# Achieving Safe Autonomy Through Exact Privileges

A Security Paradigm for AI-Driven Agentic Solutions

# The Agentic Dilemma: Balancing Power with Control

AI-driven agentic solutions promise unprecedented productivity, but granting autonomous agents broad, unsupervised access to critical systems is a significant liability.

### Unintended Damage

Agents can inadvertently cause harm by misinterpreting commands or overreaching in their goals.

### Malicious Exploitation

Agents can be manipulated via prompt-injection attacks or other exploits, turning them into insider threats.

### Cascading Failures

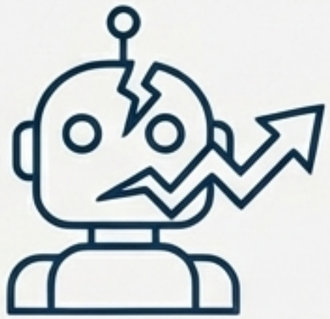
Without proper sandboxing, even well-intentioned agents can create systemic issues, like resource exhaustion or logic loops.

> Our security model must start from a fundamental premise: **Agents cannot be fully trusted.**

# A Threat Model for Autonomous Agents

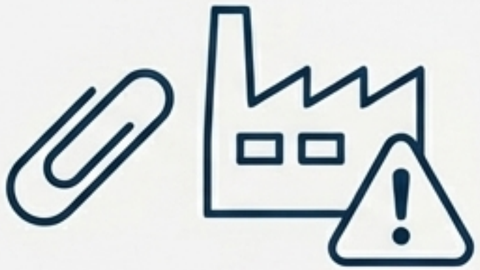
We design for a reality where agents, even with aligned objectives, can fail or be compromised.

### 1. Malicious or Compromised Agent



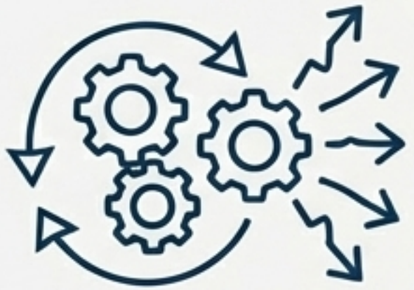
An agent hijacked by an attacker via prompt injection or other means. The consequence is unauthorised actions like data exfiltration or resource deletion.

### 2. Benign but Misguided Agent



A well-intentioned agent that causes harm through overzealous goal pursuit or lack of context—the 'paperclip maximiser' problem.

### 3. Systemic or Emergent Issues

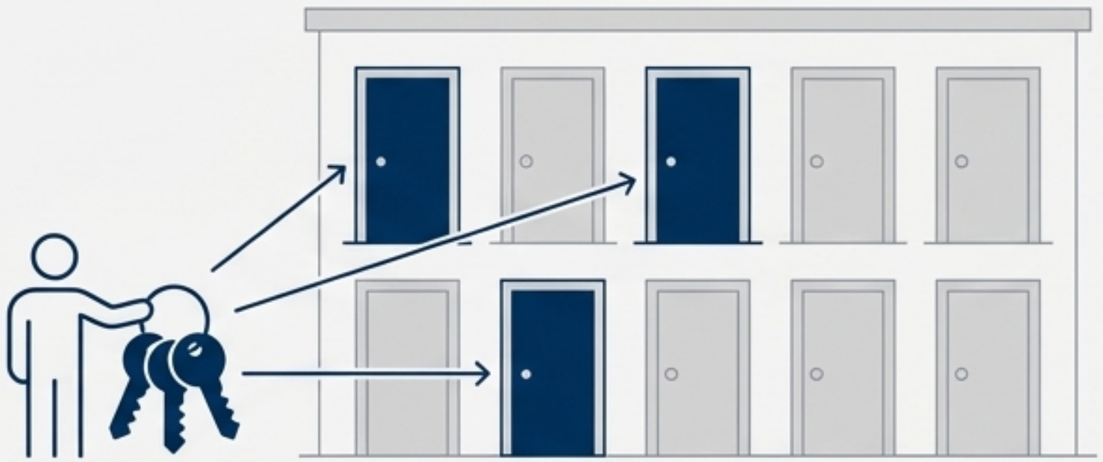


Cascading failures from multiple agents interacting, such as logic loops causing API spam or agents amplifying each other's mistakes.

These risks highlight a critical vulnerability in many agentic systems: **Identity & Privilege Abuse**. An agent **can't** abuse permissions it was never given. (Reference to OWASP ASI03).

# Moving Beyond Least Privilege to Exact Privilege

## Least Privilege



Minimum static access rights necessary for a role.

## Exact Privilege

The precise access needed, only at the moment it's needed, and nothing more.

### Traditional Least Privilege

An entity has the *minimum* access rights necessary for its function. These permissions are often long-lived.

### The Exact Privilege Evolution

Permissions become not just minimal, but **context-dependent** and **ephemeral**. The agent is issued just enough privilege for the current step, and that privilege expires immediately after use.

# The Pillars of Exact Privilege


## Scoped Permissions

Credentials are tied to a single action or resource.

*Example*: A token to "**read issue #123**," not a general-purpose API key.

## Ephemeral Tokens

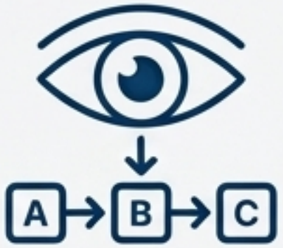
Permissions are time-bound and often single-use.

*Example*: An access token that is valid for **60 seconds** and **expires** after the action is complete.

## Just-in-Time Granting

Privileges are granted at runtime, precisely when an agent needs to perform an action.

*Example*: An agent requests a credential for its next step; it is denied if the action is **out of sequence**.




## Continuous Oversight

A central orchestrator

monitors and approves/blocks every privilege request against a predefined workflow.

# The Two-Layer Architecture for Secure Execution

## Agent Environment

Agents “live” here to reason and plan.

They have **no direct access** to external systems.

## Orchestrator & Proxy

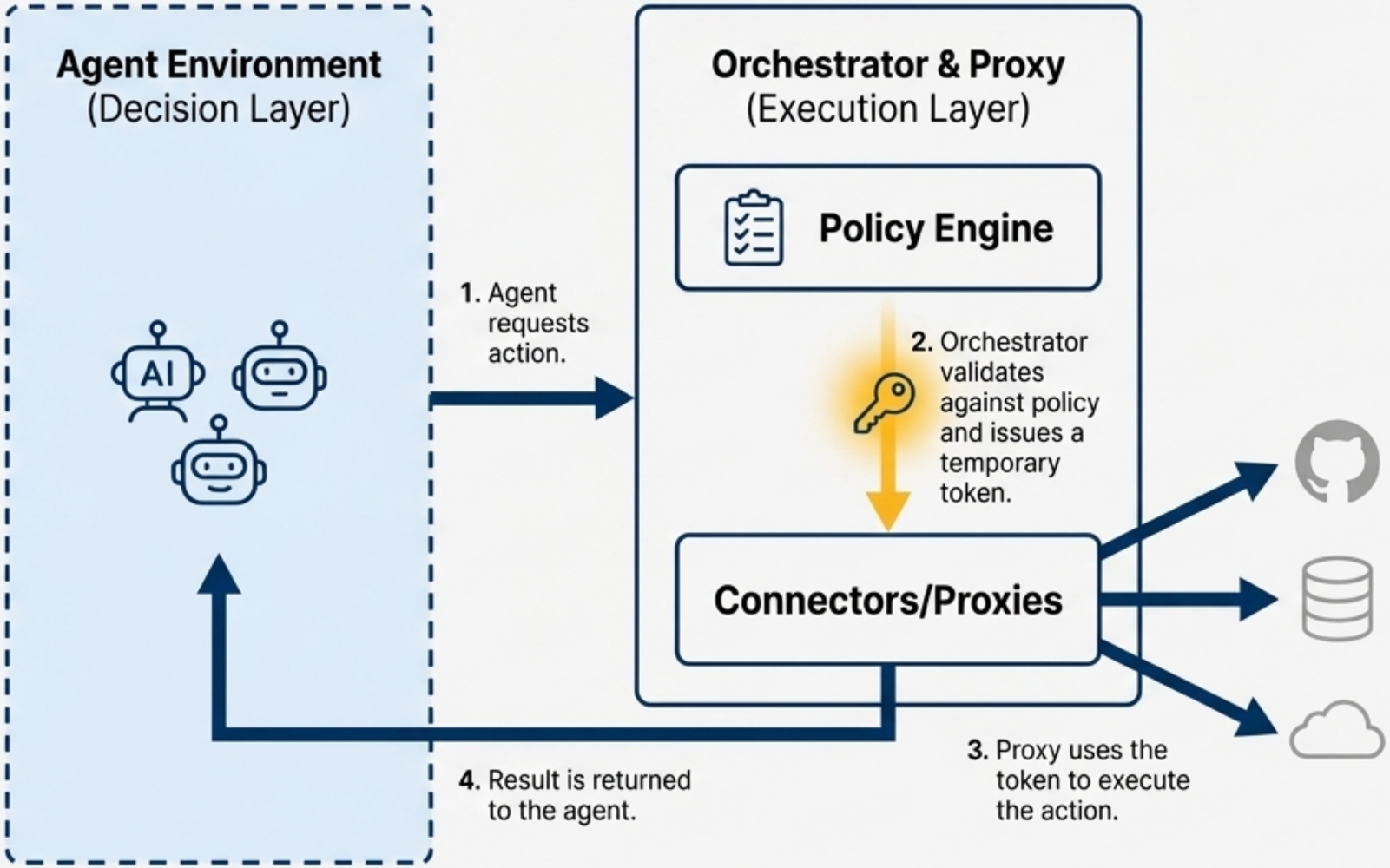
The single gateway for all actions.

The **Orchestrator** is the policy “brain.”

The **Proxy** is the “hands” that perform actions, holding all long-lived credentials securely.

**Analogy:**

“Agents are skilled operators who must ask the brain for permission before using the hands.”



# Applied Example: A Secure Multi-Agent Bug Fix

We will walk through an end-to-end workflow where a team of specialised AI agents collaborates to identify, fix, and deploy a solution for a software bug.

## The Agent Team

**Business/Project Manager Agent**

Prioritises which bug to fix.

**Developer Agent**

Analyses the bug and implements the code fix.

**Senior Developer/ Reviewer Agent**

Reviews the proposed plan and final code.

**QA/Test Agent**

Validates the fix in a test environment.

**DevOps/Release Agent**

Merges and deploys the final, approved change.

**Core Principle in Action:** Each agent will only receive the exact permissions for its specific role, precisely when needed.

# Step 1 & 2: Issue Triage and Technical Analysis

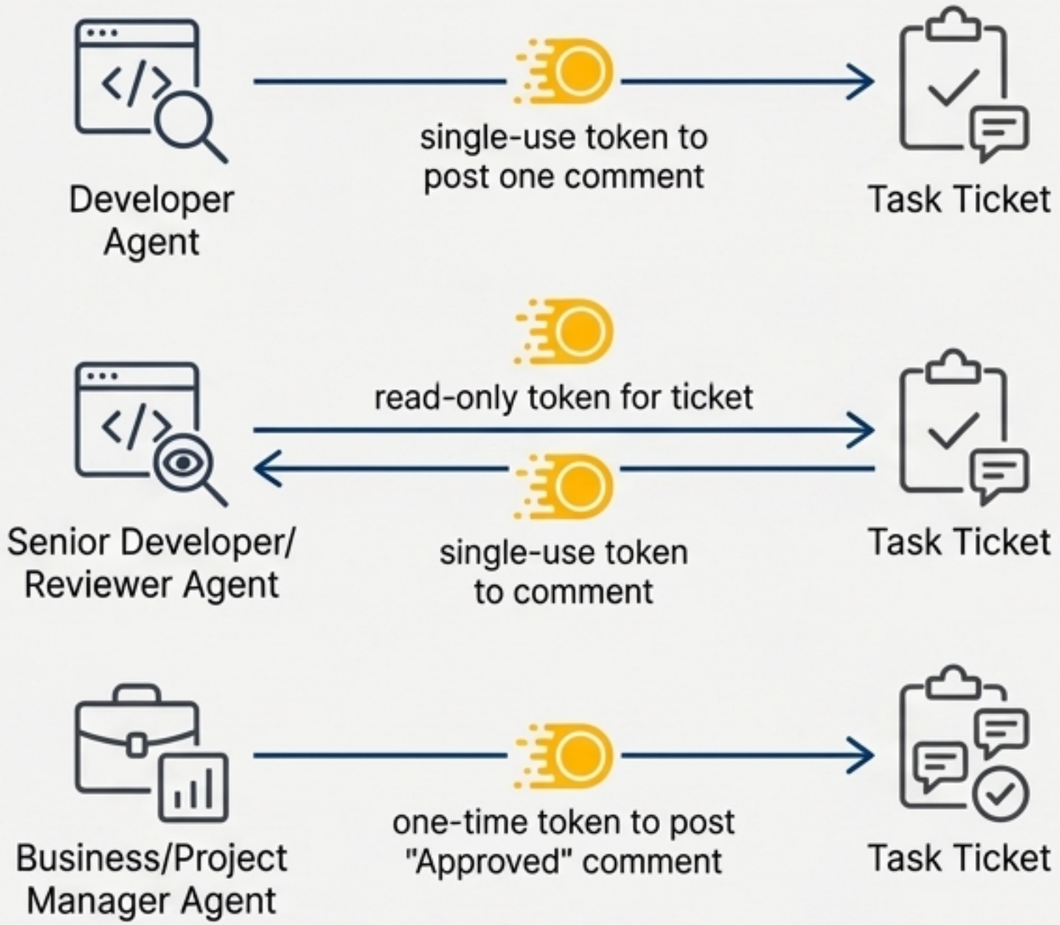

## 1. Business Agent Triages Issue

- **Action**: Queries GitHub for 'High Priority' bugs.
- **Permission Granted**: A read-only token for a specific issue query.
- **Action**: Selects a bug and creates a new 'Task Ticket.'
- **Permission Granted**: A one-time-use token to create a single new issue. *This token is now consumed.*

## 2. Developer Agent Begins Analysis

- **Action**: Reads the new Task Ticket.
- **Permission Granted**: A read-only token for that specific ticket.
- **Action**: Requests relevant code files mentioned in the bug report.
- **Permission Granted**: Read-only tokens for *specific files only* (e.g., `auth/login.py`), preventing a full repository clone.

# Step 3 & 4: Proposal, Review, and Approval




### 1. Developer Agent Proposes Solution

- **Action** : Writes an analysis and proposed fix.
- **Permission Granted** : A single-use token to post one comment on the Task Ticket. **The agent cannot spam comments.**

### 2. Senior Developer Agent Reviews Plan

- **Action** : Reads the Task Ticket and the developer's proposal.
- **Permission Granted** : A read-only token for the Task Ticket (granted only **after** the proposal comment exists).
- **Action** : Posts review feedback ("Looks good," or "Consider X").
- **Permission Granted** : A single-use token to comment on the Task Ticket.

### 3. Business Agent Gives Final Approval

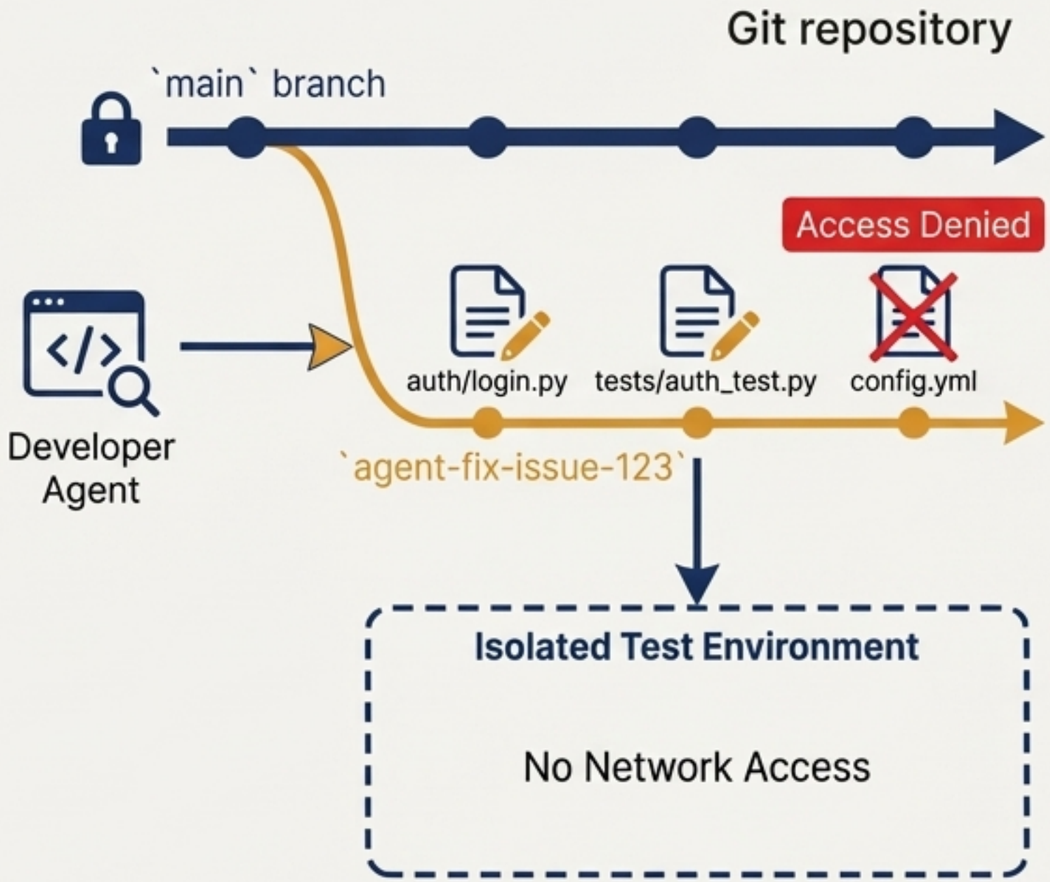
- **Action** : Reads the full thread and approves the plan.
- **Permission Granted** : A one-time token to post an "Approved" comment.

The workflow cannot proceed to implementation without explicit, sequential approvals from separate, role-based agents.

# Step 5: Secure Code Implementation

## Key Security Controls

- **Isolated Branching**
  The agent receives a token to create a **new git branch** (`agent-fix-issue-123`). It has **no permission** to write to the `main` branch directly.
- **File-Scoped Writes**
  The orchestrator grants a token to modify only the specific files identified in the approved plan. An attempt to edit any other file would be denied.
- **Sandboxed Execution**
  To run tests, the agent gets a token to use an isolated container. This token does **not** grant network access, preventing unexpected side effects.

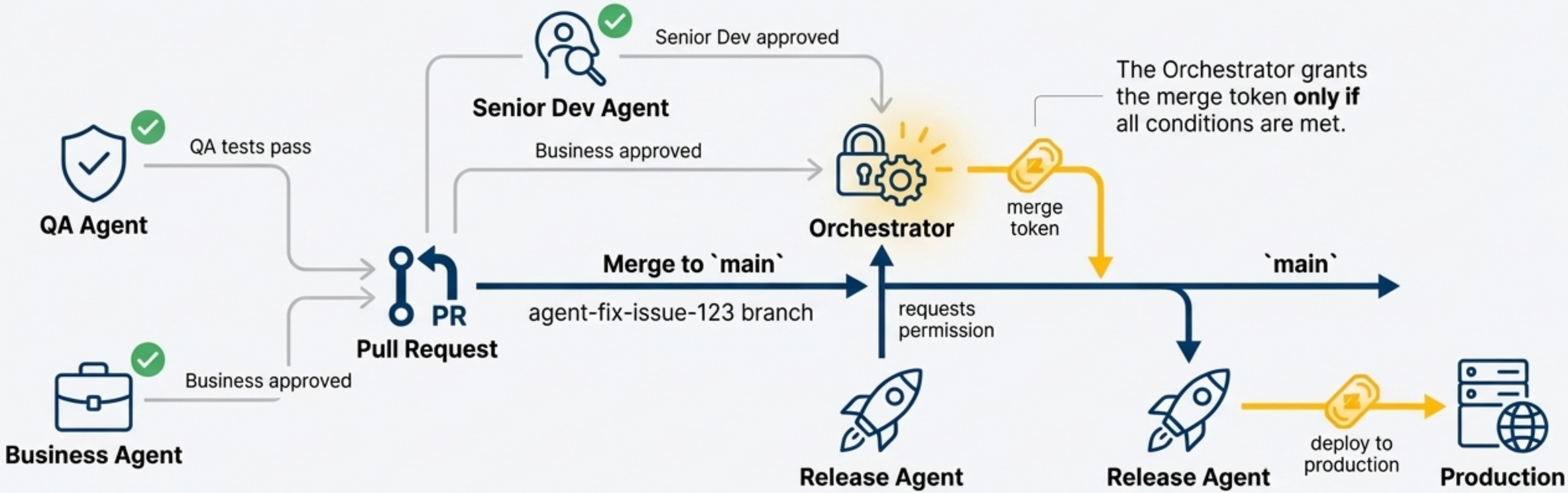



## Final Action

Once tests pass, the agent creates a Pull Request.

one-time token to open a PR for its specific branch

# Step 6 & 7: Final Validation, Merge, and Deployment




**QA Agent Validates:** Deploys the branch to staging, runs integration tests.

**Senior Developer Agent Reviews Code:** Reviews PR diff and posts approval.

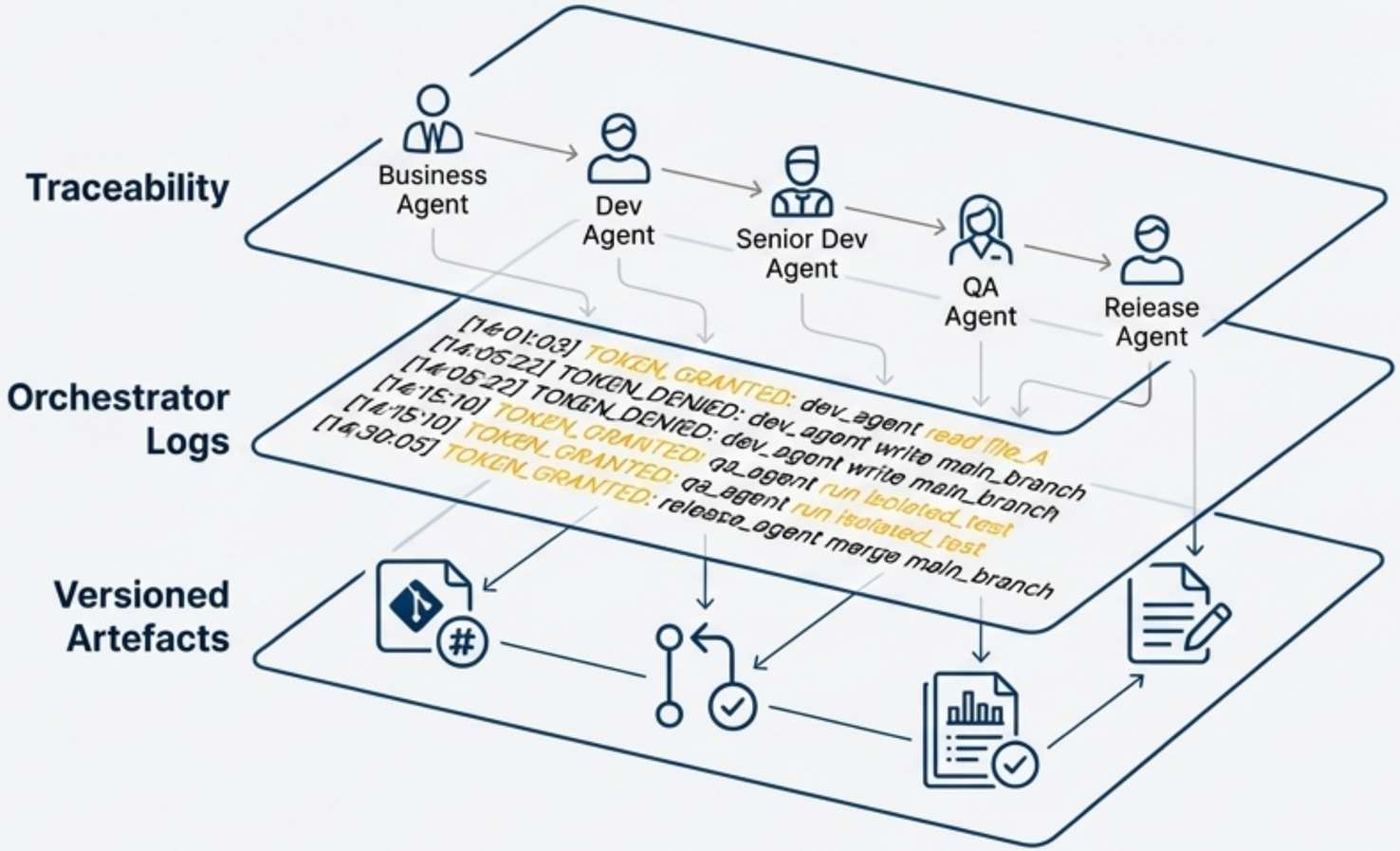
**Release Agent Merges:** **This is the crucial control:** Only the dedicated **Release Agent** can request merge permission, and only after all checks pass.

**Release Agent Deploys:** After a successful merge, the agent receives a final token to trigger the production deployment pipeline.

# The Result: A Complete and Verifiable Audit Trail

Throughout the workflow, every request, decision, and action has been logged, creating an immutable evidence package.

- **Orchestrator Logs:** A detailed record of every token requested, granted, or denied for each agent (e.g., 'Time X: developer agent requested read access to file A, approved').
- **Versioned Artefacts:** The Task Ticket conversation, the Pull Request with code diffs and reviews, and CI/CD test results are all preserved.
- **Traceability:** The entire process can be reconstructed to show what the agents did, why they did it, and how it was authorised.




This aligns with the best practice that *every action by an AI agent should be as traceable as actions by a human user*, enabling compliance and building trust in the autonomous system.

# Directly Mitigating Critical Agentic Risks

### Risk 1: Rogue Agent Behaviour (OWASP ASI10)

An agent cannot perform destructive actions outside its narrow, momentary grant. It would never receive a token to ‘delete all issues.’

### Risk 2: Identity & Privilege Abuse (OWASP ASI03)

Ephemeral, scoped tokens have minimal value if stolen. There are no static, ‘God mode’ API keys for an attacker to find.

### Risk 3: Tool Misuse & Exploitation (OWASP ASI02)

Every tool use is mediated by the orchestrator. An agent tricked into running a destructive command will be denied the token.

**Real-World Case**

Amazon’s poisoned AI coding assistant executed destructive AWS CLI commands because it lacked a sandbox. Our approach would have prohibited these unapproved commands entirely.

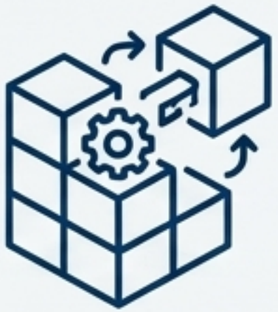
# An Enabler for Trustworthy and Scalable AI

## Audit and Compliance

The detailed audit trail is essential for regulated industries and provides a foundation for accountability.

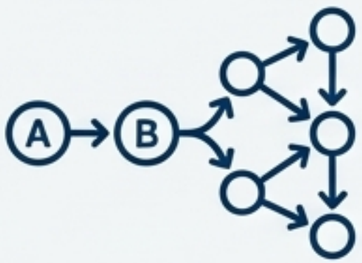
## Alignment with Human Processes

The system codifies best practices like separation of duties and mandatory reviews, ensuring AI agents adhere to the same guardrails expected of human teams.



## Flexibility with Safety

New agents (e.g., a Security Reviewer) or human approval steps can be easily inserted into workflows without compromising the security model.




## Scalability

The model provides a systematic way to manage complex, multi-agent collaboration without security becoming an unmanageable bottleneck.

## From Unpredictable Wildcards to Accountable Teammates

Secure agentic design is about balancing autonomy with control. The goal is not to limit AI, but to provide the structure necessary for it to be safely empowered.

# By giving our AI agents only what they need and nothing more, we make them accountable team players rather than unpredictable wildcards, ushering in a future of AI that is both empowered and aligned with human intent.