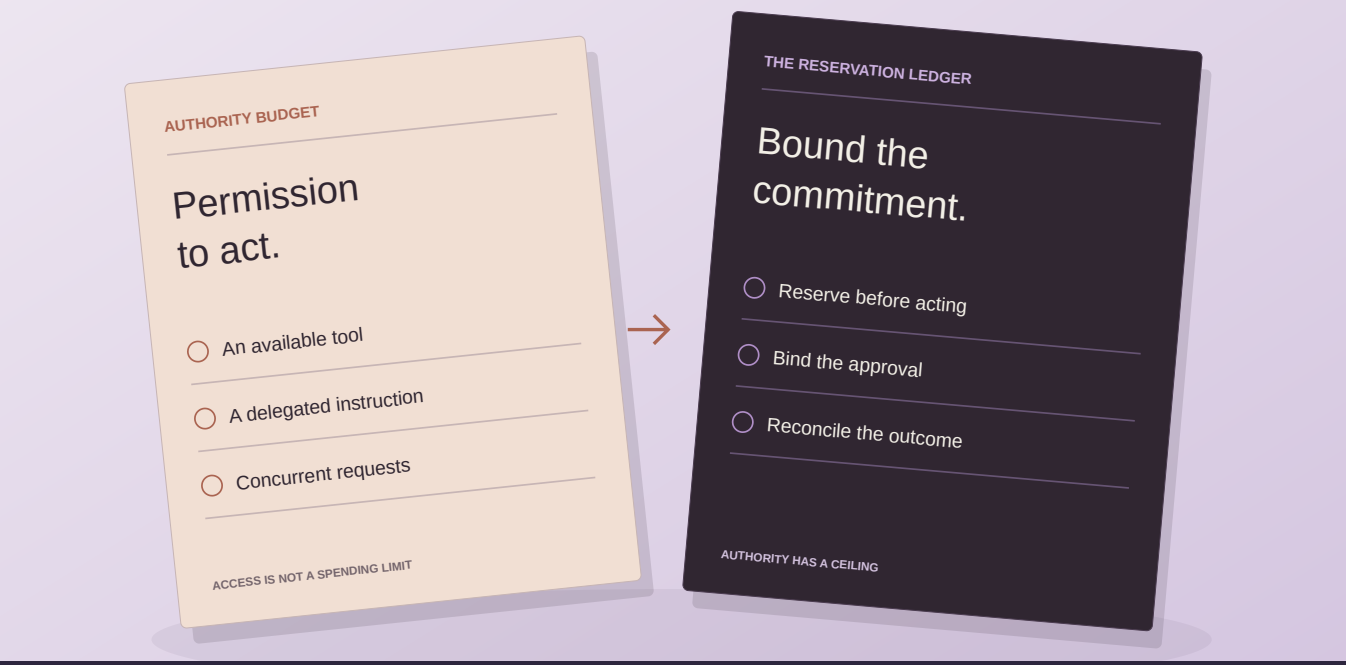
kymata labs

RESEARCH PAPER 08 / AGENT SYSTEMS & CONTROL

Give Your Agent a Budget Before You Give It Authority

A spending limit belongs in the executor, not just in the prompt.



Kymata Labs Research
2026-11-16 · Edition 1.0

Independent analysis
kymatalabs.com

Small studio. Open worlds.

THE LONGER VIEW

Executive summary.

An agent can obey every local instruction while a shared workflow overspends. The failure occurs when several requests allocate the same remaining capacity. This paper separates identity, action permission, approval and aggregate budget, then develops a reservation boundary that makes confirmed spending and unresolved commitments visible together. It examines inspected MCP, OWASP and Agents SDK sources without treating their different controls as interchangeable. A reproducible Python fixture tests concurrent reservations, changed approval terms and uncertain outcomes using fictional units and no external calls. The paper also explains where the design stops: a local ledger cannot enforce an unbounded provider charge, replace authorization, or settle an external outcome it does not know.

> Analysis of exact primary-source repository snapshots captured November 16, 2026: OWASP Excessive Agency guidance, MCP 2025–06–18 materials and OpenAI Agents SDK documentation. A separate Python standard-library SQLite fixture executed 20 rounds of 20 concurrent reservation attempts and selected policy assertions. Amounts are fictional integer units; no model, payment or provider API was called. The proposed production controls extend beyond the deliberately bounded local fixture. No claim of distributed-failover validation, provider performance or universal prevention is made.

01 / The instruction is not the limit

02 / Authority has more than one dimension

03 / The missing number is committed money

04 / Checking is not reserving

05 / Approval must travel with the proposal

06 / The gate must precede the consequence

07 / A local experiment with twenty contenders

08 / Unknown is an accounting state

09 / The case against turning every agent into a purchasing department

10 / A small transaction can still exceed the mandate

11 / What to require before the first real commitment

Q&A / Questions worth asking

Sources / References

© 2026 Kymata Labs LLC. All rights reserved.
Read the web edition: kymatalabs.com/research/give-your-agent-a-budget-before-you-give-it-authority

The instruction is not the limit

Give an agent permission to buy something and a sentence telling it to be careful. You have created a purchasing assistant. You have not yet created a spending limit.

Consider a deliberately hypothetical procurement task. An assistant has a ceiling of 10,000 units and can place orders with an approved supplier. Twenty workers each find something costing 700. Each checks a balance that still appears sufficient. Each order is individually reasonable. Together they can commit 14,000 units. No worker needs to hallucinate, ignore an instruction, or encounter a malicious webpage. The mistake lives in the system that allowed several promises to spend the same money.

The arithmetic is simple because the design failure should be visible. A budget described in a prompt is information the model can consider. A budget enforced at the point of execution is a property the application must maintain. These are useful, different things. The first helps the assistant make better proposals. The second decides which proposals can become commitments.

This paper argues for a specific boundary: **reserve the maximum authorized exposure before executing an action, and make permission, approval and reservation agree on the same action.** The model can explain its plan. It cannot be the only component deciding whether its plan is allowed. The proposal is an engineering control, not a claim that all agents are reckless or that every tool call needs human review.

Authority has more than one dimension

OWASP's 2025 Excessive Agency guidance distinguishes excessive functionality, permissions and autonomy. Those categories matter because they describe different ways to make a system too powerful. A read-only task can receive a tool that also deletes records. A narrowly named tool can run under an unnecessarily privileged account. A correctly scoped account can still perform a high-impact action without appropriate approval. OWASP recommends minimal tools and permissions, user context, approval for high-impact actions, and authorization enforced in downstream systems. [1]

A financial ceiling adds a fourth practical question: how much exposure may accumulate? A per-call permission does not answer it. Nor does a rate limit. Ten allowed actions can be cheap or expensive; one allowed action can be irreversible without costing money. A robust policy needs both qualitative boundaries and quantitative limits.

Boundary	Question it answers	Question it leaves open
Identity and scope	Which principal may use this resource?	Is this particular purchase appropriate?
Action policy	Which operation, destination and data are permitted?	How much has already been committed?
Approval	Did an authorized reviewer approve this proposal?	Is the approved proposal still the one being executed?
Budget reservation	Does this commitment fit the remaining ceiling?	Will the external system honor the maximum?

The distinction is not an argument against standards. It is a reason to use them for their actual jobs. The inspected MCP authorization specification, version 2025-06-18, describes access to protected resources and requires token audience validation. It does not provide the application-specific purchasing ledger proposed here. Being entitled to reach a server and being entitled to spend another 700 units are separate decisions. [2]

MCP's tool specification also recommends a human ability to deny invocations and tells clients to treat tool annotations as untrusted unless they come from trusted servers. A tool description is not a reliable substitute for enforcement. A purchasing gateway should know which destinations and operations it supports independently of whatever description is presented to the model. [3]

FIGURE 01 CONTROL BOUNDARY

Three gates before a commitment.

Permission, approval and budget answer different questions. All must describe the same proposed action.



Original application design. Protocol access and SDK approval mechanisms supply parts of the boundary, not an automatic aggregate spending policy.

Kymata Labs synthesis of inspected OWASP, MCP and Agents SDK materials. [1] [2] [3] [4]

The missing number is committed money

A dashboard that counts only completed charges can look reassuring while an agent has many requests in flight. The useful invariant is that confirmed spending plus outstanding reservations cannot exceed the authorized ceiling. Available capacity is what remains after both have been deducted.

A reservation is a claim on capacity, not evidence that money has been spent. Suppose an operation can cost at most 700 units. Reserving 700 prevents another operation from claiming that capacity while the first is unresolved. If the confirmed cost is 500, settlement moves 500 into spending and releases 200. If the operation is definitively canceled before any effect can occur, its reservation can be released. If the outcome is unknown, releasing it would pretend that uncertainty is free.

The word *maximum* does real work. An estimated cost is useful for planning but cannot support a hard bound unless overruns are handled. A purchase may involve delivery, tax, exchange rates, quantity changes or a later adjustment. A compute task may have a variable output length or invoke another service. The application needs either an enforceable maximum that includes relevant charges, a conservative upper bound accepted by policy, or another review before the unbounded part begins. A local ledger cannot prevent an external provider from charging beyond the amount the ledger assumed.

This is why a budget should be specified in a defined unit and scope. A monthly dollar ceiling, a per-task token allowance and a limit on emails sent are different budgets. They should not be added together. Currency conversion requires an explicit rate and policy; token estimates require a declared pricing basis. The example in this paper deliberately uses fictional integer units to make the accounting reproducible without importing changing prices.

A ceiling also needs an owner and a period. If a job crosses midnight, resetting its account must not make yesterday's unresolved commitments disappear. If several agents act for one department, giving each the department's full allowance multiplies exposure. They must reserve from a shared pool or receive non-overlapping allocations. Hierarchical limits can be useful, but every applicable ceiling needs consistent enforcement when a commitment is admitted.

Checking is not reserving

The concurrency error arises when an application separates its check from its update. Two requests read the same available balance, both pass, and both record a commitment. Each individual check was accurate at the instant it ran. The combined result is wrong.

The remedy is not a more emphatic prompt. The admission decision and the creation of the reservation must form one indivisible operation within the storage system's consistency boundary. In the local fixture, a SQLite immediate transaction serializes competing writers. Each request observes the committed reservations left by earlier admitted requests before deciding whether there is room for another. That implementation is small enough to inspect; it is not a prescription for every deployment. [7]

A distributed service may instead use a conditional update, a transaction covering the relevant accounts, or an admission service that owns the ledger. Its correctness depends on the guarantees of that mechanism, including failure and recovery. A process-local lock cannot coordinate unrelated processes on different machines. An eventually consistent dashboard is useful for reporting and may be unsuitable as the admission authority.

Every reservation also needs a durable operation identity. A worker recovering from a crash should discover the reservation it already owns, not silently create another. The local fixture binds an operation identifier to a canonical payload fingerprint. Repeating the same request returns its existing state; changing the payload under the same identifier is rejected. This protects reservation accounting. It does not, by itself, make an external purchase idempotent. The separate question of avoiding duplicate external effects is developed in the companion paper, *The Retry Button Can Spend Twice*.

Approval must travel with the proposal

A person approving “buy the replacement” has not necessarily approved every later interpretation of that instruction. The destination can change. The quantity can grow. A different agent can resume the task. The application must decide which changes preserve consent and which require another decision.

A practical approval record should bind the reviewer, authorized principal, operation, destination, relevant payload, maximum exposure, policy version and validity window. That is a proposed application contract, not a claim that an SDK automatically supplies all those fields. The reviewer must see a comprehensible summary of the consequential values. A hash can detect changes after approval; it cannot make a misleading approval screen truthful.

The inspected OpenAI Agents SDK documentation provides a useful implementation surface: tools can declare that approval is required, paused runs expose interruptions, and stored run state can be resumed after decisions. It also distinguishes a per-call decision from persistent approval of a tool identity. Persistent approval is broader authority. The captured documentation specifically warns about how sticky shell and file-edit approvals can apply across agents that share approval state. Those details are version-specific; the general lesson is to inspect what an approval actually names and how long it survives. [4]

This need for binding is older than agents. MCP's security guidance describes a confused-deputy scenario in which a proxy's handling of client identity and prior consent can let a different client benefit from consent that was not given to it. The cited scenario concerns OAuth proxy authorization, not a spending exploit. It nevertheless illustrates why “someone approved something earlier” is inadequate as an authorization record. [6]

A review must not reserve unlimited future authority by accident. For a bounded repeatable workflow, standing permission may be exactly what the user wants. Define its destination set, transaction ceiling, aggregate ceiling and expiration explicitly. For a novel high-impact action, a per-call decision is often easier to reason about. Neither policy should be hidden behind an ambiguous button.

The gate must precede the consequence

“Guardrails are enabled” is not a complete answer to where enforcement happens. An input check, an output check and a tool execution check occupy different places in the workflow. A check that rejects the final explanation cannot undo a purchase already made.

The captured Agents SDK guardrail documentation makes this ordering concrete. Its default parallel input guardrail may run while the agent has already consumed tokens or executed tools. Blocking input mode runs the check first. Agent-level output guardrails run after the agent completes. Function-tool input guardrails can run before tool execution, while other tool families use different execution paths. These are documented behaviors of the inspected version, not a comparative product test. [5]

For the proposed budget design, the decisive admission check belongs in the executor or downstream gateway that controls the side effect. A model-side check can improve the proposal and reduce needless interruptions. It cannot replace the gateway's atomic reservation. A final-output check can prevent a misleading message. It cannot be the gate for an earlier irreversible operation.

This has a mundane operational implication: keep credentials that bypass the gateway out of the proposing component. If an assistant can choose either the controlled purchase tool or an unrestricted shell that uses the same payment credentials, the controlled tool is optional. OWASP's recommendation to minimize open-ended functionality and enforce downstream authorization speaks directly to this problem. [1]

The policy service also needs an availability decision. If it cannot read or update the authoritative ledger, allowing a purchase on the basis of a stale local estimate abandons the ceiling. For spending commitments, this design stops admission and reports the reason. Read-only work can continue when its own permissions and costs allow it. A budget failure should not become a mysterious conversational refusal.

A local experiment with twenty contenders

The accompanying Python standard-library fixture is an executable illustration of the accounting boundary. Download the example listed in reference 7 and run `python3 example.py`. It creates temporary SQLite databases, performs assertions, and writes `result.json` beside the script. It calls no model, service or payment API. Its amounts are fictional units, and its results are not provider benchmarks. [7]

Each of twenty rounds starts with a 10,000–unit ceiling. Twenty threads attempt distinct 700–unit reservations concurrently. The transaction admits fourteen requests, reserving 9,800 units, and denies six for insufficient budget. Across the measured run, 400 requests produce 280 accepted reservations and 120 budget denials. Every round stays within the ceiling. Thread scheduling is real; the business scenario is simulated.

The fixture then repeats an accepted operation and asserts that its reservation does not grow. Ten accepted operations are dispatched and settle at 500 units each. Four are dispatched and marked unknown. Spending is now 5,000, unresolved reservations total 2,800, and available capacity is 2,200. The unknown operations remain funded until their outcomes can be resolved.

Fixture observation	Measured result	Meaning
Competing requests per round	20	Each requests 700 fictional units
Accepted reservations per round	14	9,800 of 10,000 units committed
Budget denials per round	6	No partial reservation created
Confirmed spending after settlement	5,000	Ten operations settle at 500
Reservations retained for unknown outcomes	2,800	Four unresolved operations at 700
Available capacity after settlement	2,200	Ceiling minus both categories

Separate assertions test a 4,000-unit proposal requiring approval. It is rejected before approval. An approval for that proposal does not permit a changed 4,100-unit proposal. The exact approved proposal is admitted; a later attempt to change its payload is rejected. An unapproved destination, a negative amount, and a settlement exceeding its reservation are also rejected. These are selected acceptance cases, not a complete security test suite. [7]

FIGURE 02

MEASURED LOCAL FIXTURE

Twenty requests cannot all spend the same balance.

In each of 20 rounds, twenty concurrent requests seek 700 fictional units from a 10,000-unit ceiling.

01

14

Admitted

Each receives a 700-unit reservation.

02

6

Budget denied

Insufficient unallocated capacity; no reservation created.

03

9,800

Peak reserved

Below the 10,000-unit ceiling in every measured round.

400 requests across 20 local rounds. SQLite serialized writers. These are fixture observations, not a guarantee for distributed deployments or real providers.

Kymata Labs example.py and recorded result.json; November 16, 2026. [7]

The example assumes a trusted caller identity and a trusted reviewer operation. It uses one wallet per database and does not implement an authentication service, expiry, revocation, multiple currencies or a real executor. SQLite serializes the local writes; the test does not establish behavior under distributed failover. Twenty successful rounds are evidence about these runs, not a proof over all possible schedules. The accounting argument and the executable evidence should remain distinguishable.

Unknown is an accounting state

There is an uncomfortable interval after a side effect might have occurred and before its outcome is known. The application can dislike that interval without making it disappear. Calling it “failed” can release money already committed. Calling it “succeeded” can mislead the user and trigger dependent work. Calling it “unknown” gives the system a fact it can act on honestly.

A production reservation lifecycle needs a deliberate transition into dispatch, a record of the external operation identity, and a reconciliation path. If a worker crashes before dispatch, the system needs evidence that the request could not have escaped before releasing the reservation. If it crashes after dispatch, an absence in a convenient local log is not necessarily proof of no effect. The release rule must follow the executor's actual delivery guarantees.

The fixture retains the full 700-unit reservation for each unknown operation. That is conservative and intentionally inconvenient. Real systems need an owner for reconciliation, an age threshold for investigation, and a way to show the user why capacity is unavailable. Otherwise a safe boundary can become a silent outage as unresolved reservations accumulate.

A timeout on the reservation is not, by itself, a cancellation of the remote operation. If the service can complete after the local timeout, automatically releasing the reservation creates the possibility of later spending above the local ceiling. A provider-enforced deadline or a confirmed cancellation can change that conclusion. A local clock cannot.

The same discipline applies to refunds and reversals. Whether they restore spending capacity is an explicit budget policy. A gross-commitment limit and a net-spending limit answer different questions. Repeated purchase-and-refund cycles should not accidentally manufacture new authority if the purpose of the ceiling is to restrict transaction exposure.

FIGURE 03 RESERVATION ACCOUNTING

Unknown is still committed.

After ten operations settle at 500 units and four remain unresolved, the original 10,000–unit ceiling has three distinct parts.

01

5,000

Confirmed spending

Ten operations with a known 500–unit result.

02

2,800

Unknown outcomes

Four full 700–unit reservations remain held.

03

2,200

Available

Only this remainder may fund new commitments.

Fictional integer units in the local fixture. A timeout does not prove cancellation; the example retains rather than releases unresolved reservations.

Kymata Labs example.py and result.json; original proposed lifecycle. [7]

The case against turning every agent into a purchasing department

The strongest objection is that this machinery can cost more than the actions it controls. A disposable local experiment, a fixed-price read operation, or a small task with a genuinely enforceable prepaid allowance may not need a bespoke ledger. Existing infrastructure may already provide the relevant account limit, approval workflow and operation history. Adding a second authoritative ledger without a clear reconciliation contract can create more ambiguity.

That objection is sound. The design should begin with the available execution boundary, not a desire to build another platform. If a downstream service supplies a hard ceiling shared by all callers, use it and verify its scope, delay and overrun behavior. If the activity has no variable external cost and negligible side effects, a simpler call limit may be proportionate. The requirement is an enforceable constraint appropriate to the consequence, not a particular database schema.

There is also a productivity cost to review. Requiring a person to approve every trivial action defeats much of the point of delegation. The answer is bounded standing authority: a narrow action set, known destinations, a sensible aggregate allowance, and an escalation path for exceptions. Human attention can then concentrate on decisions that exceed the envelope rather than repeatedly certifying routine calls.

But the opposite shortcut has its own cost. A vague approval followed by unrestricted execution transfers interpretation risk to the user. A cost dashboard discovered after the work is done is useful reporting, not prevention. Explaining an overrun cannot grant retroactive authority.

A small transaction can still exceed the mandate

A monetary ceiling cannot decide whether a transaction belongs to the task. Sending a private file may cost nothing. Publishing a damaging message may fit comfortably inside a service allowance. A ten-unit purchase from a prohibited destination can violate policy even when the wallet contains thousands. This is why the example checks the operation and destination before it checks available capacity. Passing the arithmetic test must not override a failed permission test.

The reverse problem is fragmentation. A policy that requires review above 2,000 units may be evaded, deliberately or accidentally, by splitting one consequential purchase into several smaller requests. The aggregate budget limits total exposure, but it does not establish that those fragments were independently authorized. Where the business decision is the project or order, bind review to that decision and include its subordinate commitments. Do not ask the model to invent a fresh approval boundary every time it proposes a smaller tool call.

This is a reason to preserve the user's logical task identifier alongside individual operations. It allows a reviewer to see that a sequence of locally acceptable actions has become one larger undertaking. It also clarifies cancellation: stopping future work on a task does not erase its existing commitments. The owner can revoke permission for new reservations while keeping already-dispatched operations in reconciliation. That distinction makes a stop control useful without turning it into a misleading claim that all effects have been reversed.

What to require before the first real commitment

A builder can review this design through four pieces of evidence. First, an action contract identifies the principal, destination, operation and maximum exposure. Second, an admission test shows that competing requests cannot allocate the same capacity. Third, an approval test shows that consequential changes invalidate consent when policy requires it. Fourth, a failure exercise shows that an uncertain external outcome remains visible and funded until reconciliation resolves it.

The tests should include the path users will actually take, including background workers and administrative tools. A beautifully checked primary route is insufficient if a scheduled job can submit the same action through an older credential. Keep the exception path explicit: who may raise a ceiling, what they see, and where the change is recorded. An operator should not need to disable the entire policy service to rescue one delayed order.

Finally, measure friction as well as containment. Count budget denials, time waiting for approval, unresolved reservations and genuine policy exceptions. Those measurements can expose a ceiling that is too low, a review threshold that is too broad, or a reconciliation process that is understaffed. They do not justify silently weakening a limit. They give the owner evidence for changing it deliberately.

The promise of an agent is that it can act without requiring a person to specify every step. That makes the boundary around its actions more valuable, not less. A capable assistant should be able to find the best use of the authority it has. The application must be able to say, with an accounting record rather than an encouraging sentence, exactly how much authority remains.

Questions worth asking.

Is a budget written into the system prompt enough?

It can help the model propose appropriate actions, but it does not atomically coordinate concurrent spending or prevent a separate execution path from bypassing policy. Enforce admission where the side effect is controlled, using an authoritative reservation record. The local fixture demonstrates that accounting boundary. [1][7]

Why reserve the maximum instead of counting completed charges?

Several requests can be in flight before any charge appears. Reserving a defensible maximum prevents those requests from claiming the same capacity. Settlement releases unused capacity; an unknown outcome retains its reservation. The maximum must also be enforceable or defensible at the external boundary. [7]

Does every tool call need human approval?

No. Bounded standing authority can cover routine actions within a defined destination set, per-action maximum and aggregate ceiling. High-impact or exceptional decisions can require per-call approval. Inspect the SDK and application rules because persistent tool approval is broader than approving one proposal. [1][4]

Does the example work as a production payment service?

No. It is a Python standard-library illustration using a local SQLite wallet and fictional units. Identity and reviewer trust are assumptions; real payment execution, approval expiry, revocation and distributed failover are outside its scope. Its assertions verify selected local cases. [7]

What happens when an action times out?

A timeout does not prove that an external action was canceled. Keep its reservation and reconcile the original operation using the executor's actual guarantees. Assign an owner and show why capacity remains unavailable instead of treating uncertainty as permission to spend again. [7]

SOURCES, NOT SCENERY

References.

1. **LLM06:2025 Excessive Agency**
   OWASP GenAI Security Project · 2026–11–16 · Primary security guidance

   Supports excessive functionality, permissions and autonomy; downstream complete mediation, minimum functionality and human approval. The linked third-party incidents were not independently retrieved or reused as evidence. Exact inspected path: 2_0_vulns/LLM06_ExcessiveAgency.md. Commit 99f4395589bdbd120ae961f9cd179e79d7f9b27f. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/OWASP/www-project-top-10-for-large-language-model-applications/blob/99f4395589bdbd120ae961f9cd179e79d7f9b27f/2_0_vulns/LLM06_ExcessiveAgency.md
   Accessed copy: https://raw.githubusercontent.com/OWASP/www-project-top-10-for-large-language-model-applications/99f4395589bdbd120ae961f9cd179e79d7f9b27f/2_0_vulns/LLM06_ExcessiveAgency.md

2. **MCP 2025-06-18: Authorization**
   Model Context Protocol · 2026–11–16 · Primary protocol specification

   Inspected protected-resource authorization, resource indicators, audience validation and rejection of token passthrough. This source is not claimed to implement a spending ledger. Exact inspected path: docs/specification/2025-06-18/basic/authorization.mdx. Commit ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/modelcontextprotocol/modelcontextprotocol/blob/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/specification/2025-06-18/basic/authorization.mdx
   Accessed copy: https://raw.githubusercontent.com/modelcontextprotocol/modelcontextprotocol/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/specification/2025-06-18/basic/authorization.mdx

3. **MCP 2025-06-18: Tools**
   Model Context Protocol · 2026–11–16 · Primary protocol specification

   Supports human ability to deny tool invocations and treatment of annotations from untrusted servers as untrusted. Tool descriptions do not constitute the proposed application enforcement. Exact inspected path: docs/specification/2025-06-18/server/tools.mdx. Commit ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/modelcontextprotocol/modelcontextprotocol/blob/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/specification/2025-06-18/server/tools.mdx
   Accessed copy: https://raw.githubusercontent.com/modelcontextprotocol/modelcontextprotocol/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/specification/2025-06-18/server/tools.mdx

4. **OpenAI Agents SDK: Human-in-the-loop**
   OpenAI · 2026–11–16 · Primary SDK documentation

   Inspected needs_approval, interruptions, resumable RunState, per-call versus sticky approval, and the documented cross-agent scope of sticky local shell/file-edit decisions. Version-specific snapshot, not a live product test. Exact inspected path: docs/human_in_the_loop.md. Commit 64fbde5d02b42ae5f9db668eb0184cdb3fc9d775. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/openai/openai-agents-python/blob/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/docs/human_in_the_loop.md
   Accessed copy: https://raw.githubusercontent.com/openai/openai-agents-python/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/docs/human_in_the_loop.md

5. **OpenAI Agents SDK: Guardrails**
   OpenAI · 2026–11–16 · Primary SDK documentation

   Inspected parallel versus blocking input guardrails, final output timing, and function-tool guardrail coverage. Does not claim all tool families use the same execution pipeline. Exact inspected path: docs/guardrails.md. Commit 64fbde5d02b42ae5f9db668eb0184cdb3fc9d775. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/openai/openai-agents-python/blob/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/docs/guardrails.md
   Accessed copy: https://raw.githubusercontent.com/openai/openai-agents-python/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/docs/guardrails.md

6. **MCP 2025-06-18: Security Best Practices**
   Model Context Protocol · 2026-11-16 · Primary security guidance

   Inspected OAuth proxy confused-deputy scenario and per-client consent mitigation. Used as an identity-binding illustration, not evidence of a financial exploit. Exact inspected path: docs/docs/2025-06-18/tutorials/security/security_best_practices.mdx. Commit ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b. Reference date is capture date, not original publication date. Original bytes and SHA-256 preserved in the source receipt.

   https://github.com/modelcontextprotocol/modelcontextprotocol/blob/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/docs/2025-06-18/tutorials/security/security_best_practices.mdx
   Accessed copy: https://raw.githubusercontent.com/modelcontextprotocol/modelcontextprotocol/ac00313ff7f5e9f47b0e0d1ebdf9c0ad6411380b/docs/docs/2025-06-18/tutorials/security/security_best_practices.mdx

7. **Reproducible local fixture: example.py**
   Kymata Labs Research · 2026-11-16 · Original executable illustration

   Python standard library only. Run python3 example.py to perform the assertions and write result.json. Local fictional scenario; no network, provider, model or payment calls. The publication reports the measured fixture result, not a production benchmark.

   https://www.kymatalabs.com/research/give-your-agent-a-budget-before-you-give-it-authority/assets/example.py