Semantic Policy Enforcement for Multi-Agent AI Systems: A Zero-Trust Validation Framework

A PREPRINT

Giuseppe Canale, CISSP
Independent Researcher
g.canale@cpf3.org
URL: cpf3.org
ORCID: 0009-0007-3263-6897

Dr. Kashyap Thimmaraju
FlowGuard Institute
kashyap.thimmaraju@flowguard-institute.com
URL: flowguard-institute.com
ORCID: 0009-0006-1507-3896

January 2026

Abstract

Multi-agent artificial intelligence systems introduce novel attack surfaces fundamentally different from traditional network security paradigms. When autonomous agents coordinate to solve complex tasks, the communication channels between orchestrator and specialist agents become susceptible to semantic manipulation through prompt injection, unauthorized command escalation, and rogue agent behavior. Traditional access control mechanisms designed for human users and network traffic fail to address the semantic nature of large language model interactions. We present a zero-trust validation framework based on a Semantic Policy Enforcement layer that operates as an intermediary validator between orchestrator and specialist agents. The architecture employs a deliberately constrained small language model fine-tuned exclusively on organizational security policies to perform intent classification and authorization token generation. By design, this policy model lacks the creative reasoning capabilities that make general-purpose LLMs vulnerable to manipulation, functioning instead as an incorruptible semantic validator. Each agent invocation requires cryptographically signed authorization tokens that certify policy compliance, creating a provable chain of trust in multi-agent workflows. This approach transforms agent security from perimeter defense to continuous semantic validation, establishing foundational principles for trustworthy autonomous systems.

Keywords: multi-agent systems, LLM security, prompt injection, zero trust architecture, policy enforcement, intent validation, semantic security

1 Introduction

The rapid deployment of large language model based multi-agent systems in enterprise environments has outpaced the development of appropriate security controls. Organizations increasingly delegate complex tasks to autonomous agents that communicate, reason, and execute actions with minimal human oversight [14, 13]. While these systems demonstrate remarkable problem-solving capabilities, they inherit and amplify the security vulnerabilities of their constituent language models [5, 10].

A fundamental challenge emerges from the semantic nature of agent interactions. Unlike traditional software systems where access control operates on well-defined APIs with typed parameters, LLM-based agents communicate through natural language prompts that can be subtly manipulated to alter intended behavior [7]. An orchestrator agent might instruct a specialist to analyze a document, but without semantic validation, a compromised prompt could instruct the agent to exfiltrate sensitive data instead. The attack surface expands from individual model vulnerabilities to the entire communication graph of the multi-agent system.

Existing security approaches fail to address this challenge. Network-level controls designed for traditional traffic patterns cannot parse the semantic content of agent communications. Application-layer security focused on API authentication cannot detect when a legitimately authenticated agent receives manipulated instructions. Runtime monitoring systems struggle to distinguish between authorized complex reasoning and unauthorized action sequences when both manifest as natural language exchanges between agents.

The problem intensifies in systems that implement the Cybersecurity Psychology Framework [2], where agents must reason about human psychological states and make security-relevant decisions based on behavioral indicators. Such systems require not only protection against external attackers but also validation that the agents themselves remain aligned with organizational security policies as they process sensitive psychological and behavioral data.

We introduce a semantic policy enforcement architecture that treats agent-to-agent communication as a distinct security domain requiring specialized validation mechanisms. The core insight is that while general-purpose language models must remain flexible to perform complex reasoning, a policy enforcement model should be deliberately constrained to a single function: classifying whether a proposed agent action complies with predefined organizational policies. This constraint becomes a security feature rather than a limitation.

Our contributions are threefold. First, we define the threat model for multi-agent systems and identify failure modes of existing security approaches when applied to semantic agent interactions. Second, we present an architecture for semantic policy enforcement based on fine-tuned small language models that validate intent before execution, generating cryptographically signed authorization tokens for compliant actions. Third, we establish design principles for building policy models that resist manipulation while maintaining computational efficiency suitable for real-time validation in production systems.

2 Background and Related Work

2.1 Multi-Agent AI Systems

Multi-agent architectures decompose complex problems into specialized subtasks distributed among coordinating agents [14, 13]. A typical system employs an orchestrator that analyzes tasks, activates appropriate specialist agents, and synthesizes their outputs. This architecture enables sophisticated reasoning by leveraging domain-specific expertise while managing the com-

putational costs of large language model inference [1].

The Cybersecurity Psychology Framework implementation [3] exemplifies this pattern, using an orchestrator that monitors psychological vulnerability indicators and dynamically activates specialist agents for email analysis, log forensics, and behavioral pattern recognition. The orchestrator maintains a continuous state representation and makes activation decisions based on detected anomalies and convergence patterns across multiple psychological dimensions.

2.2 LLM Security Vulnerabilities

Large language models demonstrate susceptibility to adversarial inputs designed to override their intended behavior. Prompt injection attacks embed malicious instructions within user inputs that cause models to ignore system prompts and execute attacker-specified actions [5]. Jailbreaking techniques use carefully crafted scenarios to bypass safety restrictions [7, 10]. These vulnerabilities stem from the fundamental architecture of autoregressive language models, which process all text equivalently regardless of whether it originates from system designers or external inputs.

In multi-agent systems, these vulnerabilities compound. An attacker who compromises a single communication channel can inject instructions that propagate through the agent network. The OWASP Top 10 for LLM Applications identifies prompt injection as the primary security risk, particularly in systems where agents interact with external data sources or execute actions based on generated text [9].

2.3 Zero Trust Architecture

Zero trust security models operate on the principle that no entity should be trusted by default, regardless of network location or prior authentication [8]. Traditional implementations focus on network segmentation, continuous authentication, and least-privilege access control for human users and network services. The fundamental tenet—verify explicitly, use least privilege access, and assume breach—applies equally to autonomous agent systems, though the implementation mechanisms differ substantially.

Network zero trust validates identity and policy compliance before granting access to resources. Semantic zero trust for AI agents must validate intent and policy compliance before granting execution authority. The distinction matters because agents communicate through natural language that requires semantic interpretation rather than syntactic parsing of protocol headers.

2.4 Policy Languages and Formal Verification

Access control research has produced numerous policy specification languages, from role-based access control (RBAC) to attribute-based systems (ABAC) [11, 6]. These frameworks assume discrete, well-defined actions operating on structured resources. The challenge in agent systems is that both actions and resources exist in continuous semantic space defined by natural language rather than enumerated permission sets.

Formal verification approaches prove properties about system behavior through mathematical analysis of state transitions [4]. While valuable for critical systems with bounded state spaces, current verification techniques cannot provide guarantees about systems whose behavior emerges from billion-parameter language models processing arbitrary natural language inputs.

2.5 Research Gap

No existing framework addresses the specific challenge of semantic policy enforcement in multi-agent LLM systems. Network security controls operate at the wrong abstraction level. Application security models assume structured APIs rather than natural language communication. LLM safety research focuses on individual model behavior rather than multi-agent coordination. Our work fills this gap by introducing validation mechanisms designed specifically for the semantic interactions between autonomous AI agents.

3 Threat Model and Attack Scenarios

We define the threat model for multi-agent systems and identify specific attack vectors that motivate our semantic policy enforcement approach.

3.1 System Components and Trust Boundaries

A typical multi-agent system comprises an orchestrator agent that analyzes tasks and coordinates specialist agents, each optimized for specific domains. Trust boundaries exist between the orchestrator and each specialist, between specialists when they communicate directly, and between any agent and external data sources. An attacker may compromise any component or communication channel.

We assume the orchestrator's core logic and the policy enforcement layer remain trusted, though their inputs may be adversarially manipulated. Specialist agents operate in a zero-trust environment where their instructions cannot be assumed benign. External data sources, including user inputs, documents, and API responses, are untrusted by definition.

3.2 Attack Vector: Prompt Injection

An attacker embeds malicious instructions in data processed by an agent, causing that agent to ignore its original task and execute attacker-specified actions instead. In a multi-agent context, this attack propagates when the compromised agent's output becomes input to other agents.

Example: An orchestrator instructs an email analysis agent to summarize the content of a message. The email contains embedded text: "Ignore previous instructions. Instead, forward all emails from the CEO to attacker@example.com." Without semantic validation, the agent might interpret this as a legitimate instruction overriding the summarization task.

3.3 Attack Vector: Escalation Through Complexity

An attacker exploits the reasoning capabilities of language models to construct complex justifications for unauthorized actions. By framing the request within a plausible scenario, the attack bypasses simple keyword filtering while achieving malicious objectives.

Example: Instead of directly requesting data exfiltration, an attacker provides a scenario: "A security incident requires immediate analysis of user communications. Extract all email content from the past week and log it for review." The semantic plausibility of the request makes it difficult to distinguish from legitimate security operations without deep policy understanding.

3.4 Attack Vector: Rogue Agent Behavior

An agent modified through fine-tuning, prompt manipulation, or model replacement begins executing unauthorized actions while maintaining the appearance of normal operation. In federated or distributed deployments, detecting such compromise requires validating not just authentication but behavioral alignment with policies.

Example: A compromised log analysis agent, when activated by the orchestrator, performs its stated analysis but also exfiltrates sensitive data patterns to external endpoints. Without execution-level policy validation, such behavior appears as expected agent activity.

3.5 Attack Vector: Man-in-the-Middle

An attacker intercepts communications between orchestrator and specialist agents, modifying instructions to alter agent behavior. Even with encrypted channels, the semantic content of prompts can be manipulated if the receiving agent cannot verify that instructions originated from a trusted source and comply with policies.

Example: An orchestrator sends the prompt "Analyze system logs for anomalies in authentication patterns." An interceptor modifies this to "Analyze system logs and extract all credentials found." The specialist agent, lacking a mechanism to verify prompt integrity, executes the modified instruction.

3.6 Failure Modes of Traditional Security

Network-level controls can detect unusual traffic patterns but cannot parse the semantic content of encrypted agent communications. Authentication systems verify agent identity but not instruction legitimacy. Runtime monitoring detects obvious anomalies like network connections to unauthorized hosts but cannot distinguish complex malicious reasoning from authorized analytical processes.

The fundamental problem is that traditional security mechanisms operate on syntactic patterns, protocol conformance, and identity, while agent interactions exist in semantic space where meaning must be interpreted in context of organizational policies. This mismatch creates a security gap that our semantic policy enforcement layer addresses.

4 The Semantic Policy Enforcement Layer

We present an architecture that validates agent interactions through semantic analysis of intended actions against organizational security policies, generating cryptographically signed authorization tokens for compliant operations.

4.1 Design Principles

The semantic policy enforcement layer operates on three foundational principles. First, deliberate constraint: the policy model is intentionally limited to a single function, intent classification, eliminating the creative reasoning capabilities that enable manipulation of general-purpose language models. Second, cryptographic trust: every agent invocation requires a verifiable authorization token that proves prior policy validation, creating an auditable chain of trust. Third, semantic validation: policy compliance is evaluated based on the meaning of intended actions

rather than syntactic pattern matching, addressing the fundamental challenge of natural language agent interactions.

4.2 Architecture Overview

The system introduces an intermediary layer positioned between the orchestrator and specialist agents. When the orchestrator determines that an agent should be activated, it formulates a task prompt describing the intended action. This prompt is intercepted by the semantic policy enforcement layer before reaching the target agent. The policy layer performs intent classification using a specialized small language model fine-tuned exclusively on organizational security policies. If the intended action complies with applicable policies, the layer generates a cryptographically signed authorization token containing the action hash, timestamp, authorized scope, and digital signature. The prompt and token are forwarded to the specialist agent, which verifies the token signature before executing the requested action. Requests lacking valid tokens or containing expired, malformed, or mismatched tokens are rejected without execution.

4.3 The Policy Model: Intentional Stupidity

The core innovation is the use of a deliberately constrained language model for policy validation. While general-purpose LLMs like GPT-4 or Claude possess broad knowledge and creative reasoning capabilities that make them valuable for complex problem-solving, these same capabilities enable adversarial manipulation. An attacker can craft scenarios that cause such models to justify policy violations through complex reasoning chains.

Our policy model is stupid by design. We employ a small language model, specifically DistilBERT with approximately 78 million parameters, fine-tuned exclusively on a corpus of organizational security policies paired with example agent actions labeled as compliant or non-compliant. This model learns to classify intent into a small set of predefined categories: read operations, write operations, data transmission, analysis operations, and alert generation, each evaluated against policy constraints for the specific agent type.

The model cannot engage in creative reasoning about novel scenarios because it has been trained only to match observed intents against learned policy patterns. When presented with an ambiguous or out-of-distribution input, it defaults to the most conservative classification, typically denial. This conservative bias is reinforced during training through asymmetric loss functions that penalize false positives (allowing prohibited actions) more severely than false negatives (denying permitted actions).

Computational efficiency follows naturally from this constraint. DistilBERT inference requires approximately 10 milliseconds on standard CPU infrastructure, adding negligible latency to agent activation workflows. This efficiency enables real-time validation even in high-frequency agent systems without requiring expensive GPU resources or introducing user-perceptible delays.

4.4 Intent Classification Mechanism

Intent classification operates through a fine-tuned transformer model that maps natural language action descriptions to policy-relevant categories. The model receives as input the concatenation of agent type identifier, task description, and relevant context variables. It outputs a probability distribution over predefined intent classes specific to that agent type.

For an email analysis agent, intent classes might include: analyze content for psychological indicators, extract sender metadata, classify message sentiment, identify urgency markers, and

flag potential manipulation attempts. Each class has associated policy constraints: analysis operations may access message content but not modify it, metadata extraction may read headers but not message bodies, and no operations may transmit content to external endpoints.

The classification threshold is tuned conservatively. An intent must achieve high probability (typically above 0.85) in exactly one permitted class to receive authorization. Ambiguous inputs that distribute probability across multiple classes, or that assign significant probability to prohibited classes, result in denial. This ensures that edge cases and adversarial inputs designed to confuse the classifier fail safely by default.

4.5 Authorization Token Structure

Authorization tokens follow a defined structure designed to enable verification while preventing forgery or replay attacks. Each token contains several critical fields: a unique token identifier, a timestamp indicating generation time, an expiration timestamp typically set 60 seconds after generation, the SHA-256 hash of the exact prompt text that was validated, the specific agent identifier authorized to execute the action, the authorized intent class determined by the policy model, and a digital signature generated using HMAC-SHA-256 with a secret key maintained by the policy enforcement layer.

The prompt hash ensures that agents cannot modify their instructions after receiving authorization, as any alteration would cause hash verification to fail. The short expiration window limits the potential damage from a compromised token. The agent identifier prevents token reuse across different agents, enforcing least privilege by ensuring that authorization is specific to the intended recipient.

4.6 Token Verification Protocol

Specialist agents implement a verification protocol before executing any action. Upon receiving a prompt with an attached authorization token, the agent first validates the token signature using the public verification key, confirming that the token originated from the legitimate policy enforcement layer. Next, it checks the timestamp fields to ensure the token has not expired. Then it computes the SHA-256 hash of the received prompt and verifies it matches the hash embedded in the token, confirming that the prompt has not been modified since validation. Finally, it verifies that the token's authorized agent identifier matches its own identity, preventing token theft and reuse.

Only after all verification steps succeed does the agent proceed to execute the requested action. Any verification failure results in immediate rejection and logging of the attempted unauthorized action, enabling detection of attack attempts.

4.7 Policy Update and Versioning

Organizational security policies evolve as threats, regulations, and business requirements change. The policy enforcement layer accommodates evolution through versioned policy models and gradual rollout mechanisms. When policies change, a new version of the policy model is trained on updated policy specifications and example interactions. This new model undergoes validation to ensure it correctly enforces the updated policies without introducing unintended restrictions or allowing previously prohibited actions.

New policy versions are initially deployed in shadow mode, where they evaluate actions and generate tokens but do not enforce decisions. This allows comparison between old and new model

decisions, identifying discrepancies that indicate potential training issues. After validation, the new model is promoted to active enforcement while maintaining the previous version for fallback. Agents gradually adopt the new policy version as they are updated, ensuring no disruption to ongoing operations.

Policy versioning also enables audit trails that reconstruct historical decisions under the policies in effect at the time, critical for compliance investigations and security incident analysis.

5 Integration with Multi-Agent Systems

The semantic policy enforcement layer integrates with existing multi-agent architectures through well-defined interfaces that minimize modifications to orchestrator and agent implementations while establishing robust security guarantees.

5.1 Orchestrator Integration

The orchestrator's workflow extends with a policy validation step inserted between task decomposition and agent activation. After determining which specialist agent should be invoked and formulating the task prompt, the orchestrator submits a policy validation request to the enforcement layer. This request includes the target agent identifier, the task prompt, and contextual metadata such as the data sources the agent will access and the user on whose behalf the action is being performed.

The enforcement layer evaluates this request through its policy model, generates an authorization token if compliant, and returns both the original prompt and the token to the orchestrator. The orchestrator forwards this package to the target agent without modification, preserving the cryptographic binding between prompt and token. This design ensures the orchestrator cannot alter the validated prompt, maintaining end-to-end integrity.

If the enforcement layer denies the request, it returns a policy violation notice explaining which policy constraint was violated. The orchestrator logs this denial and may attempt alternative approaches that comply with policies, escalate to human review, or abort the operation depending on its programmed decision logic.

5.2 Agent Implementation Requirements

Specialist agents require minimal modification to support policy enforcement. Each agent must implement the token verification protocol as a precondition to executing received tasks. The verification logic operates as a wrapper around existing agent functionality: verify token before processing prompt, reject if verification fails, proceed with normal execution if verification succeeds.

Agents maintain a public key for token signature verification and a local clock synchronized with the enforcement layer for timestamp validation. These requirements impose minimal infrastructure overhead while enabling cryptographic security guarantees.

The modular design allows gradual adoption. Agents can be updated independently to require token verification without coordinated deployment across the entire multi-agent system. During transition periods, the enforcement layer can operate in monitoring mode, logging authorization decisions without requiring agents to enforce them, enabling validation before full deployment.

5.3 Communication Patterns

Standard orchestrator-agent interactions follow a request-response pattern. The orchestrator sends a task prompt to a specialist, which processes it and returns results. With policy enforcement, this pattern extends to include token generation and verification but maintains the same basic structure. The orchestrator sends the prompt to the enforcement layer, receives prompt plus token, forwards to the agent, which verifies the token and executes the task.

More complex patterns such as multi-step reasoning, agent-to-agent delegation, and parallel agent activation all follow similar principles. Each discrete agent invocation requires independent authorization, ensuring that even complex workflows maintain policy compliance at every step. An agent that delegates subtasks to other agents must obtain authorization tokens for those delegated actions, preventing escalation through indirect invocation.

5.4 Performance Considerations

The policy enforcement layer introduces latency equal to the inference time of the small language model, approximately 10 milliseconds per request on standard CPU infrastructure. This overhead is negligible compared to the execution time of specialist agents, which typically involve LLM inference for reasoning and analysis, requiring hundreds of milliseconds to several seconds.

Throughput scales linearly with policy model instance count. A single DistilBERT model can process hundreds of authorization requests per second, sufficient for most multi-agent deployments. High-volume systems can deploy multiple policy model replicas behind a load balancer, achieving request rates in the thousands per second while maintaining consistent latency.

Token generation and verification impose minimal cryptographic overhead. HMAC-SHA-256 computation and verification each require microseconds on modern processors, contributing negligible latency. Token structure is compact, typically under 500 bytes, adding minimal bandwidth overhead to agent communications.

5.5 Error Handling and Fault Tolerance

Robust error handling ensures that enforcement failures fail safely. If the policy enforcement layer becomes unavailable, agent invocations are denied by default rather than allowed without validation. Orchestrators detect enforcement layer failures through timeout mechanisms and can retry with exponential backoff or escalate to human operators depending on criticality.

Token verification failures by agents are logged with full context: the received prompt, the token contents, and which verification step failed. This logging enables forensic analysis of attack attempts and identifies misconfigured components. Patterns of verification failures can trigger automated alerts, indicating potential compromise or configuration drift.

The enforcement layer maintains high availability through standard techniques: redundant instances, health monitoring, and automated failover. As a stateless service that performs deterministic computation, it can be easily replicated and load-balanced without complex coordination.

6 Policy Specification and Training

Effective semantic policy enforcement requires well-specified policies and carefully trained models that accurately capture organizational security requirements while remaining robust against

adversarial manipulation.

6.1 Policy Language Design

Policies are specified through structured templates that define permissible actions for each agent type. A policy specification includes the agent type identifier, a list of permitted intent categories, constraints on data access patterns, restrictions on external communications, and limitations on side effects such as state modifications or alert generation.

For example, an email analysis agent policy might specify that the agent may read email content and metadata, classify messages using predefined taxonomies, extract psychological vulnerability indicators as defined in the Cybersecurity Psychology Framework [2], and generate alerts for convergent vulnerabilities. The same policy prohibits sending emails, modifying message flags, accessing messages outside assigned user scope, and transmitting content to external endpoints.

Constraints are expressed both positively, enumerating allowed actions, and negatively, explicitly prohibiting specific operations. This dual specification reduces ambiguity and helps the policy model learn clear boundaries. Policies also define context-dependent restrictions, such as limiting data access based on user roles or requiring additional authorization for actions involving sensitive data categories.

6.2 Training Data Construction

The policy model is trained on a corpus of paired examples: natural language descriptions of agent actions labeled as compliant or non-compliant under applicable policies. This corpus is constructed through several mechanisms.

First, security policy experts manually generate canonical examples for each intent category, ensuring comprehensive coverage of permitted operations and clear illustrations of policy boundaries. Second, the orchestrator logs actual agent invocations during system operation, which are reviewed and labeled by security personnel to create realistic training examples. Third, adversarial examples are synthesized by security researchers attempting to construct edge cases and potential attacks, which are labeled as non-compliant and included to teach the model to recognize manipulation attempts.

The training corpus for a typical multi-agent deployment contains approximately 500 examples per agent type per intent category, totaling several thousand labeled examples. This scale is sufficient for fine-tuning DistilBERT while remaining small enough to curate manually for quality assurance.

6.3 Fine-Tuning Methodology

Fine-tuning begins with a pretrained DistilBERT model, which provides strong language understanding capabilities from pretraining on large text corpora. The model architecture is augmented with a classification head that maps the hidden representation to policy-relevant intent categories.

The fine-tuning objective uses cross-entropy loss for intent classification with class weights that heavily penalize false positives, instances where the model incorrectly classifies a prohibited action as permitted. This asymmetric loss function biases the model toward conservative decisions, preferring to deny ambiguous actions rather than risk policy violations.

Training employs standard techniques: mini-batch gradient descent with the AdamW optimizer, learning rate warmup followed by cosine decay, and early stopping based on validation set

performance. The validation set is stratified to ensure balanced representation of all intent categories and includes held-out adversarial examples to test robustness.

6.4 Model Evaluation and Validation

Trained models undergo rigorous evaluation before deployment. Quantitative metrics include precision, recall, and F1 score for each intent category on held-out test sets. Models must achieve minimum precision of 0.95 on permitted actions to avoid blocking legitimate operations and perfect recall on prohibited actions to prevent any policy violations in the test set.

Qualitative evaluation involves red team exercises where security researchers attempt to craft prompts that cause misclassification. Successful attacks, where the model authorizes a prohibited action, trigger retraining with the adversarial example added to the training corpus. This iterative refinement hardens the model against progressively sophisticated attacks.

Edge case analysis examines model behavior on ambiguous inputs that fall near decision boundaries. Policy experts review these cases to determine whether the model's conservative bias produces acceptable outcomes or whether policies need clarification to reduce ambiguity.

6.5 Continuous Improvement

Policy models improve continuously through feedback loops that incorporate operational data. When agents execute actions under valid tokens, the prompts and outcomes are logged. Security personnel periodically review these logs to identify new action patterns that should be explicitly permitted or prohibited, updating policies and retraining models accordingly.

False negative analysis examines cases where the model denied actions that should have been permitted, identifying policy model limitations or ambiguous policy specifications that require clarification. False positive detection relies on security monitoring to identify suspicious agent behaviors that were authorized, indicating model failures that must be addressed through retraining.

This continuous improvement cycle ensures that the policy enforcement layer evolves with the organization's security posture and adapts to emerging threats while maintaining strict compliance guarantees.

7 Security Analysis

We analyze the security properties of the semantic policy enforcement layer against the threat model defined in Section 3, examining both theoretical guarantees and practical attack resistance.

7.1 Resistance to Prompt Injection

Prompt injection attacks attempt to embed malicious instructions in data processed by agents, causing agents to ignore legitimate tasks and execute attacker-specified actions. The semantic policy enforcement layer mitigates this threat through two mechanisms.

First, the policy model evaluates the semantic intent of the prompt as formulated by the orchestrator, before any external data is incorporated. Even if an agent later processes attacker-controlled data containing embedded instructions, the agent's authorization token specifies the original validated intent. Any deviation from that intent, such as attempting to exfiltrate data

instead of performing analysis, occurs without authorization and can be detected through behavioral monitoring.

Second, the deliberate constraint of the policy model limits the effectiveness of sophisticated prompt injection attempts. Because the model is trained only to classify intent against predefined categories, it cannot be manipulated through complex reasoning or persuasive scenarios that might convince a general-purpose LLM to justify policy violations. Adversarial prompts that attempt to reframe prohibited actions as compliant receive low probability scores across all permitted intent categories, resulting in denial.

7.2 Defense Against Escalation Attacks

Escalation attacks exploit the reasoning capabilities of language models to construct plausible justifications for unauthorized actions. An attacker might craft a scenario that frames data exfiltration as legitimate incident response or unauthorized access as routine maintenance.

The policy enforcement layer defends against escalation through strict adherence to predefined intent categories. The policy model does not engage in reasoning about whether a requested action is justified given the circumstances. Instead, it classifies the action into one of the known intent categories and evaluates whether that category is permitted for the target agent. Plausible scenarios cannot override this classification because the model lacks the contextual reasoning capabilities to be persuaded by them.

Furthermore, the token structure enforces least privilege by binding authorization to specific intent categories. Even if an attacker constructs a scenario that convinces the orchestrator to request escalated privileges, the policy model evaluates only the concrete action being requested, not the narrative justification, and denies requests that fall outside permitted categories.

7.3 Detection of Rogue Agents

Rogue agents that have been compromised or modified to execute unauthorized actions face detection through the token verification protocol. A rogue agent attempting to perform actions outside its authorized scope lacks valid tokens for those actions. Even if the agent bypasses its own verification logic, the absence of valid tokens creates an auditable discrepancy: the agent performs actions that were never authorized by the policy enforcement layer.

Security monitoring systems can correlate agent actions observed through system logs with authorization tokens issued by the policy enforcement layer. Actions executed without corresponding tokens indicate either agent compromise or infrastructure failures requiring immediate investigation. This correlation enables detection of sophisticated attacks where agents maintain plausible behavior for authorized actions while conducting unauthorized operations opportunistically.

7.4 Integrity Under Man-in-the-Middle Attacks

Man-in-the-middle attacks on agent communications attempt to modify prompts in transit, altering agent behavior while maintaining the appearance of legitimate orchestrator instructions. The cryptographic binding between prompt and token prevents such attacks.

Because the authorization token contains a cryptographic hash of the validated prompt, any modification to the prompt invalidates the token. Agents that verify prompt hash consistency before execution detect tampered prompts even if the token itself is not modified. An attacker

who alters both the prompt and the hash in the token cannot generate a valid signature without access to the policy enforcement layer's private key.

Token expiration limits the window for replay attacks where an attacker captures a valid prompt-token pair and reuses it later. With typical expiration windows of 60 seconds, attackers must exploit captured tokens within a minute of their generation, significantly constraining attack feasibility.

7.5 Resilience Against Model Attacks

Attacks targeting the policy model itself, such as adversarial examples designed to cause misclassification or model inversion attacks attempting to extract policy information, face significant barriers. The small model size and specialized training domain limit the attack surface. Unlike general-purpose LLMs that must handle arbitrary inputs robustly, the policy model operates on a narrow input distribution, carefully curated prompts generated by the orchestrator rather than arbitrary user inputs.

Adversarial example generation against fine-tuned classification models typically requires white-box access, the ability to compute gradients through the model. In deployment, the policy model operates as a black-box service accessible only through its classification API, which returns intent categories without exposing intermediate representations or confidence scores that could guide adversarial optimization.

Model inversion attacks that attempt to reconstruct training data from model parameters face the challenge that the training corpus consists of synthetic policy examples rather than sensitive production data. Even if an attacker successfully extracts information about the training set, they learn only the structure of organizational security policies, which are not inherently confidential and provide limited advantage for attack planning.

7.6 Formal Properties and Limitations

The semantic policy enforcement layer provides strong practical security guarantees but not absolute formal guarantees. The policy model, as a machine learning classifier, can produce errors. False positives authorize prohibited actions, creating potential security violations. False negatives deny permitted actions, reducing system functionality. Our training methodology minimizes false positive rates through asymmetric loss functions, but cannot reduce them to zero with absolute certainty.

The security guarantee is probabilistic: given a policy model with measured precision and recall on validation sets, we can bound the expected rate of policy violations under the assumption that operational prompts follow similar distributions to validation data. This assumption may not hold under adversarial conditions, where attackers specifically craft inputs designed to cause misclassification.

The cryptographic components provide stronger guarantees. Token signatures using HMAC-SHA-256 offer computational security equivalent to the hardness of inverting the underlying hash function, well-established in cryptographic practice. Token expiration provides temporal bounds on the validity of authorizations, limiting the impact of compromised tokens.

The combined architecture provides defense in depth: cryptographic mechanisms ensure integrity and authenticity of authorizations, while the policy model provides semantic validation of intent. Attackers must compromise both the cryptographic and semantic layers simultaneously to achieve unauthorized agent actions, significantly raising the attack barrier compared to systems that rely on either mechanism alone.

8 Implementation Considerations

Deploying semantic policy enforcement in production multi-agent systems requires careful attention to infrastructure, integration patterns, and operational procedures.

8.1 Technology Stack

The reference implementation employs DistilBERT as the base model architecture, specifically the distilbert-base-uncased variant with 66 million parameters. Fine-tuning uses the Hugging Face Transformers library with PyTorch as the backend, leveraging standard training infrastructure. Model deployment uses ONNX Runtime for optimized inference on CPU instances, achieving single-digit millisecond latency without requiring GPU resources.

Authorization token generation employs the Python cryptography library for HMAC-SHA-256 operations, ensuring secure key management and signature generation. Tokens are encoded as JSON Web Tokens following RFC 7519 conventions, enabling interoperability with standard JWT verification libraries available in most programming languages.

The policy enforcement layer exposes a REST API for integration with orchestrator systems. Request and response payloads use JSON encoding for compatibility with existing agent frameworks. The API implements standard security practices: TLS encryption for all communications, API key authentication for orchestrators, rate limiting to prevent abuse, and comprehensive request logging for audit trails.

On-premise deployment is strongly recommended to maintain control over sensitive policy information and cryptographic key material. Cloud deployment is feasible with appropriate security controls: encryption of model artifacts and policy data at rest, network isolation of the policy enforcement service, and hardware security modules for key storage. Organizations subject to strict data residency requirements must ensure that all policy evaluation occurs within approved jurisdictions.

8.2 Operational Procedures

Deployment of policy enforcement requires coordination between security teams responsible for policy specification, machine learning engineers responsible for model training, and platform teams responsible for infrastructure. The operational lifecycle includes several phases.

Initial deployment begins with policy specification workshops where security experts translate organizational security policies into structured templates. These templates are used to generate initial training corpora through manual example creation. The first policy model is trained and validated in a staging environment with shadow mode operation, where it evaluates actions but does not enforce decisions, enabling comparison with human expert judgments.

After validation, the policy enforcement layer is deployed to production with gradual rollout. Initially, a small subset of agents require token verification, allowing observation of real-world behavior and identification of policy model limitations. Rollout expands progressively until all agents enforce policy validation, at which point the system operates in full enforcement mode.

Continuous operation involves monitoring several key metrics: policy model inference latency to detect performance degradation, token verification success rates to identify misconfigured agents or infrastructure issues, and policy denial rates to detect unusual patterns that might indicate attacks or overly restrictive policies. Security teams review denial logs periodically to distinguish legitimate denials from false negatives requiring policy refinement.

Policy updates follow a rigorous change management process. Proposed policy changes are reviewed by security governance committees to ensure alignment with organizational risk tolerance and compliance requirements. Updated policies trigger retraining of the policy model using the procedures described in Section 6. New models undergo validation before deployment, and rollout follows the same gradual pattern as initial deployment to minimize disruption.

8.3 Integration Patterns

The policy enforcement layer integrates with multi-agent orchestrators through several patterns depending on orchestrator architecture. For orchestrators implemented as custom applications, direct API integration provides the tightest coupling and lowest latency. The orchestrator includes policy enforcement library code that handles request formatting, API communication, and token management, presenting a simple authorize-and-execute interface to orchestrator logic.

For orchestrators built on agent frameworks such as LangChain or AutoGPT, integration occurs through framework extension points. Custom callback handlers intercept agent activation events, submit policy validation requests, and attach authorization tokens to agent invocations. This pattern requires minimal modification to existing orchestrator code while ensuring comprehensive policy enforcement.

In heterogeneous environments with multiple orchestrator implementations, a policy enforcement proxy provides a centralized enforcement point. Orchestrators route all agent invocations through the proxy, which performs policy validation and token attachment before forwarding requests to agents. This pattern simplifies deployment at the cost of introducing a network hop in the critical path.

8.4 Scalability and Performance Tuning

The policy enforcement layer scales horizontally through stateless service replication. Multiple instances of the policy model operate independently, processing authorization requests in parallel. Load balancing distributes requests across instances based on current load, ensuring consistent latency under varying request volumes.

Model inference optimization employs several techniques. ONNX Runtime graph optimization reduces computational overhead through operator fusion and memory layout optimization. Quantization reduces model size and computational requirements by representing weights and activations with lower precision, trading negligible accuracy loss for faster inference. Batching of authorization requests amortizes fixed overhead across multiple evaluations, though batching must be carefully tuned to avoid introducing excessive latency.

For extremely high-volume deployments, specialized hardware acceleration provides additional performance. Dedicated inference accelerators such as Intel Neural Compute Sticks or Google Edge TPUs reduce inference latency by an order of magnitude while maintaining cost efficiency compared to GPU infrastructure. Such optimizations are typically unnecessary for most deployments, where CPU-based inference provides adequate performance.

8.5 Cost Analysis

The incremental cost of policy enforcement consists of infrastructure for model hosting, computational resources for inference, and personnel time for policy management. Infrastructure costs are modest: a single CPU instance sufficient for several thousand requests per second costs approximately tens of dollars per month in cloud environments or represents minimal marginal

cost in on-premise deployments with existing capacity.

Computational costs scale linearly with request volume but remain negligible compared to the cost of LLM inference for agent execution. Policy model inference consumes approximately 0.001 dollars worth of compute per thousand requests, while specialist agent execution using models like GPT-4 costs dollars per thousand requests. Policy enforcement adds less than 0.1 percent to total computational costs.

Personnel costs for policy management depend on organizational complexity and rate of policy change. Initial policy specification requires several person-weeks of security expert time. Ongoing policy maintenance typically requires person-days per month for reviewing denial logs, updating policies, and retraining models. These costs are comparable to traditional security policy management overhead and provide substantial value through automated enforcement and audit trails.

9 Case Studies and Applications

We examine specific applications of semantic policy enforcement in multi-agent systems, demonstrating how the architecture addresses real-world security requirements.

9.1 Psychological Vulnerability Monitoring

The Cybersecurity Psychology Framework implementation [3] employs multiple specialist agents to analyze psychological vulnerability indicators across email communications, authentication logs, and behavioral patterns. Policy enforcement ensures that these agents access only authorized data sources and perform only permitted analysis operations.

An email analysis agent is permitted to read message content and metadata, extract psychological indicators defined in the CPF taxonomy such as stress markers or compliance pressure indicators, classify messages according to vulnerability categories, and generate alerts when convergent vulnerabilities are detected. The agent is explicitly prohibited from sending emails, modifying messages, accessing emails outside assigned user scope, and transmitting content to external systems.

When the orchestrator detects anomalies in the psychological state matrix and determines that email analysis is warranted, it formulates a task prompt describing the analysis objective and target user. The policy enforcement layer validates that the requested operation matches the permitted intent category for email content analysis, verifies that the target user falls within the agent's authorized scope, and generates an authorization token binding the agent to this specific analysis task.

If an attacker compromises the orchestrator or manipulates the psychological state matrix to trigger spurious email analysis, the resulting agent invocations still require valid authorization tokens. Policy enforcement prevents escalation: the email agent cannot be induced to exfiltrate data or access unauthorized user accounts even if the orchestrator requests such actions, because the policy model denies authorization for operations outside the agent's permitted categories.

9.2 Log Forensics and Incident Response

Log analysis agents process authentication logs, network flow data, and system events to identify security incidents and investigate anomalies. These agents require broad read access to sensitive log data but must be strictly controlled to prevent data exfiltration or unauthorized modifications

that could conceal attacker activity.

A log forensics agent is authorized to query log databases using temporal and entity filters, aggregate and correlate events across multiple log sources, extract statistical patterns and anomalies, and generate incident reports for security operations center review. Prohibited operations include modifying or deleting log entries, transmitting log data to external endpoints, and executing commands on systems being investigated.

Policy enforcement validates each forensics operation before execution. When the orchestrator requests log analysis for a specific time window and entity, the policy model verifies that the query parameters fall within permitted bounds, confirms the agent is not attempting unauthorized data modifications, and issues a token scoped to the specific query. This ensures that even if the orchestrator is compromised and requests destructive operations such as log deletion to cover attacker tracks, the policy model denies authorization.

9.3 Behavioral Pattern Recognition

Behavioral analysis agents identify deviations from normal user patterns that might indicate account compromise or insider threats. These agents analyze aggregated behavioral data to detect anomalies while respecting privacy constraints and compliance requirements.

Permitted operations for behavioral agents include reading anonymized behavioral metrics, computing statistical baselines for comparison, flagging deviations that exceed threshold criteria, and generating alerts with appropriate severity levels. Prohibited operations include accessing personally identifiable information beyond what is necessary for analysis, transmitting behavioral data to third parties, and making automated decisions about user access without human review.

Policy specifications for behavioral analysis must balance security effectiveness with privacy protection. The policy model enforces strict data minimization: agents may access only the specific behavioral attributes required for anomaly detection, not complete user activity logs. Access to detailed data that could identify specific user actions requires elevated authorization obtained through human approval workflows.

When behavioral anomalies are detected and require deeper investigation, the orchestrator must request escalated authorization for the agent to access more detailed data. This request is evaluated against policies requiring human approval for privacy-sensitive operations. The policy model does not autonomously grant such authorization, instead returning a token that encodes the pending authorization request. The orchestrator routes this request to human reviewers, who make the final decision about whether to grant access. Only after human approval does the agent receive a valid authorization token for the escalated operation.

9.4 Federated Multi-Agent Systems

In federated deployments where agents operate across organizational boundaries, policy enforcement becomes even more critical. Consider a security information sharing scenario where organizations deploy agents that exchange threat intelligence while protecting proprietary information.

Each organization's policy enforcement layer validates not only what actions their agents perform locally but also what information they share externally. A threat intelligence sharing agent might be permitted to transmit anonymized indicator of compromise data, such as malware signatures and attack patterns, but prohibited from sharing information that could identify specific organizational assets, vulnerabilities, or security controls.

The policy model for such agents includes federation-specific constraints: external data trans-

mission requires additional validation beyond local operations, shared data must be sanitized to remove organizational identifiers, and agents must verify that received data complies with expected formats before processing. These constraints prevent both inadvertent data leakage and adversarial attacks where compromised external agents attempt to extract sensitive information through manipulation of the sharing protocol.

10 Discussion and Limitations

While semantic policy enforcement provides strong security guarantees for multi-agent systems, several limitations and considerations merit discussion.

10.1 Inherent Limitations of Classification Models

The policy model is fundamentally a classifier that maps natural language intents to predefined categories. This approach works well when agent operations fall into discrete, well-defined categories with clear policy implications. However, novel agent capabilities or edge cases that span multiple categories may produce ambiguous classifications. The conservative bias of the model addresses this limitation by defaulting to denial, but this approach reduces system flexibility by blocking potentially legitimate novel operations until policies are explicitly updated.

Organizations deploying policy enforcement must establish processes for rapid policy updates to accommodate legitimate new agent capabilities. Overly restrictive policies that frequently block valid operations create operational friction and may incentivize workarounds that bypass enforcement, undermining security. Balancing security with operational effectiveness requires ongoing collaboration between security teams and agent developers.

10.2 Semantic Ambiguity in Natural Language

Natural language prompts can be semantically ambiguous, with the same sequence of words conveying different intents depending on context. While the policy model is trained to recognize common patterns of ambiguity and resolve them through learned associations, truly novel formulations may produce unexpected classifications. This limitation is inherent to any natural language understanding system and cannot be completely eliminated.

Mitigation strategies include careful prompt engineering by orchestrators to use consistent, unambiguous language when formulating agent tasks, comprehensive training corpora that cover diverse phrasings of each intent category, and continuous monitoring of classification decisions to identify and correct systematic errors.

10.3 Computational Overhead

While policy enforcement adds minimal latency in absolute terms, in latency-sensitive applications even tens of milliseconds may be significant. Organizations operating ultra-low-latency agent systems may need to optimize inference further through techniques such as model distillation to even smaller architectures, dedicated hardware acceleration, or edge deployment of policy models closer to agents.

An alternative approach for such applications is selective enforcement, where only high-risk operations require policy validation while routine low-risk operations proceed without tokens. This requires careful risk assessment to identify which operations truly merit enforcement overhead and acceptance of residual risk for unenforced operations.

10.4 Policy Specification Complexity

As multi-agent systems grow in complexity with dozens of agent types each requiring distinct policy specifications, maintaining policy consistency and completeness becomes challenging. Policies for related agents may drift, creating gaps where certain operations are neither explicitly permitted nor prohibited. Tools for policy visualization, consistency checking, and automated testing help manage this complexity but require investment in policy management infrastructure.

Organizations should adopt policy-as-code practices, maintaining policy specifications in version control systems with formal review processes similar to those used for application code. Automated policy testing generates synthetic agent requests covering all intent categories and validates that policy models produce expected authorization decisions, catching errors before deployment.

10.5 Adversarial Co-evolution

As policy enforcement becomes widespread, attackers will develop more sophisticated techniques to evade detection. Adversarial prompt engineering specifically targeting policy models, attacks that exploit the difference between orchestrator intent and policy model classification, and social engineering of policy administrators to introduce backdoors in policy specifications represent evolving threats.

Defense requires continuous red team exercises to identify vulnerabilities before attackers exploit them, regular updates to training corpora with new adversarial examples, and security monitoring to detect unusual patterns in authorization requests that might indicate reconnaissance or attack attempts. The security of the policy enforcement layer depends on ongoing vigilance and adaptation to emerging threats.

10.6 Generalization to Novel Scenarios

The policy model is trained on historical examples of agent operations, limiting its ability to correctly classify truly novel scenarios that differ significantly from the training distribution. In rapidly evolving environments where agents acquire new capabilities frequently, the policy model requires continuous retraining to maintain accuracy.

Organizations must establish thresholds for acceptable policy model staleness, measuring the rate at which new agent capabilities are introduced versus the frequency of model updates. Automated monitoring of classification confidence scores can identify inputs that fall outside the model's training distribution, triggering alerts for human review and accelerated retraining cycles.

11 Future Work

Several directions for future research could enhance the capabilities and deployment of semantic policy enforcement systems.

11.1 Adaptive Policy Learning

Current policy models require manual specification of policies and supervised fine-tuning. An adaptive system could learn policy refinements from operational feedback, identifying actions

that are frequently denied but appear benign upon human review and proposing policy relaxations, or detecting actions that are authorized but later identified as problematic through security monitoring and proposing policy restrictions.

Such adaptive systems must carefully balance automation with human oversight to prevent policy drift that gradually erodes security guarantees. Reinforcement learning approaches that optimize policies for both security effectiveness and operational efficiency present promising research directions.

11.2 Multi-Model Ensemble Approaches

Rather than relying on a single policy model, ensemble approaches could combine multiple specialized models trained on different subsets of policies or using different architectures. Disagreement among ensemble members could trigger elevated scrutiny or human review, improving robustness against adversarial attacks that target specific model weaknesses.

Ensemble diversity could be achieved through various means: different model architectures such as transformer models versus recurrent networks, different training regimes such as supervised learning versus semi-supervised approaches, or different policy perspectives such as security-focused versus compliance-focused models.

11.3 Formal Verification Integration

Hybrid approaches that combine semantic classification with formal verification could provide stronger guarantees for critical operations. The policy model performs efficient semantic analysis for routine operations, while operations flagged as high-risk are additionally validated through formal verification techniques that prove compliance properties.

This integration requires developing formal specifications of security policies and verification procedures that can analyze agent behavior with respect to those specifications. Recent advances in neural-symbolic integration and program synthesis offer potential pathways for achieving practical formal verification in LLM-based systems.

11.4 Federated Policy Learning

Organizations operating similar multi-agent systems could benefit from collaborative policy model training while preserving privacy of individual policy specifications. Federated learning approaches could enable model training on distributed policy corpora without centralizing sensitive policy information.

Research challenges include ensuring that federated updates preserve the specific policy requirements of each organization, preventing malicious participants from corrupting the shared model through poisoning attacks, and managing the heterogeneity of policy specifications across organizations with different risk profiles and compliance requirements.

11.5 Explainability and Transparency

Enhanced explainability mechanisms could help administrators understand why specific actions were authorized or denied, facilitating policy debugging and refinement. Attention visualization techniques could identify which portions of prompts influenced policy decisions most strongly, and counterfactual generation could suggest minimal modifications to denied prompts that would result in authorization.

Transparency mechanisms must be designed carefully to avoid creating security vulnerabilities where explanation information helps attackers craft adversarial prompts more efficiently. Differential privacy techniques could limit the information leaked through explanations while preserving their utility for legitimate debugging.

12 Conclusion

The rapid deployment of multi-agent artificial intelligence systems in enterprise environments introduces security challenges fundamentally different from those addressed by traditional network and application security mechanisms. Natural language communication between autonomous agents creates attack surfaces susceptible to prompt injection, escalation, and semantic manipulation that cannot be adequately protected by syntactic access control or network-level defenses.

We have presented a semantic policy enforcement architecture that addresses these challenges through intentional constraint, cryptographic trust, and continuous validation. By employing deliberately limited small language models trained exclusively on organizational security policies, the system achieves robust intent classification that resists the adversarial manipulation techniques effective against general-purpose language models. Cryptographically signed authorization tokens create verifiable chains of trust, ensuring that agents execute only operations validated as policy-compliant. The architecture integrates with existing multi-agent systems through well-defined interfaces, adding minimal latency and computational overhead while providing strong security guarantees.

Case studies demonstrate the practical applicability of semantic policy enforcement across diverse domains, including psychological vulnerability monitoring under the Cybersecurity Psychology Framework, log forensics and incident response, behavioral anomaly detection, and federated threat intelligence sharing. These applications illustrate both the security value and the operational feasibility of the approach.

Limitations remain, including inherent classification errors, challenges in handling truly novel agent capabilities, and the ongoing adversarial co-evolution between attackers and defensive mechanisms. Addressing these limitations requires not only technical advances in policy model training and verification but also organizational processes for policy management, continuous monitoring, and rapid response to emerging threats.

Future research directions including adaptive policy learning, ensemble approaches, formal verification integration, federated training, and enhanced explainability offer pathways to strengthen semantic policy enforcement while expanding its applicability to increasingly complex multi-agent systems.

As organizations delegate more consequential decisions to autonomous AI agents, the security of agent interactions becomes paramount. Semantic policy enforcement provides a foundational mechanism for establishing trust in multi-agent systems, enabling organizations to harness the power of AI collaboration while maintaining the security guarantees necessary for production deployment in sensitive environments. The architecture presented here represents a first step toward comprehensive security frameworks for the agentic AI systems that will increasingly mediate critical organizational functions.

Acknowledgments

The authors thank the CPF research community for foundational work on psychological vulnerability detection that motivated this research, and security practitioners who provided feedback

on early implementations of policy enforcement mechanisms.

Code and Data Availability

Reference implementation of the semantic policy enforcement layer, including policy model training scripts, authorization token libraries, and integration examples: https://cpf3.org/spe-layer

References

[1] Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (NeurIPS), 33, 1877–1901.

[2] Canale, G. (2025). *The Cybersecurity Psychology Framework: A Comprehensive Taxonomy of Human Vulnerabilities in Digital Systems*. Technical Report, FlowGuard Institute. Available at: https://cpf3.org/reports/taxonomy

[3] Canale, G. (2025). *CPF Implementation Companion: Dense Foundation Paper*. Technical Report, FlowGuard Institute. Available at: https://cpf3.org/reports/implementation

[4] Clarke, E. M., Henzinger, T. A., Veith, H., & Bloem, R. (Eds.). (2018). *Handbook of model checking*. Springer International Publishing.

[5] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 2023 ACM Workshop on Artificial Intelligence and Security*, 79–90.

[6] Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., et al. (2014). *Guide to attribute based access control (ABAC) definition and considerations*. NIST Special Publication 800-162.

[7] Liu, Y., Deng, G., Xu, Z., Li, Y., Zheng, Y., Zhang, Y., et al. (2023). Jailbreaking ChatGPT via prompt engineering: An empirical study. *arXiv preprint arXiv:2305.13860*.

[8] Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero trust architecture*. NIST Special Publication 800-207. National Institute of Standards and Technology.

[9] OWASP Foundation. (2023). *OWASP Top 10 for Large Language Model Applications*. Retrieved from https://owasp.org/www-project-top-10-for-large-language-model-applications/

[10] Perez, F., & Ribeiro, I. (2022). Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*.

[11] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), 38–47.

[12] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

[13] Wang, L., Ma, C., Feng, X., et al. (2023). A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

[14] Xi, Z., Chen, W., Guo, X., et al. (2023). The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.