ChatGPT

Executive Summary

Multi-agent AI systems (agents communicating via social feeds, shared tool servers or delegated chains) introduce complex security and governance challenges. Attacks can exploit inter-agent channels (prompt injection, “memetic” spreading of malicious instructions), identity spoofing, and uncontrolled delegation, leading to unauthorized actions and data leaks. For example, a **recent research scenario** showed a malicious agent exploiting session-based trust to inject hidden commands into another agent’s multi-turn conversation [1]. Similarly, an analysis of the OpenClaw/Clawdbot platform found **30,000+ Internet-exposed instances** with weak controls, confirming that unprotected agent networks attract real attack traffic [2] [3].

To counter these threats, we map them to OWASP’s Agentic Top 10 and MITRE ATLAS AI-agent tactics, and propose a comprehensive governance framework. Key patterns include **scoped delegation tokens** (e.g. Google’s Biscuits/Macaroons) for provable least-privilege across hops [4] [5]; **out-of-band consent flows** (push notifications or separate UIs) for high-risk actions [6]; and enforced **audit trails** linking every agent action back to a human user. We detail **10 governance patterns** (Table 2) that cover identity verification, token attenuation, and session anchoring, balancing usability for non-experts with security needs.

Finally, we translate these into an **operational “Quest Pack” of 18 routine checks** (daily, weekly, and on-event) to detect and prevent issues before they become incidents. Each quest has a *safe agent lane* (only read-only or simulated actions), a structured proof artifact format (JSON/CSV summary of results), and clear escalation criteria. For example, a weekly “Delegation Audit Quest” inspects all active agent delegation tokens for overly broad scopes, flagging any that violate least-privilege (escalating to an admin). We also provide a **simple consent checklist** for product documentation, ensuring that end users understand delegation risks in plain language.

This report combines primary sources (OWASP, MITRE ATLAS, vendor docs, incident analyses) to ground each recommendation in evidence. By treating multi-agent interactions as a distinct threat domain, and by operationalizing security as daily/weekly habits (“quests”), organizations can enable agents to collaborate safely under human oversight [6] [1].

1. Threat Catalog

We enumerate the top 12 security and governance failure modes in multi-agent ecosystems. Each threat includes (1) Preconditions (when it can occur), (2) Impacts, and (3) Detection Signals. Table 1 summarizes these threats and references OWASP Agentic Top 10 (ASI) categories and MITRE ATLAS where applicable.

Threat	Preconditions	Impact	Detection Signals	Taxonomy Mapping
1. Inter-Agent Prompt Injection (Memetic Propagation) [1] [7]	Agents share context via feeds or protocol; input from one agent influences another.	Malicious instructions spread through agent chain; unwanted actions or data leaks.	Sudden unrelated actions by agent; new commands appearing in conversation logs.	OWASP ASI01 (Goal Hijack); ATLAS AI Agent Context Poisoning (AML.T0058) [7] .
2. Agent Session Smuggling (Context Poisoning) [1]	A stateful session (e.g. A2A chat) exists between agents; one agent is malicious.	Stealthy injection of hidden instructions mid-conversation, altering agent behavior.	Unexpected "extra" messages in conversation; inconsistencies between user intent and agent action.	OWASP ASI01 (Goal Hijack); ATLAS Thread Poisoning / Session Hijack techniques.
3. Impersonation / Identity Spoofing	Agents trust each other by default; no strong identity proof (AgentCard) is used.	Malicious agent pretends to be a trusted agent; user or system misattributes requests.	Mismatch in cryptographic identity tokens; alerts for duplicate agent IDs on network.	OWASP ASI03 (Identity Abuse); MITRE ATLAS (Agent Identity Spoofing techniques).
4. Delegation Chain Confusion (Confused Deputy) [4]	A planner agent delegates tasks to executors using shared tokens or context.	A compromised agent injects a task into the chain (e.g. the "buy_stock" hidden trade), inheriting upstream privileges [4] .	Detection of expanded scopes: executor using capabilities not intended by planner; audit logs showing unexpected chain.	OWASP ASI02 (Tool Misuse); ATLAS AI Agent Tool Invocation / Delegate (AML.T0058, Exfil via tools [8]).
5. Lack of Scope Attenuation [4]	Delegation tokens (e.g. OAuth) are reused across agents without narrowing privileges.	Downstream agents have equal or more privileges than upstream, enabling privilege escalation.	Audits reveal agents holding overly broad tokens; no evidence of token limitation.	ASI03 (Privilege Abuse); ATLAS "Excessive Delegation" concept (new technique).

Threat	Preconditions	Impact	Detection Signals	Taxonomy Mapping
6. Rogue Social Feed Content (Memetic Poisoning)	Agents consume user posts or open channels (e.g. Slack, forums). Malicious content is posted.	Poisonous instructions become memes that agents automatically amplify or execute, causing cascading failures.	Spikes in similar malicious outputs across agents; detection of viral malicious prompts.	ASI08 (Cascading Failures); ATLAS Activation Triggers (AML.T0059).
7. Unsafe Delegation Escalation (Chain Misuse) [9]	A verifying agent inadvertently gains executor rights or vice versa; unclear agent roles.	Agents perform tasks beyond their intended role (e.g. QA agent executing actions) leading to breaches.	Logs showing an agent performing out-of-role actions; absence of separation in agent design.	ASI02/ASI03 (Tool Misuse/Privilege); OWASP “unsafe delegation” (not fully covered), ATLAS Recursive delegation.
8. Policy Drift / Context Mismatch	Task context is shared among heterogeneous agents without alignment; no session anchoring.	Agents gradually drift from original task, leading to inappropriate actions (semantic privilege escalation).	Ambiguous or conflicting goals in logs; agent performing steps unrelated to initial request.	ASI09 (Trust Exploitation); ATLAS Context Drift detection.
9. Memory/ Config Poisoning [1] [7]	Agents share memory or configuration (e.g. RAG databases, config files).	Poisoned memory or config alters behavior of multiple agents over time (persistent backdoor).	Unexpected changes in shared memory or RAG results; suspicious entries in config files (e.g. new URLs/keys).	ASI06 (Memory Poisoning); ATLAS Modify AI Agent Config (AML.T0052), Memory.
10. Shared Tool/ Server Compromise	Multiple agents rely on a common tool server (MCP registry, API hub).	Compromise of this hub poisons all connected agents with malicious tools or responses.	Simultaneous anomalies across agents using the same server; signature mismatches in tool definitions.	ASI04 (Supply Chain); ATLAS Tool/Definition Poisoning (e.g. AML.T0053).

Threat	Preconditions	Impact	Detection Signals	Taxonomy Mapping
11. Unintended Cross-Agent Coordination [10]	Compliant agents independently split tasks in parallel (e.g. to optimize).	A benign chain-of-responsibility inadvertently bypasses safeguards (e.g. splitting policy checks), or agents coordinate covertly.	Unusual multi-agent traffic patterns; tasks completed via unexpected delegation graph.	ASI08 (Cascading); ATLAS Emergent Coordination (not explicitly named in ATLAS yet).
12. Excessive Outgoing Permissions	Agent has rights (e.g. email, webhook, FS writes) that can reach external systems.	An agent uses its authorized output channels to exfiltrate data or trigger harmful external actions.	Large or high-frequency external messages (e.g. many emails sent); network logs showing data leakage from agent.	ASI02/ASI03 (Tool Misuse/Privilege); ATLAS Exfiltration via Tool Invocation (AML.T0072) [8].

Table 1: Summary of multi-agent security threats (details below). Each threat is mapped to OWASP Agentic Top10 categories and MITRE ATLAS AI-agent TTPs where applicable [11] [7].

Threat Descriptions

1. **Inter-Agent Prompt Injection (Memetic Propagation)**: When agents pass context (e.g. via chat, shared memory), malicious actors can inject instructions that propagate through the agent network [1]. For instance, a poisoned news snippet or social media post could carry embedded commands that each agent in the chain executes. *Impact:* widespread unauthorized actions or leaks across agents. *Detection:* monitoring chat logs for unusual commands; anomaly detection if agent outputs deviate semantically from intent.

2. **Agent Session Smuggling**: In stateful agent-to-agent protocols (e.g. A2A), a malicious agent injects hidden instructions mid-session [1]. For example, a remote planner sends normal responses but secretly slips instructions back to the client. *Impact:* context poisoning and data exfiltration without user noticing [12]. *Detection:* verify that multi-turn responses remain on-topic; use signed session tokens (AgentCards) to detect unexpected directives [13].

3. **Impersonation/Identity Spoofing**: Without cryptographic identity checks, a rogue agent can pose as a trusted agent. For instance, a tool server could publish an `AgentCard` with misleading metadata (akin to "AgentCard Shadowing" [14]). *Impact:* execution of harmful commands under false identity, bypassing trust. *Detection:* require signed agent identities, monitor for duplicate or similar agent names (typosquatting) [15].

4. **Delegation Chain Confusion (Confused Deputy)**: When one agent delegates tasks to another, a malicious intermediary can inject extra tasks. A research scenario showed a “research” agent inheriting a hidden `buy_stock` command from a financial advisor agent [4]. *Impact:* unauthorized operations executed using upstream privileges. *Detection:* token lineage checks; flag when delegated actions exceed planner’s scope.

5. **Lack of Scope Attenuation**: Delegated tokens often do *not* narrow with each hop. An executor may retain full user privileges instead of just the subtask scope. *Impact:* privilege escalation across delegation chains. *Detection:* audit tokens: any agent holding broader scopes than intended is a red flag [4].

6. **Rogue Social Feed Content (Memetic Poisoning)**: Agents ingest and act on popular content. A viral malicious prompt (e.g. on Slack) could be automatically consumed and re-executed by many agents, amplifying harm (cascading failure [16]). *Detection:* content filtering, quarantining suspicious feeds, tracking agent performance before/after feed events.

7. **Unsafe Delegation Escalation**: When roles blur (e.g. a verification agent is tricked into executing a task), restrictions collapse. If a verifier suddenly has executor privileges, it can be coerced by the planner to do destructive work. *Detection:* enforce separation of duties between agent roles.

8. **Policy Drift / Context Mismatch**: Without anchoring, multi-agent workflows can “drift” – each agent adds its own spin. Over time, actions deviate from original intent (a known semantic escalation issue [17]). *Detection:* continuous context-validation: checkpoints where output is checked against original goals.

9. **Memory/Config Poisoning**: Shared knowledge bases or config files can be tampered with. For example, modifying a common RAG database or agent config affects every agent using it [7]. *Detection:* versioning and integrity checks on config and memory stores.

10. **Shared Tool/Server Compromise**: If multiple agents fetch tools from a central registry (MCP), a “trusted” tool server could turn malicious (Rug Pull) or be shadowed by a lookalike tool [18]. *Detection:* signed tool manifests, replica-checking of tool content.

11. **Unintended Cross-Agent Coordination**: Even agents all following rules can inadvertently coordinate to bypass checks (e.g. splitting a forbidden query across agents). *Detection:* unusual multi-agent patterns, requiring cross-checks on coordination.

12. **Excessive Outgoing Permissions**: Agents given broad outgoings (like email or FS writes) might abuse them. E.g. authorized `send_email` can be used to exfiltrate confidential data by encoding it in a message. *Detection:* rate-limit and inspect high-volume external actions, log all data outputs.

2. Governance Pattern Library

To mitigate these threats, we propose 10 governance patterns that govern agent delegation, consent, and identity. Each pattern is described with its mechanism, UX considerations, and where it suits self-hosted consumer agents (e.g. personal assistants). Table 2 compares them.

Pattern	How It Works	UX Trade-offs	Use Case / Suitability
Scoped Delegation Tokens	Use attenuated tokens (e.g. Google Biscuits/Macaroons [19]) that chain delegations with *non-expandable* scopes. Each hop can only add restrictive caveats. Cryptographically verifiable offline.	Users grant broad consent once (usability), while actual agents automatically narrow permissions (security). Managing token formats may add complexity.	Suitable for workflows where an agent delegates multiple subtasks; fits in enterprise settings with existing OAuth.
Out-of-Band Approval (E2E)	Require human approval for sensitive steps via a separate channel (push notification or email), not through agent chat. E.g. Auth0 Async Auth [20].	Delays operations but ensures true user intent. May frustrate users if overused.	High-risk actions (finance, admin tasks); when agents share mobile apps, etc. Improves trust without complex crypto.
Context Anchoring	At session start, establish an immutable "task anchor" (summary of intent). All agent outputs are checked against this anchor (context grounding) [21].	Adds behind-the-scenes validation; transparent to user, though sessions may have safety interruptions if drift detected.	Multi-step tasks where semantic drift is a concern; e.g. planning delegations. Good default for long-running workflows.
Agent Identity Certification	Agents present signed, verifiable identities (AgentCards [13]). Remote agents only accept tasks from known identities.	Initial setup may require certificate exchanges (usability hit), but improves mutual trust greatly.	All inter-agent communication in enterprise networks; federated agent ecosystems.
Audit Trail & Visibility	Record every agent action and delegation in an immutable log tied to user IDs. Provide dashboards for humans to review actions.	Minimal user friction; users gain reassurance. Requires log management.	Universal. Essential for compliance (required by EU AI Act Article 14 [22]).
Consent with Scope & Expiry	When user grants permission to an agent, require specifying what (scope) and how long (time-bound). After expiry, capability auto-revokes.	Users must understand scopes (higher complexity). Friendly UIs can help (sliders, default durations).	Consumer agents requiring sensitive data (calendar, email); ensures consent is never "forever".
Delegation Chain Verification	Use cryptographic delegation tokens (e.g. Wafers [5]) that embed full history of who delegated what. Agents can prove the chain at each hop.	Mostly invisible to users, but demands rigorous identity management and key handling.	Scenarios where unbroken chain-of-trust is critical (e.g. automated purchasing, medical advice delegation).

Pattern	How It Works	UX Trade-offs	Use Case / Suitability
Least-Privilege Defaults (Safe Mode)	Agents start in safe mode with minimal capabilities (e.g. read-only, `net_read`, no `exec_shell`). New capabilities require explicit human enable (Authorized Mode).	Users must manually elevate when needed (inconvenient for power users). But reduces accident risk for casual use.	Best for consumer/ self-hosted agents where security is a concern; any agent with auto-learning features.
Revocation Mechanisms	Provide easy ways for users to revoke agent permissions (e.g. "revoke" command, UI button) that immediately halts delegation chains [23].	Needs visible controls; users might not want complexity. Crucial for regaining control after incidents.	All user-facing agents; must be prominent especially in personal assistants.
Transparent Agent Composition	When an agent delegates to another, the UI should display "Agent A -> Agent B" so user knows chain. Show origin of each action.	Requires UI/UX development, but greatly enhances trust. May clutter interface.	Collaborative agent scenarios (e.g. multi-bot Slack); always-on desktop/mobile assistants.

Table 2: Governance patterns for multi-agent delegation and consent. Each pattern balances security vs usability for self-hosted agents.

Governance Pattern Details

1. **Scoped Delegation Tokens**: Inspired by Google's *Macaroons/Biscuits* [5], each delegation grants an append-only caveat, narrowing privileges. For example, a token might start with `user:read` scope; when agent B is given a subtask, it adds a caveat restricting to `read:calendar`. This enforces least-privilege across hops. The UX impact is minimal once set up, as users grant broad consent initially but agents are auto-limited internally.

2. **Out-of-Band Approval (E2E)**: Critical steps (e.g. "transfer $1000") trigger a push notification or email to the user's device, requiring confirmation outside the chat interface [6]. Unlike in-chat approvals (which could be spoofed via prompt injection), this uses independent channels (Okta calls it *Asynchronous Authorization* [20]). The trade-off is latency: each such action pauses the task, but it thwarts any in-band hijack.

3. **Context Anchoring**: At the start of a multi-agent workflow, the original user goal is stored and cryptographically bound to the session. Downstream agents compare new tasks against this anchor. If an agent's subtask deviates semantically, it is rejected. This dynamic check (similar to Unit42's recommendation [21]) adds invisible safeguards: the user isn't bothered unless drift occurs.

4. **Agent Identity Certification**: Agents present digitally signed identity tokens (e.g. "AgentCard" certificates [13]). Every agent's public key and role is registered in an Identity Authority. When an agent A delegates to B, A verifies B's signature. UX-wise, obtaining and installing these identities

requires a one-time process, but after that agents only accept delegation from known peers, preventing simple spoofing.

5. **Audit Trail & Visibility**: All agent decisions and delegation paths are logged in a secure ledger. Users can review logs like "Agent A asked Agent B to read file X at time Y." This transparency lets non-experts detect anomalies with minimal expertise. Modern standards (EU AI Act Article 14) require such traceability [22], so this pattern has regulatory support.

6. **Consent with Scope & Expiry**: For any requested permission, require user to explicitly confirm a defined scope (e.g. "Agent can read calendar events") and optionally set an expiration time. The agent system enforces these limits automatically. Onboarding UI might provide default scopes and durations to guide novices. This pattern ensures delegation isn't open-ended (in line with Okta's timeline emphasis [23]).

7. **Delegation Chain Verification**: Similar to certificate chains, each agent attaches a proof of who delegated to it. Using "Wafers" or similar constructs [5], every step is verifiable offline. Non-experts simply see a consistent chain of "signed tasks," giving confidence. Any break (e.g. if an agent misses a signature) halts the flow.

8. **Least-Privilege Defaults (Safe Mode)**: By default, put agents in **Safe Mode** with only non-invasive capabilities: e.g. `list_integrations`, `read_config`, `net_read`, and `fs_read`. Sensitive ops (`exec_shell`, `net_write`, `fs_write`) are disabled until explicitly enabled. This mirrors the "rule of two" idea (no untrusted input + sensitive data + action at once [24]). It means the average user benefits from security by default, with a clear toggle to enter "Authorized Mode" (for advanced usage).

9. **Revocation Mechanisms**: Provide a clear "Revoke Agent Access" command or UI that immediately invalidates the agent's credentials and stops ongoing tasks. Combined with short-lived tokens, this ensures that misbehavior can be stopped by even non-experts. (Think of it like a "kill switch" on an autonomous script.)

10. **Transparent Agent Composition**: All user-facing interfaces (chat UI, dashboards) should label agent interactions with their origin: e.g. "MetaBot → DocAgent → EmailTool". This contextual labeling helps users see the flow of actions. While it requires thoughtful UX design, it prevents "mystery" actions by agents and educates users on delegation.

3. Quest Pack Proposal

We propose **18 operational quests** to embed these controls into daily practice. Each quest is a short task (daily, weekly, or on-event) that a human or "guardian agent" performs to check security hygiene. The quests focus on delegation, consent, and agent behaviors. Table 3 summarizes each quest with frequency, objective, agent-lane steps, proof artifact, and escalation criteria.

Quest	Freq	Objective	Agent-Lane Steps (Safe)	Proof Artifact	Escalation Rule
1. **Delegation Audit**	Weekly	Verify all active agent delegation tokens and scopes	Use a read-only script to list all tokens/ capabilities each agent holds. Compare to intended scopes.	CSV of agent→capabilities	Any token exceeding its agreed scope flagged
2. **Identity Verification Check**	Weekly	Ensure agents' signed identities are valid and current	Fetch all agent identity certificates and check signatures against Authority key.	JSON report of agent statuses	Any agent with an invalid or missing cert → alert
3. **Consent Scope Review**	Weekly	Review user-granted consents and expirations	List all user consents (from IAM logs) with scopes and expiry dates. Highlight expired/ granted.	Consent list CSV	If broad consent lingers >1 month, escalate
4. **Revocation Drill**	On-Event	Test revocation process	Intentionally revoke an agent's permission via API. Confirm the agent can no longer act.	Success/fail status	If agent still acts after revoke → immediate alert
5. **Social Feed Sanitization**	Daily	Check for malicious content in agent-shared feeds	Scan latest posts/feeds for known malicious patterns (regex for "openpuppy" etc).	List of flagged posts	If any flagged, quarantine and human review

Quest	Freq	Objective	Agent-Lane Steps (Safe)	Proof Artifact	Escalation Rule
6. **Prompt Injection Test**	Weekly	Simulate a prompt-injection payload through agent network	Send a benign “injection” test payload to a non-critical agent workflow. Observe output for misbehavior.	Output diff log	If agent executes test injection, flag
7. **Delegation Reproducibility**	Monthly	Ensure delegation tokens can be audited	Pick a completed task chain and trace its token chain (using token proofs). Verify lineage.	Chain-graph (Mermaid)	If lineage breaks or missing, investigate
8. **Safe Mode Validation**	Weekly	Confirm Safe Mode capabilities are enforced	In Safe Mode, attempt a forbidden action (e.g. `fs_write`). Confirm it’s blocked.	Command response log	If action succeeds in Safe Mode, escalate
9. **Authorized Mode Approval**	On-Event	Ensure human approval is required for high-risk ops	As a test, request a high-risk action (e.g. “delete database”) and verify prompt for approval.	Approval-request snapshot	If no user prompt appears, immediate halt
10. **Audit Log Review**	Daily	Inspect logs for anomalous agent behavior	Parse last 24h agent logs for error messages or unusual patterns (spike in errors, new endpoints).	Alert summary	If anomalies found, human triage required

Quest	Freq	Objective	Agent-Lane Steps (Safe)	Proof Artifact	Escalation Rule
11. **Tool Server Integrity Check**	Weekly	Verify shared tool server hasn't been tampered	Fetch tool definitions from the MCP or registry and compare checksums to known-good.	Diff report	If differences in critical tool schema, alert
12. **Capability Whitelist Check**	Daily	Ensure only allowed capabilities are active	Query agent config for enabled capabilities and compare against a safe whitelist.	Capability status CSV	If any disallowed capability active, block
13. **Context Anchor Test**	Monthly	Validate context anchoring mechanism	Start a dummy multi-step task, then deviate instructions. Ensure anchor prevents drift.	Test transcripts	If agent accepts deviation, escalation
14. **Impersonation Detection**	On-Event	Detect unknown agents on network	Use discovery (AgentCard listing) to identify any new/ unregistered agents communicating.	Discovery report	If an unknown agent is found, isolate it
15. **Shared Memory Scan**	Weekly	Check shared knowledge bases for unauthorized edits	Compare current RAG or memory DB to baseline; flag any unexpected entries (especially URLs/exec).	DB diff report	Unexpected entries trigger audit
16. **Delegation Compliance**	Weekly	Ensure tokens follow attenuation rule	Inspect live tokens to verify each delegation is narrower than its issuer.	Token trace report	Non-attenuated hop causes escalation

Quest	Freq	Objective	Agent-Lane Steps (Safe)	Proof Artifact	Escalation Rule
17. **Chain Reaction Scenario**	On-Event	Simulate malicious content propagation	Introduce a benign “virus” post into feed; observe if multiple agents adopt dangerous behaviors.	Timeline of agent actions	If >1 agent is affected, alert
18. **Incident Response Drill**	Monthly	Full test of governance processes	Inject a mock breach (e.g. unauthorized agent command) and run full playbook response.	Incident report	Review completeness (any missed step)

Table 3: Security “quests” for multi-agent governance. Each quest defines a safe agent-lane *(only monitoring or simulation actions), the expected proof output format (JSON/CSV or diagrams), and clear escalation conditions for human review.*

Sample Quest: Delegation Audit (Weekly)

- **Agent-Lane Steps (safe):** The supervisory agent runs `list_delegations()` API to collect all active delegation tokens. For each, it invokes a *simulate* function (no real action) to test the token’s permissions against a minimal whitelist (just `read_config`, etc.).
- **Proof Artifact:** A CSV report with columns: *Agent ID, Token ID, Allowed Scopes, Intended Scopes, Status*.
- **Escalation:** Any row where *Allowed Scopes* ≠ *Intended Scopes* is flagged, triggering a human-admin alert to investigate.

4. Capability Model (Safe vs Authorized Mode)

We recommend a minimal capability taxonomy for agents (e.g. `list_integrations`, `read_config`, `net_read`, `net_write`, `fs_read`, `fs_write`, `exec_shell`, etc.) and define defaults:

- **Safe Mode (Default):** `list_integrations`, `read_config`, `fs_read`, `net_read` only. (No shell execution, no writes.) Agents can inspect context but cannot alter it or send network data.
- **Authorized Mode (Human-approved):** Grants additional caps as needed: `fs_write`, `exec_shell`, `net_write`, etc. Human gate ensures critical actions are deliberate.

Under this model, *by default* every agent runs in a locked-down environment. If an authorized user explicitly requests new rights (via consent UI), the system switches that agent to Authorized Mode with a scoped, time-limited token.

5. Usability Guidance for Non-Experts

- **Expose Decisions Clearly:** Use plain language when requesting approvals (e.g. “Allow Agent Alice to schedule all calendar events this week?”) and provide simple on/off toggles for agent permissions.
- **Educate on Scope:** Show examples of “scope” at grant time (“This allows reading only your calendar events”). Avoid technical jargon in UI.
- **Logs as Stories:** Present audit logs in story form (e.g. “Yesterday at 3pm, Agent Bob asked Agent Carol to send an email; you approved”). This helps non-experts follow chain of events.
- **Default Safety:** Keep Default Mode safe. Don’t ask the user about Safe Mode; just require extra taps to elevate, so casual users stay protected without extra effort.
- **Accessible Revocation:** Provide a big “Revoke All Agents” button; users should feel in control at all times.

By combining these patterns and practices, non-technical stewards can safely manage multi-agent systems through transparent controls and regular “quests” that guard against complex, cascading failures.

```
flowchart LR
    subgraph Delegation Chain
      user[User] --> |"delegate task"| agentA[Agent A<br/>(Planner)]
      agentA --> |"delegate subtask"| agentB[Agent B<br/>(Executor)]
      agentB --> |"invoke tool"| Tool[External Tool]
      agentB --> |"return result"| agentA
      agentA --> |"complete task"| user
    end
    user -- Revocation -->|"revoke token"| agentA
    user -- Audit -->|"view logs"| Logs[Audit Log]
```

Figure: Delegation chain example with user oversight. Each arrow is a delegation or action; user can revoke tokens and view an audit log of the chain.

```
sequenceDiagram
    participant User
    participant Agent
    participant Chain as "Consent UI"
    User->>Agent: "Request sensitive action"
    Agent->>Chain: Display consent prompt
    Chain->>User: "Do you approve? (Yes/No)"
    User-->>Chain: "Yes"
    Chain->>Agent: "Approval granted (expires 1h)"
    Agent->>Agent: Perform action
```

Figure: Consent flow. The agent requests permission through an external UI; the user grants scoped consent which the agent then uses.

Sources: The threats and controls above draw on the OWASP Agentic Top10 [11], MITRE ATLAS (AI-Agent TTPs) [7] [8], vendor guidance (Okta/Auth0 on delegation [4] [6]), and incident analyses (Palo Alto Unit42 on session smuggling [1], Zenity Labs on agent hijacking [25], Bitsight on exposed agents [2] [3]). Each quest and pattern is evidence-based to ensure comprehensive multi-agent security.

---

[1] [12] [13] [14] [15] [18] When AI Agents Go Rogue: Agent Session Smuggling Attack in A2A Systems
https://unit42.paloaltonetworks.com/agent-session-smuggling-in-agent2agent-systems/

[2] [3] OpenClaw Security: Risks of Exposed AI Agents Explained | Bitsight
https://www.bitsight.com/blog/openclaw-ai-security-risks-exposed-instances

[4] [5] [6] [19] [20] [21] [22] [23] Agent Session Smuggling: Fixing AI Delegation Risks | Okta
https://www.okta.com/blog/ai/agent-security-delegation-chain/

[7] [8] Zenity & MITRE ATLAS Expand AI Agent Attack Coverage
https://zenity.io/blog/current-events/zenity-labs-and-mitre-atlas-collaborate-to-advances-ai-agent-security-with-the-first-release-of

[9] [10] Extending the OWASP Multi-Agentic System Threat Modeling Guide: Insights from Multi-Agent Security Research
https://arxiv.org/html/2508.09815v1

[11] OWASP Top 10 for Agentic Applications - The Benchmark for Agentic Security in the Age of Autonomous AI - OWASP Gen AI Security Project
https://genai.owasp.org/2025/12/09/owasp-top-10-for-agentic-applications-the-benchmark-for-agentic-security-in-the-age-of-autonomous-ai/

[16] [17] [24] The Clawdbot Dumpster Fire: 72 Hours That Exposed Everything Wrong With AI Security - Acuvity
https://acuvity.ai/the-clawdbot-dumpster-fire-72-hours-that-exposed-everything-wrong-with-ai-security/

[25] Research shows AI agents are highly vulnerable to hijacking attacks | Cybersecurity Dive
https://www.cybersecuritydive.com/news/research-shows-ai-agents-are-highly-vulnerable-to-hijacking-attacks/757319/