Veto Protocol:
Authorization Infrastructure for Agentic Systems

Veto Protocol

30 September 2026
Research draft v0.2

Delegate intelligence. Retain authority.

Document status. This paper describes proposed architecture and ongoing research. It does not establish production readiness, completed independent audit, or deployed wallet protection. Individual authors and institutional affiliations are not specified in this edition.

Abstract

AI agents are beginning to act across applications rather than simply respond to questions. This makes delegation more useful, but it also exposes a mismatch between the task a user intends and the access a connected system may receive. Permission to retrieve one document should not become unrestricted authority over an account.

Veto Protocol proposes an authorization gateway for supported agent actions. The architecture separates an agent's reasoning from the decision to permit execution. Requests are associated with authenticated identities, evaluated against resource and action constraints, and executed through controlled integrations only when the applicable conditions are satisfied.

The design focuses on scoped permissions, time limits, action-specific approvals, revocation, and reviewable decision records. Its protection is bounded by the execution path: independent provider connections remain outside Veto's control. Authorization also does not establish that an action is beneficial or that the agent's reasoning is correct.

This paper presents the proposed architecture, trust assumptions, permission lifecycle, and validation requirements in practical terms. It treats wallet authorization as a separate research extension and does not assume that the core gateway requires a token. The central objective is to keep an agent's authority tied to what the user deliberately delegates.

Keywords: AI agents; authorization; delegated access; least privilege; approval workflows; controlled execution.

Scope of claims. Design requirements in this paper are not measurements, formal proofs, or statements that all described controls have been implemented. Development demonstrations and security assurance are discussed separately in Section 9.

Contents

1 Introduction 5

1.1 The central idea 5

1.2 Three things that should remain separate 5

1.3 Where Veto fits 5

1.4 Where it does not fit 6

2 Security Model and Trust Assumptions 6

2.1 Prompt injection and excessive requests 6

2.2 Stolen credentials and independent connections 6

2.3 The gateway is a trust dependency 6

2.4 Trust in external providers 6

2.5 Authorization is not a judgment of quality 6

3 System Architecture 6

3.1 Identity and policy management 7

3.2 The request gateway 7

3.3 The authorization decision 7

3.4 The integration executor 7

3.5 Decision records 7

3.6 Relationship to existing authorization standards 7

4 Permissions and Delegation 8

4.1 Resource and action scope 8

4.2 Time limits and revocation 8

4.3 Approval for a particular action 8

4.4 When the user is away 8

4.5 Delegation to another agent 8

5 Execution Integrity and Integrations 8

5.1 Check again when circumstances change 9

5.2 Treat retries carefully 9

5.3 Fail closed for protected actions 9

5.4 External authorization services 9

5.5 Integration contracts 9

6 Privacy and Credential Handling . . . 9

6.1 Understand the data path . . . 9

6.2 Keep logs useful without copying everything . . . 10

6.3 Protect credentials as high-value assets . . . 10

6.4 Separate revocation from deletion . . . 10

6.5 Claims must follow architecture . . . 10

7 Wallet Authorization: Research Direction . . . 10

7.1 Where a restriction is enforced matters . . . 10

7.2 Potential controls . . . 10

7.3 Transactions can do more than their labels suggest . . . 10

7.4 What must be demonstrated . . . 11

8 Economics and Token Position . . . 11

8.1 A possible service model . . . 11

8.2 The position on $VETO . . . 11

8.3 What could create a defensible product . . . 11

9 Validation and Development Roadmap . . . 11

9.1 The evidence required . . . 12

9.2 Current baseline . . . 12

9.3 A staged release path . . . 12

9.4 Audit claims need a scope . . . 12

10 Conclusion . . . 12

A Conceptual Request Model . . . 13

References . . . 13

1 Introduction

AI agents are becoming useful in a new way. They can move beyond answering a question to opening a document, calling an application, or carrying out a task on someone else's behalf. The opportunity is substantial. So is the responsibility that comes with access.

Most people do not think in terms of access tokens or authorization scopes. They think in terms of tasks: summarize this file, prepare this message, help me complete this workflow. The permission a system receives, however, can be much broader and last much longer than the task the user had in mind.

Veto Protocol is being built around that mismatch. Its proposed role is to check what an agent is allowed to do before a supported action is executed. The agent can decide what to request; a separate authorization layer decides whether that request may proceed.

1.1 The central idea

An agent's ability to take an action is not permission to take it.

Veto proposes a gateway through which supported agent requests are authenticated, checked against user-defined permissions, and passed to connected applications only when authorized. The aim is practical: give agents enough access to be useful without treating every connection as an open-ended grant of authority.

Consider a straightforward request: ask an agent to summarize a project document. The user intends to share one resource for one purpose. A connection that exposes a much wider account creates a different arrangement, even if the agent has been instructed to stay within the original task.

That distinction matters because agents can make mistakes, encounter malicious content, or continue operating after the user stops watching. Clear instructions are useful, but they are not a substitute for a boundary that can reject an unauthorized operation.

1.2 Three things that should remain separate

Capability is what the software can technically attempt. **Intent** is what the user wants accomplished. **Authority** is what the software is permitted to execute. A secure delegation system needs to distinguish all three.

The problem is not that every agent is malicious. It is that the consequences of a mistake depend on the access available when that mistake happens. A narrowly scoped permission can limit the impact. An unnecessarily broad one can turn a small error into an account-wide event.

1.3 Where Veto fits

Veto is intended to sit between an agent and supported application actions. It does not replace the model, the application provider, or the user's judgment. It adds a place where explicit permissions can be enforced independently of the agent's reasoning.

1.4 Where it does not fit

Veto is not a device-wide security system. It cannot govern a separate connection that an agent already holds, remove information the agent has received, or undo every completed action. Its protection is limited to supported requests that actually pass through its enforcement path.

2 Security Model and Trust Assumptions

The primary security objective is bounded: a supported action should not execute through Veto unless the applicable policy authorizes it. That is different from promising to identify every malicious intention or protect every application.

2.1 Prompt injection and excessive requests

Malicious content can influence what an agent tries to do. Independent authorization can reject a request outside its scope, even if the agent believes the request is justified. It may not stop a harmful action that falls within a broad permission. Least privilege remains essential.

2.2 Stolen credentials and independent connections

Scoped and revocable agent credentials can limit the reach of a stolen credential, but an attacker may still act within its valid scope. Separately held provider credentials create a different problem: they may allow requests to bypass Veto altogether. Deployments seeking exclusive enforcement must address those paths.

2.3 The gateway is a trust dependency

An operator or infrastructure compromise could affect policies, credentials, or execution. Isolation, restricted administrative access, monitoring, and independent review are therefore part of the security requirements. An API gateway is not trustless simply because it serves a Web3 audience.

2.4 Trust in external providers

Veto also depends on application providers to implement their APIs and authorization mechanisms correctly. Provider outages, permission changes, and undocumented behavior can affect the system. These dependencies belong in the threat model.

2.5 Authorization is not a judgment of quality

A permitted action can still be a poor decision. Veto is intended to answer whether the action is authorized, not whether the agent's reasoning is correct or the outcome will benefit the user. Those questions require other safeguards.

3 System Architecture

The architecture separates the parts that manage authority from the parts that use it. A user defines the boundaries. An agent submits a request. Veto evaluates the request, and a controlled integration carries out the operation if authorization is valid.

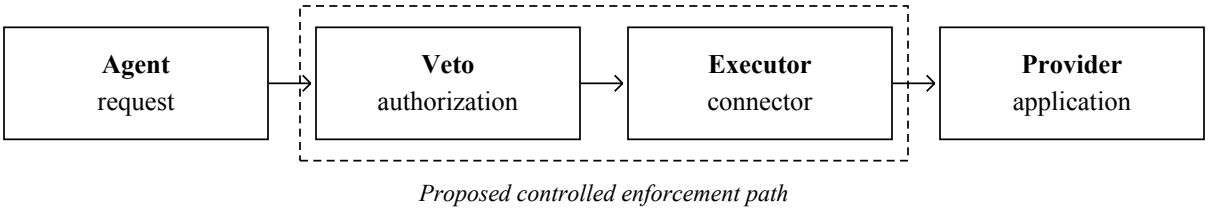


Figure 1. Proposed request path. Provider credentials are intended to remain within controlled execution. Separate agent-to-provider connections fall outside this boundary.

3.1 Identity and policy management

Each request must be associated with an authenticated agent and the correct user. A display name is not an identity credential. Agent credentials need defined scope, rotation, expiry, and revocation behavior. Changing a policy is itself a sensitive action: an agent should not be able to increase its own authority simply because it can call tools.

3.2 The request gateway

The gateway receives and validates the request. It identifies the resource, operation, and relevant parameters before evaluating the applicable rules. Ambiguous identifiers or malformed requests should not be passed through for execution.

3.3 The authorization decision

The proposed decision model has three outcomes: allow, deny, or require approval. An approval requirement means execution remains blocked until a valid user decision is recorded and the relevant authorization is checked again. Silence or user absence must not become implicit consent.

3.4 The integration executor

The executor uses a supported provider API to perform the authorized action. Where possible, provider credentials stay within this controlled environment rather than being handed to the agent. If another execution route can use the same credentials without checking policy, the boundary is incomplete.

3.5 Decision records

The system should record what was requested, what decision was made, and whether execution succeeded. These are separate events. An allowed request may still fail at the provider, and a timeout may leave the final outcome uncertain.

3.6 Relationship to existing authorization standards

Veto is not a replacement for provider authorization standards. OAuth integrations should be evaluated against the security guidance in RFC 9700 [1], which updates OAuth 2.0 security practice. A policy gateway still needs correct credential handling, provider authentication, and integration-specific enforcement. Referencing that standard does not establish compliance by the current implementation.

4 Permissions and Delegation

A useful permission answers a small set of concrete questions: which agent, acting for which user, may perform which operation on which resource, under what conditions? The more clearly those questions are answered, the less authority needs to be inferred.

4.1 Resource and action scope

Reading a selected document should not imply permission to edit, share, or delete it. Access to one document should not silently expand to the surrounding folder or account. The precision available depends on the integration and the provider's resource model.

4.2 Time limits and revocation

A task does not always require a permanent connection. Permissions should be able to expire or be withdrawn. But expiry and revocation have limits: they do not erase information already delivered, and they may not cancel work that has already started inside an external application.

Revocation needs a measured operational definition. The system must account for queued requests, cached decisions, and active executions. Promising an instant universal kill switch would hide the most important implementation details.

4.3 Approval for a particular action

An approval should identify the action the user is authorizing. If the resource, recipient, amount, or other material parameter changes, the previous approval should no longer be sufficient. Approval records need expiry and protection against reuse.

4.4 When the user is away

The intended behavior is straightforward: actions already permitted by policy may proceed; actions requiring the user's approval wait or expire. Being offline should not widen an agent's permissions.

4.5 Delegation to another agent

Transitive delegation is not assumed in the initial architecture. Allowing agents to grant access to other agents requires additional rules to ensure that the new authority never exceeds the original grant.

5 Execution Integrity and Integrations

Authorization is valuable only if the action that executes is the action that was authorized. The time between those events introduces practical problems that the implementation must address.

5.1 Check again when circumstances change

A request may wait for approval while its permission expires or the user changes a policy. Relevant authority should be revalidated before execution after a material delay. Policy versioning and short-lived execution authorizations are possible design tools, not guarantees established by this paper.

5.2 Treat retries carefully

An external application can complete an operation even when the gateway does not receive a successful response. Retrying blindly could repeat the action. Each connector needs a documented approach to idempotency, ambiguous outcomes, and recovery rather than assuming a failed response means nothing happened.

5.3 Fail closed for protected actions

If authorization cannot be established, the target behavior is to stop protected execution. This reduces availability during failures, but prevents an outage from becoming a permission bypass. Timeouts, dependency failures, stale decisions, and recovery behavior all need explicit tests.

5.4 External authorization services

Veto may use an external policy service for part of the decision process. That can help with policy management, but it does not automatically solve credential handling, connector security, approval integrity, or execution enforcement. The product must be evaluated as a complete request path.

5.5 Integration contracts

Each connector should state its supported operations, provider scopes, data handling, rate limits, disconnect behavior, and known limitations. A successful connection screen is not evidence that every operation is safely constrained.

6 Privacy and Credential Handling

A service built to control access can itself become a sensitive point of concentration. Veto must therefore be judged by what it processes and retains, not by privacy language alone.

6.1 Understand the data path

Depending on the integration, the gateway may process account identifiers, agent credentials, permission settings, request metadata, and content returned by an application. The production privacy documentation must identify which of these are stored, why they are needed, and who can access them.

6.2 Keep logs useful without copying everything

A reviewable record does not always require the complete document or message involved. The design goal is to retain enough structured information to explain a decision without unnecessarily placing sensitive content in logs. Retention periods must be explicit rather than indefinite by default.

6.3 Protect credentials as high-value assets

Provider credentials need protection in storage and transit, restricted operational access, rotation procedures, and isolation between users. Encryption at rest is useful, but it does not by itself mean that the service operator cannot access the underlying credentials.

6.4 Separate revocation from deletion

Revoking an agent, disconnecting an application, invalidating a provider token, and deleting an account are different actions. User guidance must explain what each one stops or removes, including any retention in backups or records.

6.5 Claims must follow architecture

This paper does not claim zero-knowledge processing or end-to-end confidentiality from the gateway operator. Those properties would require a specific technical design and evidence. The privacy policy must reflect the deployed implementation before a production release.

7 Wallet Authorization: Research Direction

The same delegation question appears in wallets: permission to help with a transaction should not automatically grant unrestricted financial authority. However, wallet protection is not simply another application connector. It introduces signing, contract, and execution risks that require separate analysis.

7.1 Where a restriction is enforced matters

A rule enforced only in a backend or interface cannot prevent a transaction signed through an independent route. Stronger controls may need to live in the wallet's actual authorization mechanism or smart-account execution logic.

7.2 Potential controls

Research areas include restricted destinations, spending limits, transaction-specific approvals, and narrowly scoped session permissions. No particular wallet standard, chain, or deployed implementation is established by this paper.

7.3 Transactions can do more than their labels suggest

An approved address or function name may not describe the full effect of a transaction. Contract composition, token approvals, upgradeable contracts, and indirect execution can change the risk. Simulation can inform a decision, but should not be presented as a guarantee of the eventual outcome.

7.4 What must be demonstrated

Before claiming wallet protection, Veto needs to identify every supported signing path, explain where restrictions are enforced, and test how an unauthorized transaction might bypass them. Recovery and revocation behavior must be documented as carefully as the successful path.

Wallet authorization remains a planned extension, not a live capability established here.

8 Economics and Token Position

The core app-access gateway does not technically require a blockchain or a token. Its usefulness should be demonstrated by whether it gives users and developers reliable control over agent actions.

8.1 A possible service model

Subscriptions, usage-based fees, or enterprise support could fund a maintained authorization service. Those models would need to reflect the cost of reliable execution, connector maintenance, security review, and customer support. This paper does not establish pricing or commercial availability.

8.2 The position on $VETO

Any future token should have a role that is justified separately from the product narrative. A token is not a substitute for authorization, and owning one should not grant access to another user's resources.

No allocation, staking program, governance entitlement, fee-sharing arrangement, or return expectation is defined in this document. Product adoption, service revenue, and token demand are distinct propositions.

8.3 What could create a defensible product

The opportunity is not the invention of authorization itself. It is making authorization practical for agent workflows: understandable permissions, reliable integrations, secure approvals, and evidence that restrictions hold when an agent behaves unexpectedly.

A growing connector list is useful only if the enforcement behind each connector is dependable. Trust will come from clear behavior and repeatable results, not a claim of universal coverage or first-mover status.

9 Validation and Development Roadmap

A demonstration of an agent retrieving a file proves less than a demonstration that the same agent cannot retrieve a prohibited file. Security validation must cover both successful workflows and deliberately unauthorized requests.

9.1 The evidence required

Testing should cover out-of-scope resources, unauthorized operations, expired permissions, revoked access, changed parameters after approval, replayed approvals, duplicate requests, and isolation between users. It should also cover dependency outages and ambiguous responses from external providers.

Operational reporting should distinguish authorization latency, revocation propagation, service availability, and connector execution reliability. Performance claims need a defined workload and measured results.

9.2 Current baseline

Development demonstrations have included agent setup, permission configuration, Google Drive connection, file retrieval through Veto, and passkey setup. These workflows are evidence of development progress, not independent proof of production-grade enforcement or security.

9.3 A staged release path

First: verify scoped execution for a small integration set. **Next:** test approval, expiry, and revocation behavior. **Then:** publish implementation-aligned APIs and connector documentation, followed by operational hardening and independent review appropriate to the release scope. Wallet research proceeds as a separate workstream.

9.4 Audit claims need a scope

A future review should identify the version, components, findings, and remediation status. An audit of one component does not certify the entire system. No completed independent audit is claimed in this paper.

Release decisions should follow this evidence. A control appearing in an interface is not sufficient reason to describe it as fully enforced.

10 Conclusion

AI agents can make delegation easier. They can also make it easier to give away more authority than a task requires. Solving that problem does not begin with asking a model to be careful. It begins with deciding which actions should be possible and enforcing those decisions where execution happens.

Veto Protocol proposes that boundary for supported agent workflows. The ambition is clear, but the standard of proof should be equally clear: narrowly scoped permissions, controlled execution, reliable lifecycle behavior, and transparent disclosure of what the system cannot control.

The goal is not to stop agents from acting. It is to keep their authority tied to what users deliberately choose to delegate.

Delegate intelligence. Retain authority.

A Conceptual Request Model

The notation below is a compact description of the intended decision inputs. It is not an API schema or a formally verified protocol.

$$Q = (u, a, i, r, o, p, c)$$

Symbol	Meaning
u	User on whose behalf the request is made
a	Authenticated agent identity
i	Target integration
r	Requested resource
o	Requested operation
p	Parameters that determine the action
c	Context relevant to policy evaluation

Table 1. Conceptual request fields.

Execution is intended to require an authenticated identity, matching permission scope, valid conditions, and a valid approval when the policy requires one. A deny decision blocks execution. A require-approval decision remains blocked until approval and revalidation succeed. The exact consistency model, cache behavior, and revocation timing remain implementation matters.

References

[1] T. Lodderstedt, J. Bradley, A. Labunets, and D. Fett. *Best Current Practice for OAuth 2.0 Security.* RFC 9700, BCP 240, 2025. rfc-editor.org/rfc/rfc9700. Technical background for Section 3.

[2] L. Breidenbach et al. *Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks.* Version 1.0, 15 April 2021. Chainlink Research. Document-format reference only; no Chainlink architecture, endorsement, or security properties are claimed for Veto.

Publication note. This edition uses a conventional research-paper format informed by [2]. Veto's text and system proposal remain distinct. This document is not an investment solicitation, security certification, or service-level commitment. Official project channels: vetoprotocol.live and x.com/vetoprotocol.