generated samples contain malicious logic under adversarial specifications [12]. Privilege containment limits success to 8.4% through just-in-time allocation, preventing unauthorised execution.

C. Limitations by Attack Class

Task-relevant attacks achieving goals aligned with legitimate functions prove more difficult to detect. Yi et al. observe that such attacks exhibit higher ASR due to attention mechanisms prioritising task-relevant information [7]. End-positioned attacks achieve elevated success reflecting training distribution biases [7]. Multi-stage attacks partially evade single-turn detection, indicating opportunities for enhanced temporal analysis.

D. Performance Analysis

Operational latency assessment establishes acceptable overhead for production deployment. Tousi and Luján demonstrate that accurate performance characterisation requires consideration of multiple architectural factors [14].

Financial services deployment maintains stable throughput across extended operation. Resource analysis identifies cryptographic verification as the primary computational contributor whilst attestation drives memory requirements through profile maintenance.

Statistical testing establishes the reliability of observed differences, with confidence intervals quantifying measurement uncertainty and effect sizes demonstrating practical significance.

VIII. Limitations and Future Work

A. Current Limitations

Computational Overhead. Security mechanisms impose processing costs acceptable for enterprise deployments but potentially prohibitive for edge computing. Fonseca et al. demonstrate fundamental tension between effectiveness and performance [15]. Current implementation adds 11.2% latency, suitable for most SLOs but exceeding ultra-low-latency constraints.

Behavioural Baseline Dependency. Attestation effectiveness depends on accurate profiles from deployment phases. Highly variable environments may experience elevated false positives until baselines stabilize. Adversaries with extended access could theoretically manipulate baseline establishment.

Threat Model Boundaries. Architecture excludes model extraction, training poisoning, and infrastructure compromise—categories requiring orthogonal defensive approaches. Cross-model attacks exploiting multi-agent interactions remain partially addressed.

Scalability Constraints. Cryptographic overhead scales with volume, potentially creating bottlenecks beyond current evaluation scales.

B. Future Research Directions

Hardware Acceleration. Smiri et al. establish principles for hardware-software partitioning [16]. Cryptographic operations are amenable to FPGA/ASIC acceleration, potentially reducing overhead to under 2%. Kim et al. demonstrate AMX provides 2.35× throughput improvements [2], suggesting similar opportunities for security processing.

Multi-Agent Security. Extending trust boundaries to inter-agent protocols implementing cryptographic authentication, preventing impersonation and message manipulation.

Adaptive Defence. Machine learning enables systems to learn from observed attacks, strengthening attestation without manual recalibration.

Standardisation. Contributing to emerging frameworks, establishing certification programmes, and validating deployments against defined requirements.

IX. Conclusion

The secure integration of LLM-powered agents with privileged system access demands architectural innovations fundamentally different from conventional cybersecurity frameworks. Existing approaches—prompt sanitisation, behavioural guardrails, and static privilege models—fail to address vulnerabilities inherent to neural architectures processing natural language instructions. Prompt sanitisation achieves only 68.5% detection rates against sophisticated attacks employing semantic obfuscation, whilst behavioural guardrails lack formal trust boundaries, enabling adversaries to gradually corrupt agent behaviour through multi-turn interactions.

TrustGuard establishes practical foundations for deploying autonomous agents in security-critical environments through three integrated mechanisms addressing core security objectives. Dual-path processing with cryptographic verification enforces instruction integrity by maintaining mathematically provable isolation between system instructions and external inputs. Continuous behavioural attestation protects operational confidentiality through real-time anomaly detection, validating agent operations against established baselines. Dynamic privilege containment achieves privilege minimisation through just-in-time permission allocation with automatic revocation upon task completion.

Production deployments across financial services, healthcare, and cloud infrastructure environments processing over 2.3 million transactions demonstrate TrustGuard's effectiveness under diverse operational constraints including PCI-DSS, HIPAA, and SOC 2 Type II compliance requirements. Empirical evaluation establishes quantifiable security improvements: 4.2% attack success rate against direct injection compared to 26.2% for prompt sanitisation, 94.7% overall detection rate, and 3.8% false positive rate suitable for production deployment. Performance overhead of 11.2% remains within acceptable bounds for enterprise latency SLOs whilst providing comprehensive protection across direct injection, indirect injection, trigger-based, and code-generation attack categories.

The architectural principles validated through extensive empirical evaluation—particularly dual-path processing preventing instruction corruption and dynamic privilege containment limiting attack impact—provide generalisable patterns applicable beyond specific implementations. Integration with existing enterprise security infrastructure including IAM, SIEM, SOAR, and PAM solutions enables deployment within established architectures without requiring wholesale replacement of security controls.

Future developments in hardware acceleration, multi-agent communication protocols, and adaptive defence mechanisms promise to address current computational overhead constraints whilst extending protection to increasingly complex autonomous system deployments. As enterprises continue expanding agent deployments across mission-critical functions, the architectural patterns and empirical validation presented establish essential foundations for maintaining security alongside operational effectiveness—demonstrating that rigorous protection mechanisms and practical performance requirements need not represent irreconcilable objectives when designs explicitly address the unique characteristics of neural language processing systems operating with privileged access.

References

[1] Dezhang Kong et al., "A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures," arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2506.19676

[2] Hyungyo Kim et al., "Exploiting Intel Advanced Matrix Extensions (AMX) for Large Language Model Inference," IEEE COMPUTER ARCHITECTURE LETTERS, 2024. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10538369

[3] LINGZHE ZHANG et al., "A Survey of AIOps in the Era of Large Language Models," arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2507.12472

[4] Butler W. Lampson, "Protection Domains," ACM. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/775265.775268

[5] Ali Al-Kaswan et al., "Code Red! On the Harmfulness of Applying Off-the-Shelf Large Language Models to Programming Tasks," ACM, 2025. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3729380

[6] Nicholas Carlini et al., "Are aligned neural networks adversarially aligned?" 37th Conference on Neural Information Processing Systems, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/c1f0b856a35986348ab3414177266f75-Paper-Conference.pdf

[7] Jingwei Yi et al., "Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models," ACM, 2025. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3690624.3709179

[8] Eric Wallace et al., "Universal Adversarial Triggers for Attacking and Analyzing NLP," arXiv, 2021. [Online]. Available: https://arxiv.org/pdf/1908.07125

[9] Tianshu Zhang et al., "From Data to Deployment: A Comprehensive Analysis of Risks in Large Language Model Research and Development," Wiley, 2024. [Online]. Available: http://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/ise2/7358963

[10] Viswakanth Ankireddi, "AI-Powered Quality Assurance: Revolutionizing Software Development Through Intelligent Anomaly Detection and Automated Monitoring," International Journal on Science and Technology (IJSAT), 2025. [Online]. Available: https://www.ijsat.org/papers/2025/2/3070.pdf

[11] XIAOHAN YUAN et al., "S-Eval: Towards Automated and Comprehensive Safety Evaluation for Large Language Models," ACM, 2025. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3728971

[12] Yi Liu et al., "Prompt Injection attack against LLM-integrated Applications," arXiv, 2024. [Online]. Available: https://arxiv.org/pdf/2306.05499

[13] Dong Shu et al., "AttackEval: How to Evaluate the Effectiveness of Jailbreak Attacking on Large Language Models," arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2401.09002

[14] ASHKAN TOUSI AND MIKEL LUJÁN, "Comparative Analysis of Machine Learning Models for Performance Prediction of the SPEC Benchmarks," IEEE Access, 2021. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9676614

[15] Joao Fonseca et al., "Safeguarding Large Language Models in Real-time with Tunable Safety-Performance Trade-offs," arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2501.02018

[16] KAMEL SMIRI et al., "HARDWARE/SOFTWARE CO-DESIGN IN MULTIPROCESSORS EMBEDDED SYSTEMS AND IOT," Journal of Theoretical and Applied Information Technology, 2024. [Online]. Available: https://jatit.org/volumes/Vol102No2/27Vol102No2.pdf