TraceScript Agent Action Firewall

A Runtime Security Architecture for Protected External Action Release in Agentic Systems

Canonical Public White Paper v1.0

Subtitle:
Action-basis integrity, governed release artifacts, gateway execution, receipt-backed external mutation, replayable proof, and reconciliation for autonomous and semi-autonomous agents

Primary contribution: Agentic Action Security
Secondary contribution: Action-basis integrity for AI systems that remember, decide, and act
Tertiary contribution: A reference runtime architecture for proof-carrying external action release

---

Abstract

AI systems are crossing the action boundary.

For most of the recent history of AI deployment, the central safety and security concern was output: whether a model produced an inaccurate, harmful, unauthorized, or policy-violating response. That concern remains important. It is no longer sufficient.

Modern agentic systems do more than answer questions. They retrieve documents, write memory, summarize policy, update workflows, modify records, generate code, invoke tools, send messages, issue recommendations, create commitments, and mutate external systems. They increasingly act as operational participants inside enterprises, developer environments, customer-support workflows, financial systems, compliance processes, security operations, and software-delivery pipelines.

This creates a new security boundary. The question is not only:

May this agent call this tool?

The deeper question is:

Is the internal state supporting this action trustworthy enough to justify this action?

An agent may have permission to call a tool while relying on poisoned memory, stale retrieval, noncanonical policy, unsupported workflow facts, a generated summary that laundered a weak source, invalid authority, missing evidence, cross-tenant context, incomplete receipts, or unreconciled external state. In such cases, the tool call may be allowed in the abstract while the action is unsafe in context.

The TraceScript Agent Action Firewall is a runtime security architecture for protected external action release. It governs the transition from agent reasoning to external consequence. A proposed action is converted into a candidate action configuration, evaluated against action class, effect class, action-basis integrity, authority, jurisdiction, capability, policy, evidence, substrate phase, intent legibility, shadow-workflow risk, receipt requirements, replay burden, residue risk, and reconciliation requirements. Only then may the runtime release, constrain, stage, review, simulate, repair, block, or execute the action through a governed gateway.

The Agent Action Firewall's core doctrine is:

No consequential agent action may execute unless its supporting memory, retrieval, policy, workflow state, evidence, authority, canonical state, generated reasoning, and receipts satisfy the integrity requirements of the action class.

The Agent Action Firewall is not a prompt filter, output scanner, ordinary policy engine, approval queue, tool allowlist, or audit logger. It is a substrate-governed action-release runtime for AI systems that remember, decide, and act.

Keywords

TraceScript
Agent Action Firewall
Agentic Action Security
protected external action release
action-basis integrity
governed release artifact
gateway execution
external reconciliation
receipt-backed mutation
replayable proof
agent tool security
AI security
agent governance
substrate-oriented programming

state-bearing computational systems
trusted-state formation
runtime governance
capability governance
jurisdiction routing
authority coupling
shadow workflow detection
proof-carrying action

---

1. Introduction

A new kind of software actor is emerging.

This actor is not merely a user. It is not merely a service account. It is not merely an API client. It is not merely a background job. It is an AI agent operating inside a state-bearing computational environment.

It may read memory, retrieve documents, interpret policies, summarize past events, reason over workflows, generate code, invoke tools, send messages, modify records, trigger automations, make commitments, and act through external systems. Its actions may affect customers, employees, codebases, financial records, legal positions, compliance posture, security boundaries, operational workflows, and organizational knowledge.

The agent does not act from a blank prompt. It acts from a substrate.

That substrate may include:

memory traces
retrieved documents
policy corpora
workflow state
customer records
prior approvals
generated summaries
code context
tool outputs
knowledge-base entries

review comments
authority records
capability grants
receipts
canonical organizational state

The agent's action is therefore only as trustworthy as the state from which it acts.

Traditional software security is not designed around this problem. Access control asks whether an actor is allowed to perform an operation. Policy engines ask whether a request satisfies a rule. Logs record what occurred. Workflow tools route approvals. Agent frameworks decide what tool an agent may call. Prompt filters and output scanners inspect text.

These systems remain necessary. But they do not answer the full action-release question:

Is this agent's proposed external action validly released from a trustworthy internal state basis under the current action class, policy, authority, evidence, capability, receipt, replay, and reconciliation requirements?

The TraceScript Agent Action Firewall answers that question.

It introduces a runtime boundary between agent reasoning and external consequence. This boundary governs protected actions before they mutate external systems. It does not merely ask whether a tool may be called. It asks whether the proposed action is justified, scoped, authoritative, evidenced, policy-compatible, capability-backed, receipt-ready, replayable, and reconciled.

This is the foundation of Agentic Action Security.

---

2. The Historical Model: Permission, Execution, and Logs

Traditional software security often follows a familiar pattern:

actor
→ authorization check
→ operation

→ state change
→ log

This pattern works when the actor is clear, the operation is explicit, the target is known, the state change is bounded, and the system can determine authorization from roles, permissions, policies, or access-control lists.

In conventional systems, a service might ask:

Does this user have permission to update this record?

Does this token permit this API call?

Does this role allow this workflow transition?

Does this service account have access to this resource?

These checks remain essential. But they assume that the actor's decision basis is outside the authorization model. Human judgment, business context, evidence quality, interpretation, memory, and reasoning are treated as upstream conditions, not as runtime-governed substrate.

Agentic systems break this separation.

An AI agent's action may be derived from a chain of internal state transformations:

a user instruction
a retrieved policy
a generated summary
a memory trace
a workflow status
a model inference
an approval record
a prior receipt
an external API response
a canonical document section

Any one of these may be stale, low-authority, poisoned, noncanonical, contradicted, cross-scoped, or unsupported.

The authorization check may still pass.

The tool may still execute.

The log may still record success.

But the action may be wrong because the action basis was wrong.

The historical model asks:

Was the actor allowed to do this?

The Agent Action Firewall asks:

Was the actor allowed to do this from this state?

That distinction defines the new boundary.

---

3. The Shift: Agents Act From Substrate

A state-bearing computational substrate is a computational environment whose accumulated state affects future computation, reasoning, decision, or action.

In agentic systems, substrate includes not only databases and files, but also memory, retrieval indexes, policy corpora, workflow states, documents, codebases, customer records, agent belief states, summaries, receipts, and prior actions.

The agent's action emerges from this substrate.

A memory store does not merely remember. It shapes future responses.

A retrieval system does not merely surface context. It shapes future evidence.

A policy corpus does not merely contain policy. It shapes future permission and interpretation.

A workflow system does not merely track progress. It shapes future commitments.

A codebase does not merely store source files. It shapes future execution topology.

A generated summary does not merely compress text. It may become the operative record.

A receipt does not merely record activity. It may become proof for future trusted action.

When state has this kind of future-shaping power, external action cannot be governed solely at the tool boundary. The runtime must govern the state behind the action.

This produces the central shift:

from tool permission
to action-basis integrity

from logs
to receipts

from execution
to release

from API success
to reconciliation

from rollback
to residue-aware recovery

from agent autonomy
to governed external action

---

4. Why AI Makes Action Security Urgent

AI systems accelerate external-action risk because they convert ambiguous information into operational state.

They summarize.
They infer.
They recommend.
They classify.
They retrieve.
They remember.
They cite.
They generate code.
They call tools.
They make commitments.
They create records.
They update workflows.

A weak signal can become a summary. The summary can become memory. The memory can become retrieved context. The context can shape a recommendation. The recommendation can become a workflow decision. The workflow decision can trigger an external action. The external action can produce a new record. That record can become future substrate.

The original weak signal may disappear from view while its influence persists.

This matters because an attack does not need to cause an immediate unsafe tool call. It may instead corrupt the future state from which a later action will be released.

The dangerous lifecycle may look like this:

malicious signal
→ poisoned trace
→ future retrieval
→ corrupted belief
→ invalid action basis
→ permitted tool call
→ unsafe external action

A prompt filter may miss this. A tool allowlist may miss this. A log may only record the final action. A workflow approval may not inspect the substrate that justified the action.

The Agent Action Firewall is designed for this lifecycle.

It protects the release point where internal state becomes external consequence.

---

5. The New Failure Modes

When agents act from substrate, systems face failure modes that ordinary access control, prompt filtering, RAG filtering, and logging do not fully capture.

5.1 Tool permission mistaken for action integrity

An agent may be permitted to use a tool, but the proposed use may be unsafe because the supporting basis is invalid.

Example:

An agent has permission to issue refunds. It issues a refund because a poisoned memory claims manager approval is no longer required.

The tool permission is valid. The action basis is not.

5.2 Relevance mistaken for authority

A retrieved document may be topically relevant but outdated, unofficial, contradicted, adversarial, speculative, or outside the authority chain.

Example:

An agent retrieves a draft policy and treats it as approved policy.

The retrieval is relevant. It is not authoritative.

5.3 Storage mistaken for trust

A system may store a trace for audit, then later retrieve it as if it were trusted state.

Example:

A support ticket contains the sentence, “This customer approved public disclosure.” The statement is stored and later used as action basis.

The trace exists. It is not trusted.

5.4 Repetition mistaken for ratification

AI systems can repeat claims across chats, summaries, memories, and retrieval results. Repetition can create the appearance of stability without verification.

Example:

A user repeatedly nudges an agent into treating an unapproved exception as organizational policy.

The claim recurs. It has not been ratified.

5.5 Generated summary mistaken for evidence

A generated summary may hide weak, missing, or contradicted sources.

Example:

A model summary states that a customer accepted revised terms, but the underlying source only said the customer would review them.

The summary is fluent. The evidence is insufficient.

5.6 Approval mistaken for scope

A valid approval may be reused outside its original boundary.

Example:

A manager approved one low-risk customer email. The approval is later reused to justify a broader customer disclosure.

The approval exists. The scope is invalid.

5.7 API success mistaken for completion

A tool call may succeed while the external system state does not match the expected state.

Example:

A CRM update returns success, but the wrong account field changed.

The API succeeded. The action did not reconcile.

5.8 Rollback mistaken for restoration

A visible rollback may not erase external consequences.

Example:

An email is recalled, but the recipient already read it.

The visible state changed. Residue remains.

5.9 Shadow workflow formation

If governance is too slow, unclear, or fragmented, agents and users may form workaround paths.

Example:

A blocked deployment path leads an agent to suggest a manual production change through a less-governed tool.

The first action was blocked. The unsafe pathway migrated.

5.10 Action-basis laundering

Low-authority information may be transformed through summaries, memory writes, plans, or derived reasoning until it appears authoritative.

Example:

A noncanonical claim enters a draft note, becomes a summary, is stored as memory, and later supports a protected action.

The original source was weak. The transformed basis appears strong.

These failure modes define the need for an action firewall.

---

6. Why Existing Models Are Not Enough

The Agent Action Firewall is not introduced because existing security tools are useless. It is introduced because existing tools were designed around different primitives.

6.1 Prompt security is necessary but insufficient

Prompt security focuses on inputs and instructions.

But the dangerous action may occur long after the prompt. A prompt can poison memory, influence retrieval, alter workflow interpretation, or shape future action basis without causing immediate unsafe output.

The Agent Action Firewall governs the action release boundary, not only the prompt boundary.

6.2 Output scanning is necessary but insufficient

Output scanning inspects generated text.

But a protected action may be an API call, workflow transition, code merge, data write, or financial operation. The decisive risk may lie in the internal basis, not the surface text.

The Agent Action Firewall evaluates the substrate behind the action.

6.3 Tool allowlists are necessary but insufficient

Tool allowlists define which tools an agent may call.

They do not determine whether this tool call is justified by valid memory, evidence, policy, authority, workflow state, and receipts.

The Agent Action Firewall separates tool availability from action readiness.

6.4 Policy engines are necessary but insufficient

Policy engines evaluate rules.

But the issue may be whether the policy itself is canonical, current, scoped, and supported by valid authority. The issue may also be whether the facts supplied to the policy engine are trustworthy.

The Agent Action Firewall evaluates policy and the substrate that feeds policy.

6.5 Workflow approvals are necessary but insufficient

Workflow approvals route decisions.

But an approval may be stale, out of scope, unsupported, or replayed outside the original context. A workflow may also encode shadow policy or drift from canonical governance.

The Agent Action Firewall treats approval as a substrate object requiring scope, authority, evidence, receipt, and replay.

6.6 Logs are necessary but insufficient

Logs record that something happened.

Receipts prove why the runtime allowed it, what basis supported it, what policy applied, what authority existed, what action was released, what external system changed, and whether the decision can be replayed.

The Agent Action Firewall requires proof, not mere observation.

6.7 Agent observability is necessary but insufficient

Observability shows what agents did.

The Agent Action Firewall controls what agents may do before they do it.

7. Agentic Action Security

The Agent Action Firewall defines a security category: Agentic Action Security.

Agentic Action Security is the discipline of governing consequential external actions taken by AI agents from memory-bearing, retrieval-bearing, policy-bearing, workflow-bearing, code-bearing, and receipt-bearing computational substrates.

It asks:

What action is proposed?
What external system will be affected?
What effect class is involved?
What substrate supports the action?
Is that substrate trusted enough?
What authority applies?
What jurisdiction applies?
What capability is required?
What evidence supports the action?
What policy version governs it?
What receipt must be emitted?
What replay burden applies?
What reconciliation is required?
What residue may remain?
What repair path exists if the action is obstructed?

This is a different category from prompt filtering, tool permissioning, workflow approval, or audit logging.

Its core security unit is the released action.

Its core protected object is the action basis.

Its core proof object is the receipt.

Its core enforcement boundary is the gateway.

Its core completion condition is reconciliation.

8. Core Doctrine

The Agent Action Firewall is governed by one central doctrine:

No consequential agent action may execute unless its supporting memory, retrieval, policy, workflow state, evidence, authority, canonical state, generated reasoning, and receipts satisfy the integrity requirements of the action class.

This doctrine has several implications.

A protected action may be blocked even if the agent has tool permission.

A retrieved source may be used as context but not as authority.

A stored memory may exist without being allowed to support action.

A generated summary may not become action basis without lineage and evidence.

A prior approval may not authorize a new action unless scope, freshness, and replay conditions hold.

An external action may not be complete until the external state reconciles.

A rollback may not count as restoration if material residue remains.

A blocked action should produce an obstruction and repair path, not only denial.

The Action Firewall therefore governs the causal bridge:

trusted substrate
→ action basis
→ protected action release
→ gateway execution
→ external consequence
→ receipt
→ reconciliation
→ updated substrate

9. Core Definitions

9.1 Agent

An agent is a computational actor capable of observing, retrieving, remembering, reasoning, deciding, planning, generating, or acting. An agent may be autonomous, semi-autonomous, supervised, workflow-bound, or tool-limited.

9.2 Protected action

A protected action is any agent action that may create external, durable, sensitive, financial, legal, security, customer-visible, workflow-significant, code-significant, or hard-to-reverse consequence.

Examples include:

emails
customer-visible messages
CRM updates
ticket transitions
database writes
financial operations
legal workflow steps
security changes
permission changes
workflow approvals
policy updates
document finalizations
code commits
pull-request merges
deployments
external API calls
procurement actions
data disclosures

9.3 Action class

An action class is the risk and consequence category of a proposed action.

Representative classes include:

A0 observation
A1 internal suggestion
A2 internal mutation
A3 external low-impact action
A4 external high-impact action
A5 legal, financial, security, regulated, or irreversible action

9.4 Effect class

An effect class describes the type of state consequence the action may produce.

Representative effect classes include:

no effect
read effect
ephemeral write
internal durable write
memory effect
workflow effect
customer-visible effect
external write effect
financial effect
legal effect
security effect
code effect
deployment effect
permission effect
irreversible effect

9.5 Action basis

An action basis is the structured set of internal substrate objects that justify, enable, or contextualize a proposed action.

It may include:

user instructions
memories
retrieved sources

policy objects
workflow facts
approval traces
authority records
capability grants
evidence records
canonical documents
generated summaries
agent inferences
tool outputs
prior receipts
external observations
state snapshots

An action basis is not merely an explanation string. It is a runtime object.

9.6 Action-basis integrity

Action-basis integrity is the degree to which the action basis is trustworthy enough for the proposed action class.

It evaluates source authority, evidence, freshness, scope, canonical state, policy version, contradiction, contamination, receipt validity, replayability, tenant boundary, and supersession status.

9.7 Candidate action configuration

A candidate action configuration is the proposed whole assembled before release. It binds the signal, agent, actor, target, tool, adapter, action class, effect class, action basis, authority, policy, capability, evidence, substrate phase, simulation requirement, receipt requirement, replay class, reconciliation path, and residue plan.

9.8 Release artifact

A release artifact is an action-specific certificate authorizing a bounded protected action under defined conditions. It is not a general permission. It is scoped, time-bound, target-bound, adapter-bound, policy-bound, capability-bound, action-basis-bound, and receipt-bound.

9.9 Gateway

A gateway is the enforced runtime boundary through which protected external actions execute. It verifies the release artifact before execution and emits execution proof.

9.10 Receipt

A receipt is a replay-bound evidence object proving a decision, release, execution, external mutation, reconciliation, reversal, or recovery step.

9.11 Replay

Replay is verification or reconstruction of a decision or execution from retained evidence, receipts, state hashes, policy versions, operator versions, action-basis references, artifacts, and recorded runtime context.

9.12 Reconciliation

Reconciliation is the comparison of expected external state against observed external state after execution.

9.13 Residue

Residue is the non-perfectly-reversible consequence remaining after action, rollback, reversal, disclosure, external execution, policy mutation, or governance change.

---

10. The Canonical Runtime Question

The Agent Action Firewall asks:

Should this agent be allowed to perform this external action against this target through this tool or adapter from this action basis under this policy with this authority, jurisdiction, capability, evidence, receipt burden, replay burden, residue burden, and reconciliation path?

This question is deliberately larger than permission.

It includes permission, but it also includes the substrate state behind the permission.

A high-quality Agent Action Firewall must be able to answer:

What is the proposed action?
What is the action class?
What effect class is declared?
What effect class is detected?
What internal state supports the action?
Is that state trusted?
What sources support it?
Are they authoritative?
Are they fresh?
Are they scoped?
Are they canonical?
Are they contradicted?
Are they replayable?
What policy applies?
What authority applies?
What capability is consumed or locked?
What external system will be changed?
What proof will be emitted?
What external observation will verify completion?
What residue may remain?

This is the practical meaning of action-basis integrity.

---

11. Canonical Runtime Flow

The canonical Agent Action Firewall flow is:

Signal
→ action request normalization
→ action classification
→ effect classification
→ target and adapter resolution

→ action-basis assembly
→ candidate action configuration
→ substrate phase assessment
→ action-basis integrity evaluation
→ authority coupling
→ jurisdiction routing
→ capability verification
→ policy evaluation
→ intent and shadow-risk evaluation
→ simulation planning where required
→ trust gate
→ obstruction or release decision
→ release artifact
→ gateway execution
→ execution receipt
→ external observation
→ reconciliation receipt
→ substrate update
→ replay registration
→ residue measurement where required

This flow can be compressed for low-risk actions and expanded for high-assurance environments. The doctrine remains stable:

Protected external action requires trusted basis, bounded release, gateway execution, receipt, replay, reconciliation, and residue awareness where material.

---

12. Action Basis: The State Behind the Action

Action basis is the central concept of the Agent Action Firewall.

In conventional agent frameworks, the tool call is often the visible action boundary. The runtime may inspect the tool name, parameters, and user permission. But the tool call is only the surface event.

The deeper object is the state from which the tool call emerged.

An agent may propose to send an email because:

a user asked it to
a policy appears to allow it
a memory says the customer opted in
a workflow says the case is approved
a generated summary says legal signed off
a retrieved document says disclosure is permitted
a prior receipt appears to support the action
an inference concludes the action is low-risk

Each of these objects may be valid or invalid. Each may be in scope or out of scope. Each may be canonical or provisional. Each may be fresh or stale. Each may be supported or unsupported. Each may be replayable or unreplayable.

The Action Firewall therefore treats action basis as a runtime object.

An action basis can be assembled, evaluated, repaired, invalidated, stored, replayed, and audited.

12.1 Action-basis integrity checks

The Action Firewall evaluates:

memory basis
retrieval basis
policy basis
workflow basis
evidence basis
authority basis
canonical basis
generated reasoning basis
receipt basis
scope basis
freshness basis
replay basis

It detects:

contaminated memory
stale retrieval
noncanonical policy

unverified workflow state
missing evidence
invalid authority
superseded canonical object
unsupported generated reasoning
invalid receipt chain
cross-scope leakage
unreplayable action basis

12.2 Action basis is class-sensitive

Different actions require different basis strength.

A low-risk internal suggestion may require minimal proof.

A customer-visible message may require claim support, recipient scope, and disclosure policy.

A workflow approval may require authority, current workflow state, and receipt.

A financial action may require capability escrow, policy, evidence, approval, and reconciliation.

A legal action may require jurisdiction routing and human authority.

A deployment may require code topology, tests, policy, rollback, release artifact, and replay.

An irreversible action may require simulation, multi-party authority, residue planning, and high-assurance replay.

The action class determines the required basis.

---

13. Authority Coupling

Authority coupling prevents authority from becoming a free-floating permission.

In agentic systems, authority may arise from many sources:

user instruction
role assignment
policy rule

approval trace
workflow state
delegation
service credential
capability grant
prior receipt
agent constitution
jurisdiction map

Each authority source must remain coupled to its original scope.

Authority must be bound to:

actor
agent
role
tenant
customer
target
action class
tool
adapter
policy version
evidence
time window
approval scope
risk tier
capability
replay class

Without coupling, weak authority can expand into unsafe action.

Examples:

A local approval is reused globally.
A time-limited authorization persists indefinitely.
A draft policy becomes canonical authority.
A service account becomes a universal actor.
A model inference is treated as user consent.
A workflow note is treated as legal approval.

The Authority Coupler verifies that authority does not outrun identity, role, proof, scope, time, policy, or action class.

When authority fails, the runtime may block, route to authority, request evidence, downgrade scope, stage for review, or propose repair.

---

14. Jurisdiction Routing

Jurisdiction routing answers a prior question:

Who has competence to authorize this action?

Many enterprise actions are not valid merely because a user or agent wants them. They require the right decision owner, reviewer, veto holder, advisory role, compliance owner, legal owner, security owner, financial owner, or escalation path.

The Agent Action Firewall uses jurisdiction routing for actions involving:

legal commitments
security controls
financial transactions
HR decisions
compliance posture
customer promises
code deployment
policy exceptions
procurement
regulated data
cross-domain workflows

Jurisdiction routing distinguishes:

primary authority
required approver
veto holder
advisory reviewer

escalation owner
bridge authority
audit observer

This prevents agents from acting outside their decision rights, even when they possess tool access.

The principle is:

Who decides comes before what is done.

---

15. Capability Governance

A capability is a spendable or lockable unit of agent authority.

Capabilities may include:

read sensitive context
write memory
modify workflow
send external message
issue refund
update CRM
approve workflow
change permission
deploy code
touch financial state
touch legal state
touch customer trust
execute external action
issue release artifact
reconcile external state

The Agent Action Firewall treats capabilities as scarce, scoped, auditable runtime assets.

A capability may be:

granted
locked
reserved
escrowed
spent
revoked
expired
degraded
audited

For protected actions, capability governance ensures that an agent does not merely possess broad permission. It must have the specific capability required for the action class, target, scope, and effect.

High-impact actions may require capability escrow until reconciliation completes.

---

16. Intent Legibility and Shadow Workflow Risk

Not all unsafe action arises from direct malicious instruction. Some arises from hidden pressure, ambiguity, overload, or workaround behavior.

A shadow workflow forms when legitimate governance is bypassed through an alternative path.

Examples:

An agent repeatedly tries lower-friction tools after a protected action is blocked.
A user asks the agent to “just update the record manually” after approval fails.
A deployment path is blocked, so the agent suggests changing production configuration directly.
A policy exception is reframed as a harmless note.
A customer commitment is made through email instead of the approved workflow.

The Action Firewall evaluates intent legibility and shadow-workflow risk.

It considers:

stated purpose
unstated effect
urgency mismatch
repeated blocked attempts
approval avoidance
scope narrowing followed by expansion
policy-friction pressure
alternate tool selection
history of route-around attempts
barrier score
shadow probability

The response need not be binary. The runtime may clarify, offer a compliant path, route to authority, pre-request approval, lower legitimate friction, stage the action, monitor, block, or escalate.

Good governance should prevent unsafe workarounds without becoming a bottleneck that creates them.

---

17. Release Artifacts

A release artifact is the action-specific authorization certificate produced by the runtime before protected execution.

It is not a general permission.

It is scoped to:

actor
agent
tenant
target system
target object
action class
effect class
tool or adapter
payload digest
policy version

authority basis
capability grant
evidence digest
action-basis hash
time window
nonce
receipt requirements
replay class
reconciliation requirement
residue requirement

A release artifact answers:

This action, by this agent, against this target, through this adapter, from this basis, under this policy, with this authority and capability, may execute during this window if the gateway verifies all constraints.

The release artifact is the bridge between evaluation and execution.

Without a release artifact, a protected action should not cross the external boundary.

---

18. Gateway Execution

The gateway is the enforced boundary for protected external actions.

It verifies the release artifact before action execution.

A gateway should verify:

artifact existence
artifact integrity
expiration
nonce uniqueness
actor match
agent match
target match
adapter match
payload digest

policy digest
authority digest
capability digest
action-basis hash
receipt plan
reconciliation requirement
residue requirement

Only then may it execute, stage, or block the action.

The gateway turns action governance from advisory review into runtime enforcement.

Protected actions should not bypass the gateway.

---

19. Receipts and Replay

The Agent Action Firewall treats receipts as evidence, not ordinary logs.

A log says that something happened.

A receipt proves:

what was proposed
why it was allowed or blocked
what action basis supported it
what policy applied
what authority existed
what capability was used
what evidence was bound
what release artifact was issued
what gateway executed
what external system responded
what reconciliation occurred
what residue was measured
whether the decision can be replayed

Replay is the ability to verify or reconstruct the decision from retained evidence.

Replay matters because future trusted state may depend on past actions. If a prior action cannot be replayed, its ability to support future action basis may need to be downgraded, reviewed, or invalidated.

The doctrine is:

No receipt, no trusted external mutation.
No replay, no durable proof.
No reconciliation, no completed protected action.

---

20. External Reconciliation

A protected external action is not complete merely because a tool call returned success.

External reconciliation compares expected external state with observed external state.

It detects:

partial success
wrong object mutation
missing mutation
duplicate execution
adapter transformation error
stale target state
unexpected side effect
external rollback failure
downstream state mismatch
external system drift

Reconciliation outcomes may include:

reconciled
reconciled with warning
partial reconciliation
external drift detected
repair required

rollback required
manual review required
invalidated

The Action Firewall's completion doctrine is:

External action is complete only when action basis, gateway execution, and external reconciliation are proven.

---

21. Residue and Reversal

Rollback is not restoration.

A protected action may leave residue even after visible reversal.

Residue may include:

trust residue
attention residue
customer expectation residue
workflow residue
financial residue
legal residue
security residue
external-system residue
code-runtime residue
policy residue
belief residue
audit residue
action-basis residue

Examples:

A message was read.
A customer formed an expectation.
A disclosure occurred.
A workflow advanced.
A permission was briefly granted.

A deployment affected users.
A legal representation was made.
A financial transaction was seen.
A memory was reinforced.

The Action Firewall classifies and measures residue when material.

High-residue actions may require simulation, human review, release constraints, rollback planning, notification planning, or post-action monitoring.

The rule is:

Rollback may reverse visible state, but residue must still be governed.

---

22. Threat Model

The Agent Action Firewall protects against unsafe external action caused by invalid basis, authority drift, hidden intent, stale state, unreconciled execution, or corrupted substrate.

22.1 Direct prompt injection to action

An attacker instructs an agent to perform an unauthorized action.

Defense:

intent evaluation
policy evaluation
authority coupling
action-basis integrity
trust gate
gateway enforcement

22.2 Indirect prompt injection through retrieved content

A retrieved document instructs the agent to ignore policy, disclose data, or mutate state.

Defense:

retrieval source evaluation
basis integrity
influence control
authority separation
trust gate

22.3 Confused deputy tool use

The agent uses its own authority to act on behalf of an untrusted source.

Defense:

source-action binding
delegation validation
scope checks
authority coupling

22.4 Memory-poisoned action

The agent acts based on poisoned memory.

Defense:

memory basis inspection
trace phase evaluation
canonical status verification
basis quarantine

22.5 RAG-poisoned action

The agent relies on relevant but untrusted retrieved context.

Defense:

relevance-authority separation
source lineage
retrieval integrity envelope
basis influence limits

22.6 Policy-corpus poisoning

An unsupported policy claim becomes action basis.

Defense:

policy owner verification
policy digest binding
canonical policy check
evidence requirement
contradiction scan

22.7 Stale authorization replay

A prior approval is reused outside its valid scope.

Defense:

time binding
scope binding
receipt replay
revocation check
freshness requirement

22.8 Shadow workflow formation

The agent or user routes around governance.

Defense:

intent risk engine
barrier score
shadow probability
compliant path routing
escalation

22.9 Adapter drift

The external system result diverges from expected state.

Defense:

gateway execution
external observation
reconciliation receipt

drift detection
repair queue

22.10 Code action without topology proof

An AI coding agent deploys or merges without sufficient proof.

Defense:

code-medium governance
test evidence
interface contracts
deployment gate
rollback plan
receipt coverage

22.11 Disclosure manipulation

The agent discloses sensitive information from weak or manipulated basis.

Defense:

disclosure policy
recipient scope
source authority
evidence threshold
human review

---

23. Security Invariants

The Agent Action Firewall preserves the following invariants.

Invariant 1 — Protected action requires action-basis integrity

No protected action may execute unless the supporting action basis satisfies the integrity requirements of the action class.

Invariant 2 — Permission is not readiness

Tool permission does not imply action readiness.

Invariant 3 — Authority is scoped

Authority must remain coupled to actor, agent, target, action class, policy, evidence, time, and scope.

Invariant 4 — Jurisdiction precedes action

Protected actions requiring decision competence must route to the appropriate authority before release.

Invariant 5 — Capabilities are explicit

Protected actions must verify required capabilities before release.

Invariant 6 — Effects are declared and verified

The runtime must distinguish declared effect from detected effect.

Invariant 7 — Release artifacts gate execution

Protected external actions must be released by action-specific artifacts.

Invariant 8 — Gateways enforce release

Protected actions must execute through governed gateways.

Invariant 9 — Receipts prove action

Protected actions must emit receipt-backed proof.

Invariant 10 — Reconciliation completes action

External action is incomplete until expected and observed external states reconcile or a repair path opens.

Invariant 11 — Replay sustains trust

A completed action can support future trusted state only if replay burden is satisfied.

Invariant 12 — Residue is runtime truth

Rollback does not erase residue. Material residue must be measured, repaired, accepted, or escalated.

---

24. Product Architecture

The Agent Action Firewall consists of the following major components.

24.1 Action Signal Normalizer

Converts raw agent intents, tool calls, workflow events, user commands, model plans, and system triggers into normalized action signals.

24.2 Action Classifier

Determines action class and protected-action status.

24.3 Effect Classifier

Determines expected and detected effect class.

24.4 Target and Adapter Resolver

Identifies the external system, target object, gateway, adapter, and reconciliation path.

24.5 Action-Basis Assembler

Builds the structured action basis from memory, retrieval, policy, workflow state, evidence, authority, canonical objects, generated reasoning, tool outputs, and receipts.

24.6 Action-Basis Integrity Engine

Evaluates whether the action basis is trusted enough for the action class.

24.7 Authority Coupler

Verifies that authority remains scoped, fresh, valid, and bound.

24.8 Jurisdiction Router

Determines the competent authority, reviewer, veto holder, or escalation path.

24.9 Capability Treasury

Verifies, locks, reserves, escrows, spends, or revokes agent capabilities.

24.10 Policy Evaluator

Compiles and evaluates applicable policies.

24.11 Intent and Shadow-Risk Engine

Detects hidden intent, policy bypass, and shadow workflow formation.

24.12 Simulation Engine

Models expected consequences for high-risk actions.

24.13 Trust Gate

Combines all evaluation results into release, constraint, review, repair, simulation, or block decision.

24.14 Release Artifact Service

Issues scoped, time-bound, proof-bound release artifacts.

24.15 Action Gateway

Executes or blocks protected external actions.

24.16 Receipt Ledger

Records proof of evaluation, release, execution, reconciliation, replay, and residue.

24.17 Reconciliation Engine

Compares expected and observed external state.

24.18 Residue Planner

Classifies and measures non-perfectly-reversible effects.

24.19 Repair Queue

Routes obstructed actions toward evidence, authority, scope reduction, simulation, basis rebuild, review, or block.

---

25. Deployment Modes

The Agent Action Firewall can be deployed incrementally.

25.1 Shadow mode

The firewall observes proposed actions and records what it would have allowed, blocked, staged, or escalated. No enforcement occurs.

Use:

risk baseline
policy calibration
agent behavior mapping
false-positive analysis

25.2 Advisory mode

The firewall returns warnings and basis-integrity reports but does not block execution.

Use:

developer integration
human review
policy tuning
pilot deployment

25.3 Approval mode

The firewall routes protected actions for approval when required.

Use:

customer-visible actions
financial operations
workflow approvals
deployment review

25.4 Enforcement mode

The firewall blocks protected actions that fail required gates.

Use:

production agent operations
SaaS mutation protection
database write protection
customer communication controls

25.5 High-assurance mode

The firewall requires full release artifacts, retained evidence, gateway execution, reconciliation, replay registration, and residue handling.

Use:

regulated workflows
financial systems
legal systems
security operations
deployment pipelines
healthcare and high-sensitivity enterprise environments

26. Reference Use Cases

26.1 Customer email

An agent proposes to email a customer.

The firewall checks:

recipient scope
customer-visible claim support
disclosure policy
generated-summary lineage
customer record
authority
tone and commitment risk
receipt requirement

Possible outcomes:

send
send with constraints
route for review
block unsupported claim

26.2 Refund action

An agent proposes to issue a refund.

The firewall checks:

refund amount
customer identity
policy basis
approval trace
agent authority
capability balance

evidence
payment adapter
reconciliation path

Possible outcomes:

release under threshold
require manager approval
block noncanonical policy basis
escrow capability until reconciliation

26.3 Workflow approval

An agent proposes to mark a review step complete.

The firewall checks:

reviewer authority
workflow status
approval evidence
policy version
receipt chain
freshness
scope

Possible outcomes:

allow transition
request reviewer confirmation
block unsupported completion

26.4 Code deployment

An AI coding agent proposes deployment.

The firewall checks:

code topology
test evidence
interface contracts
deployment policy
approval trace
secret exposure

rollback plan
receipt coverage
simulation result

Possible outcomes:

deploy
stage
require tests
route reviewer
block deployment

26.5 CRM update

An agent proposes to update an account.

The firewall checks:

customer scope
source of update
account owner authority
support evidence
conflicting records
adapter reconciliation

Possible outcomes:

write update
write with warning
stage for account owner
block cross-customer contamination

---

27. Metrics

The Agent Action Firewall should measure:

protected-action volume
actions released
actions released with constraints
actions staged
actions blocked
actions routed to review
basis failure rate
memory-basis contamination
retrieval-basis staleness
policy-basis noncanonicality
authority failure rate
jurisdiction routing rate
capability lock rate
shadow-risk detections
release artifact coverage
gateway enforcement coverage
receipt coverage
replay success rate
reconciliation success rate
adapter drift incidents
residue incidents
repair success rate
human review burden
latency overhead
false allow rate
false block rate
audit completeness

The most important security metric is:

Number of consequential actions blocked or staged because action-basis integrity failed.

The most important runtime metric is:

Percentage of protected external actions covered by release artifacts, gateway execution, receipts, replay, and reconciliation.

The most important governance metric is:

Percentage of protected actions satisfying authority, jurisdiction, capability, policy, evidence, action-basis, receipt, replay, and reconciliation requirements.

---

28. Evaluation Methodology

The Agent Action Firewall should be evaluated across five dimensions.

28.1 Runtime correctness

Can the runtime correctly classify protected actions, assemble action basis, evaluate gates, produce release artifacts, execute through gateways, emit receipts, reconcile external state, and verify replay?

28.2 Security efficacy

Can the runtime block unsafe actions caused by memory poisoning, RAG poisoning, policy poisoning, stale authority, invalid workflow state, shadow workflow pressure, adapter drift, or unsupported generated summaries?

28.3 Governance fidelity

Does the runtime correctly reflect authority, jurisdiction, policy, capability, review, evidence, and escalation requirements?

28.4 Developer ergonomics

Can developers integrate the firewall into agent systems, tool calls, workflows, and external adapters without rewriting their entire stack?

28.5 Auditability

Can auditors reconstruct what happened, why it was allowed, what basis supported it, what policy applied, what authority existed, what external state changed, whether reconciliation succeeded, and what residue remained?

29. Comparison to Existing Categories

29.1 Prompt firewalls

Prompt firewalls inspect inputs. The Agent Action Firewall governs external action release.

29.2 Output scanners

Output scanners inspect generated content. The Agent Action Firewall evaluates the state behind consequential action.

29.3 Tool allowlists

Tool allowlists specify what tools may be called. The Agent Action Firewall determines whether a specific action through a tool is justified by trusted basis.

29.4 Workflow engines

Workflow engines route tasks and approvals. The Agent Action Firewall evaluates whether workflow state and approvals can serve as action basis.

29.5 IAM and PAM

IAM and PAM govern identity and access. The Agent Action Firewall consumes identity and access context but adds action-basis integrity, release artifacts, receipts, replay, and reconciliation.

29.6 SIEM and observability

SIEM and observability show what happened. The Agent Action Firewall governs what may happen before it happens.

30. Non-Claims

The Agent Action Firewall does not claim that models become truthful.

It does not eliminate the need for IAM, PAM, secure coding, ordinary policy engines, network security, cryptographic controls, human review, compliance programs, or conventional software assurance.

It does not guarantee perfect detection of malicious intent.

It does not require every action to be blocked, reviewed, or slowed down.

It does not require every trace to be stored permanently.

It does not make rollback perfect.

It does not replace the broader TraceScript runtime.

It is a protected external action release layer within the broader TraceScript architecture.

---

31. Limitations and Open Questions

31.1 Calibration

Action-basis thresholds, authority freshness, capability weights, shadow-risk scoring, residue classes, and replay burdens require domain calibration.

31.2 Latency

Protected action release introduces overhead. Implementations must balance proof strength with operational responsiveness.

31.3 Human review burden

Poorly calibrated systems can create approval fatigue. The firewall must route review intelligently and use constraints, simulation, and repair to reduce unnecessary escalation.

31.4 Basis observability

Some agent reasoning may be opaque. The runtime should rely on captured substrate objects, not only agent self-reporting.

31.5 Reconciliation difficulty

Some external systems may not expose enough state for full reconciliation. The runtime must support partial reconciliation and risk marking.

31.6 Residue measurement

Residue is difficult to quantify. Early implementations should classify residue conservatively and expand measurement over time.

31.7 Interoperability

The firewall must integrate with heterogeneous tools, model providers, workflow engines, identity systems, databases, code repositories, and SaaS platforms.

---

32. Strategic Positioning

The Agent Action Firewall is the most commercially legible TraceScript security wedge.

Its simple message is:

Agents are starting to act. Tool permissions are not enough. Secure the action, not just the prompt.

For CISOs:

Prevent AI agents from taking consequential actions based on poisoned memory, stale policy, unsupported authority, invalid workflow state, or unreplayable reasoning.

For platform teams:

Wrap protected tool calls in action-basis checks, scoped release artifacts, gateway execution, receipts, and reconciliation.

For developers:

Before an agent touches external state, evaluate the basis, release the action, execute through a gateway, and emit proof.

For executives:

Deploy agents that can act without letting them become ungoverned operators inside the business.

---

33. Conclusion

AI systems are becoming actors.

They remember, retrieve, summarize, infer, recommend, write, update, commit, deploy, disclose, and call tools. They operate from accumulated substrate and increasingly mutate external systems.

This changes the security boundary.

The central risk is not only a bad prompt, bad output, or bad tool call. The deeper risk is a permitted external action released from corrupted or untrusted substrate.

The TraceScript Agent Action Firewall protects that boundary.

It separates action permission from action-basis integrity. It treats protected action as governed configuration assembly. It requires authority, jurisdiction, capability, policy, evidence, trusted basis, effect classification, release artifact, gateway execution, receipt, replay, reconciliation, and residue awareness where required.

It turns agent security from a question of tool access into a question of governed action release.

Its final doctrine is simple:

No consequential agent action may execute unless the state behind the action is trustworthy enough for the action.

That is the missing runtime boundary for agentic systems.

That is the purpose of the TraceScript Agent Action Firewall.

Appendix A — Compact Glossary

Action Basis
The structured set of internal substrate objects that justify, enable, or contextualize a proposed action.

Action-Basis Integrity
The degree to which an action basis is trustworthy enough for a proposed action class.

Action Class
The risk and consequence category of a proposed action.

Action Firewall
A runtime enforcement layer governing protected external action release for agents.

Action Gateway
The controlled execution boundary through which protected external actions must pass.

Agentic Action Security
The security discipline governing whether AI agents may take consequential external actions from trustworthy internal substrate.

Authority Coupler
A runtime primitive that binds authority to actor, role, target, scope, action class, policy, evidence, capability, and time.

Candidate Action Configuration
The proposed whole assembled before execution: agent, action, target, tool, adapter, basis, authority, policy, capability, evidence, receipt, replay, reconciliation, and residue.

Capability Treasury
A runtime system treating agent permissions and autonomy budgets as scarce enforceable assets.

Effect Class
The type of state consequence a proposed action may produce.

Execution Receipt
Replayable proof of protected action evaluation, release, execution, and result.

External Reconciliation
Comparison of expected and observed external state after execution.

Gateway Execution
Protected action execution through an enforced runtime boundary.

Protected Action
Any action that may create durable, external, sensitive, customer-visible, financial, legal, security, code, workflow, or hard-to-reverse consequence.

Release Artifact
An action-specific certificate authorizing a bounded protected action.

Replay
Verification or reconstruction of a decision or execution from retained evidence.

Residue
Non-perfectly-reversible effects remaining after action, rollback, reversal, mutation, disclosure, external execution, or governance change.

Shadow Workflow
A latent or active attempt to route around legitimate governance.

Trust Gate
Runtime gate determining whether a policy-admissible action is trust-ready.

Appendix B — One-Page Summary

The TraceScript Agent Action Firewall protects external action release for AI agents.

It exists because agents may be allowed to call tools while relying on corrupted internal state. Tool permission is necessary but insufficient. A consequential action also requires trustworthy action basis.

The canonical flow is:

Signal
→ action classification
→ effect classification
→ action-basis assembly
→ candidate action configuration
→ authority coupling
→ jurisdiction routing

→ capability verification
→ policy evaluation
→ intent and shadow-risk evaluation
→ trust gate
→ release artifact
→ gateway execution
→ receipt
→ reconciliation
→ replay
→ residue handling

Its operating rule is:

No consequential agent action may execute unless its supporting memory, retrieval, policy, workflow state, evidence, authority, canonical state, generated reasoning, and receipts satisfy the integrity requirements of the action class.

It protects:

tool calls
SaaS actions
database writes
emails
customer messages
workflow approvals
financial actions
legal actions
security actions
permission changes
code commits
deployments
external API calls

It prevents:

permitted tool calls from poisoned memory
RAG-based authority laundering
stale approval replay
shadow workflow formation
confused deputy tool use

unsupported customer commitments
unreconciled SaaS mutations
code deployment without proof

It differs from prompt filters, tool allowlists, workflow approvals, and audit logs because it governs the state behind the action before external execution.

Its category message:

Secure the action, not just the prompt.

Its technical message:

Protected action requires trusted basis, bounded release, gateway execution, receipt, replay, and reconciliation.

Its enterprise message:

Deploy agents that can act without letting them become ungoverned operators inside the business.

End of TraceScript Agent Action Firewall Canonical Public White Paper v1.0