OSSA

NIST CAISI RFI Response: Open Standard for Software Agents (OSSA)

Formal response to NIST Request for Information on Cyber-Informed AI Safety and Security (CAISI). Docket NIST-2025-0035. Submitted by BlueFly.io on behalf of the OSSA open-source community.

Response to the NIST Center for AI Standards and Innovation (CAISI)

Request for Information on AI Agent Interoperability and Security

Federal Register Docket: NIST-2025-0035 | regulations.gov **Submission Deadline:** March 9, 2026 **Submitted by:** BlueFly.io Platform Team, on behalf of the Open Standard for Software Agents (OSSA) open-source community **Contact:** info@blueflyagents.com | https://openstandardagents.org **Date:** March 8, 2026

Table of Contents

Cover Letter

Dear NIST Center for AI Standards and Innovation,

We appreciate the opportunity to respond to this Request for Information on AI Agent Interoperability and Security. The rapid proliferation of autonomous AI agents across enterprise, government, and open-source ecosystems has created an urgent need for standardized approaches to agent identity, authorization, discovery, and governance.

BlueFly.io has spent the past six months building the **Open Standard for Software Agents (OSSA)** — an open-source, MIT-licensed specification designed to solve exactly the problems this RFI identifies. OSSA is not a whitepaper or a proposal. It is a **production-deployed specification** with:

- **A published JSON Schema** validated against thousands of agent manifests
- **A federated discovery protocol** (DUADP) with live nodes serving agent metadata
- **16 Cedar policy files (229 authorization policies)** enforcing agent security in production today
- **A compliance API** serving real-time policy evaluation at compliance.blueflyagents.com
- **An agent marketplace** with verified manifests at marketplace.blueflyagents.com
- **A discovery explorer** demonstrating federated agent lookup at discover.duadp.org

We submit this response not as a theoretical framework, but as evidence that the problems NIST seeks to address can be solved today with open standards, formal verification, and composable architecture. We welcome collaboration and would be honored to contribute OSSA as a reference implementation for the CAISI initiative.

Respectfully, The BlueFly.io Platform Team

Executive Summary

The **Open Standard for Software Agents (OSSA) v0.4** is the first agent specification to natively combine five critical security capabilities in a single, composable, machine-readable schema:

1. **Persistent Identity** — W3C Decentralized Identifiers (DIDs) for globally unique, cryptographically verifiable agent identity that survives restarts, migrations, and redeployments
2. **Formal Authorization** — Amazon Cedar, a formally verified policy language, for pre-execution authorization of every tool call and agent action
3. **Cryptographic Attestation** — Signed manifest verification with tiered trust levels, from community-submitted to organizationally verified
4. **Federated Discovery** — The Decentralized Universal Agent Discovery Protocol (DUADP) for peer-to-peer agent lookup across organizational boundaries
5. **Resource Governance** — Machine-readable token budgets, rate limits, and execution constraints enforced at the specification level

OSSA aligns with the **NIST AI Risk Management Framework (AI RMF) 1.0** across all four functions (GOVERN, MAP, MEASURE, MANAGE) and maps directly to **NIST SP 800-53 Rev. 5** control families for federal deployment readiness.

Why This Matters for Federal AI

The federal government is deploying AI agents at scale — for document processing, cybersecurity monitoring, infrastructure management, and citizen services. Without standards for identity, authorization, and governance, each deployment becomes a bespoke security problem. OSSA provides the contract layer that makes AI agents auditable, interoperable, and controllable by default.

Key differentiators from existing approaches:

Capability	OSSA v0.4	OpenAI Agents SDK	Google A2A	LangChain
Persistent DID Identity	Native (W3C DID Core)	None	Partial (Agent Card)	None

Capability	OSSA v0.4	OpenAI Agents SDK	Google A2A	LangChain
Formal Authorization	Cedar (verified)	None	None	None
Signed Manifests	x-signature tiers	None	None	None
Federated Discovery	DUADP protocol	Centralized	Centralized	None
Resource Governance	token-efficiency ext	None	None	None
NIST 800-53 Mapping	AC, AU, SC, IA	None	None	None
Open Source (MIT)	Yes	Partial	Yes	Yes
Production Deployed	Yes	Yes	Preview	Yes

About the Respondent

BlueFly.io is a software engineering organization focused on AI agent infrastructure, open standards, and developer tooling. We maintain:

- **OSSA Specification** — The Open Standard for Software Agents, MIT-licensed, with schema validation, CLI tooling, and platform exporters (openstandardagents.org)
- **DUADP Protocol** — The Decentralized Universal Agent Discovery Protocol for federated agent lookup (duadp.org)
- **Cedar Policy Engine** — 3,200+ lines of Cedar authorization policies for agent security (gitlab.com/blueflyio/cedar-policies)
- **Agent Platform** — Production infrastructure including agent marketplace, compliance engine, fleet management, and observability stack

Our work is grounded in established standards: W3C DID Core 1.0, W3C Verifiable Credentials, OpenAPI 3.1, AsyncAPI 3.0, JSON Schema 2020-12, and Amazon Cedar. We

have contributed to open-source AI agent ecosystems including Drupal AI, MCP (Model Context Protocol), and the broader OSSA community.

Section 1: Agent Identity

Response to Question 1a — Persistent Global Identifier

Question (as construed): How should agent identity be established and maintained across deployments and restarts?

The Problem

Today's AI agents are ephemeral. When an agent restarts, migrates to a new container, or is redeployed across cloud providers, its identity is lost. This makes it impossible to maintain audit trails, enforce revocation, or establish trust relationships that persist across the agent lifecycle. In federal contexts, where accountability and traceability are paramount, ephemeral identity is unacceptable.

OSSA's Solution: Global Agent Identifier (GAID)

OSSA implements a mandatory **Global Agent Identifier (GAID)** using W3C DID syntax. The GAID is the agent's permanent identity — it persists across container restarts, framework migrations, cloud provider changes, and version updates because it is bound to the agent's cryptographic identity, not to ephemeral runtime state.

Syntax and Resolution:

```
 did:ossa:550e8400-e29b-41d4-a716-446655440000
did:web:openstandardagents.org:agents:content-agent
did:web:agency.gov:agents:document-processor-v3
```

Every OSSA manifest contains an `identity` extension with a full W3C DID Document conformant to W3C DID Core 1.0. The DID Document includes:

- `verificationMethod` — Public keys for signature verification
- `authentication` — Methods for authenticating the agent
- `serviceEndpoint` — Discovery and communication endpoints
- `assertionMethod` — Keys for issuing verifiable credentials

Example from a production OSSA manifest:

```yaml
# Excerpt from an OSSA v0.4 agent manifest
apiVersion: ossa/v0.4
kind: Agent
metadata:
  name: document-processor
  namespace: gov.agency.documents
identity:
  did: "did:web:agency.gov:agents:document-processor"
  didDocument:
    id: "did:web:agency.gov:agents:document-processor"
    verificationMethod:
      - id: "did:web:agency.gov:agents:document-processor#key-1"
        type: Ed25519VerificationKey2020
        controller: "did:web:agency.gov:agents:document-processor"
        publicKeyMultibase: "z6MkhaXg..."
    authentication:
      - "did:web:agency.gov:agents:document-processor#key-1"
    service:
      - id: "did:web:agency.gov:agents:document-processor#mcp"
        type: MCPServer
        serviceEndpoint: "https://agents.agency.gov/mcp/document-processor"
      - id: "did:web:agency.gov:agents:document-processor#duadp"
        type: DUADPNode
        serviceEndpoint: "https://discover.agency.gov/agents/document-processor"
```

Persistence guarantees:

1. The GAID is immutable for the lifetime of an agent identity. The agent may be redeployed, updated, or migrated — the GAID does not change.
2. Registries (including DUADP-compliant nodes) store and serve the manifest; the identity is anchored in the registry, not in runtime state.
3. Key rotation updates the DID Document's `verificationMethod` without changing the DID itself, following W3C DID Core best practices.

Why This Matters for Federal Deployment

Federal agencies need to answer: "Which agent performed this action, and can we trace it back to an accountable entity?" The GAID provides this. A `did:web:agency.gov:agents:document-processor` can be resolved by any party to obtain

the agent's public keys, service endpoints, and trust status — enabling cross-agency verification without centralized identity providers.

Recommendation to NIST: We recommend NIST mandate that federal AI agents have a persistent, globally resolvable identifier conformant to W3C DID Core 1.0 that survives restarts and redeployments. The identifier should be bound to cryptographic material, not to runtime state.

Response to Question 1d — Revocation and Identity Lifecycle

Question (as construed): How should agent identity be revoked or invalidated?

The Challenge of Revocation at Scale

When an agent is compromised, decommissioned, or found to be operating outside its authorized scope, its identity must be invalidated immediately and that invalidation must propagate to all parties that trust the agent. In federated environments where agents operate across organizational boundaries, revocation lag creates a window of vulnerability.

OSSA + DUADP Revocation Architecture

OSSA and its companion discovery protocol DUADP support immediate, auditable, multi-layer revocation:

Layer 1 — Registry-Level Revocation:

An administrator calls `PUT /registry/revoke/{agentId}` on any DUADP-compliant registry. The agent is immediately marked as revoked. All discovery endpoints exclude revoked agents by default.

```
PUT /registry/revoke/did:web:agency.gov:agents:compromised-agent
{
  "reason": "Security incident SI-2026-0042",
  "revokedBy": "did:web:agency.gov:admins:security-team",
  "timestamp": "2026-03-08T15:30:00Z",
  "propagate": true
}
```

Layer 2 — Event-Driven Propagation:

DUADP emits real-time revocation events via AsyncAPI 3.0 channels. All subscribing systems receive notification within seconds — no polling required.

```
# DUADP AsyncAPI event definition
channels:
  agent.revoked:
    description: Real-time agent revocation notification
    messages:
      agentRevoked:
        payload:
          agentId: "did:web:agency.gov:agents:compromised-agent"
          revokedBy: "did:web:agency.gov:admins:security-team"
          reason: "Security incident SI-2026-0042"
          timestamp: "2026-03-08T15:30:00Z"
          scope: "full"  # or "partial" for capability-specific revocation
```

Layer 3 — DID Document Update:

The agent's DID Document is updated to include revocation status. DID resolvers return this status automatically, so any party verifying the agent's identity will see the revocation.

Layer 4 — Cedar Policy Enforcement:

Even if revocation propagation is delayed, Cedar policies provide a defense-in-depth check. The compliance engine evaluates the agent's trust status before every action:

```
// From our production Cedar policies: deny all actions by revoked agents
@id("deny-revoked-agents")
forbid(
  principal is BlueflyCompliance::Agent,
  action,
  resource
) when {
  principal has is_revoked &&
  principal.is_revoked == true
};
```

Audit trail: Every revocation event includes `agentId`, `timestamp`, `revokedBy` (DID of the revoking principal), and `reason`. These are persisted in the audit log and can be exported to SIEM systems.

Recommendation to NIST: We recommend NIST require that agent revocation be immediate (sub-minute propagation), auditable (with reason and revoking principal), and multi-layer (registry, event stream, DID document, and runtime enforcement). Revocation must not depend on a single point of failure.

Section 2: Authorization and Access Control

Response to Question 2a — Authorization Model

Question (as construed): What authorization model should apply to agent actions?

Why Existing Approaches Fail

Most AI agent frameworks have no authorization model at all. The agent calls whatever tools it wants, with whatever parameters it wants, limited only by the LLM's instruction-following capability. This is equivalent to giving every employee root access and trusting them to only do appropriate things. In federal contexts, this is a non-starter.

Some frameworks implement basic role-based access control (RBAC), but RBAC cannot express the nuanced, context-dependent policies that federal AI governance requires. "Agent X can use tool Y" is insufficient; federal deployments need: "Agent X can use tool Y on resources in namespace Z, only during business hours, only if the originating user has clearance level SECRET, and only if the action has been pre-approved by a human supervisor."

OSSA's Solution: Cedar Pre-Authorization

OSSA uses **Amazon Cedar** — a formally verified policy language — for authorization. Cedar was designed by Amazon for AWS Verified Permissions and has been mathematically proven to always terminate, to be decidable, and to produce consistent results. These properties are essential for security-critical authorization decisions.

Key principle: Pre-authorization, not post-audit. Every tool invocation in OSSA is authorized *before* execution. The compliance engine evaluates the principal (agent DID), action,

resource, and context against Cedar policies. If the policy decision is `deny`, the tool call never executes. There is no "run first, audit later" for sensitive operations.

Principal hierarchy in OSSA:

```
Human User (OAuth/session)
 └─ Orchestrator Agent (tier_3_write_elevated)
      ├─ Worker Agent A (tier_2_write_limited)
      │    └─ Tool calls (pre-authorized by Cedar)
      └─ Worker Agent B (tier_1_read)
           └─ Tool calls (pre-authorized by Cedar)
```

The originating human principal's DID is carried through the entire chain and logged at every step. When Agent B calls a tool, the audit log records both Agent B's DID and the human user who initiated the chain.

Real Cedar policies from our production deployment:

```
// MCP Tool Security: Analyzers cannot call destructive tools
@id("mcp-tool-analyzer-deny-high")
forbid(
  principal is BlueflyCompliance::Agent,
  action == BlueflyCompliance::Action::"callTool",
  resource is BlueflyCompliance::MCPTool
) when {
  principal.role == "analyzer" &&
  resource.sensitivity == "high"
};

// Resource limits: Enforce OSSA max_tool_calls guardrail
@id("enforce-agent-max-tool-calls")
forbid(
  principal is BlueflyCompliance::Agent,
  action == BlueflyCompliance::Action::"execute",
  resource
) when {
  principal.max_tool_calls.is_some() &&
  context has current_tool_calls &&
  context.current_tool_calls >= principal.max_tool_calls.unwrap()
};

// Deployment safety: Require human approval for production deployments
@id("forbid-unapproved-prod-deploy")
```

```
  forbid(
    principal in Role::"Agent",
    action in [Action::"deploy", Action::"modify"],
    resource in Environment::"production"
  ) unless {
    context has approvalRequired &&
    context.approvalRequired == true &&
    context has approvedBy
  };
```

Cedar Entity Schema (production):

Our Cedar schema defines OSSA-native entity types that map directly to the OSSA manifest structure:

```
namespace BlueflyCompliance {
  entity Agent {
    agent_id: String,
    agent_type: String,  // "analyzer", "executor", "reviewer", "orchestrator"
    role: String,
    capabilities: Set<String>,
    autonomy_level: String,  // "supervised", "autonomous", "fully_autonomous"
    approval_required: Bool,
    allowed_actions: Set<String>,
    blocked_actions: Set<String>,
    max_tool_calls: Long?,
    max_execution_time_seconds: Long?,
    max_concurrent_requests: Long?,
    ossa_manifest: String?,
    ossa_version: String?,
  };

  entity MCPTool {
    tool_name: String,
    sensitivity: String,  // "low", "medium", "high", "critical"
    rate_limit: Long,
    requires_auth: Bool,
  };
}
```

Scale of our Cedar deployment: 16 policy files, 2,892 lines of Cedar code, 229 individual policies served via our compliance API. These are not demonstration policies — they enforce real security constraints in our production platform.

Recommendation to NIST: We recommend NIST adopt a formally verified policy language (such as Cedar) for high-assurance agent authorization in federal contexts. We further recommend that NIST mandate pre-authorization (not post-audit) for all tool execution by federal AI agents, with the authorization decision recorded in the audit log.

Response to Question 2e — Interoperability of Authorization

Question (as construed): How can authorization policies interoperate across organizations or systems?

Cross-Organizational Trust

When Agency A's agent needs to interact with Agency B's resources, both agencies need to evaluate authorization. OSSA enables this through three mechanisms:

1. Machine-Readable Capabilities in the Manifest

Every OSSA manifest declares the agent's security tier and capabilities. Any system that can read the manifest can map the agent into its local authorization framework:

```yaml
security:
  tier: tier_2_write_limited
  capabilities:
    - read_documents
    - write_summaries
    - call_search_api
  constraints:
    max_tokens_per_call: 4096
    allowed_namespaces:
      - "gov.agency-a.documents"
      - "gov.shared.search"
```

2. DID as Cross-System Principal

The same GAID/DID is used across all systems. Agency A's Cedar policies can reference Agency B's agent by its DID:

```
// Agency B allows Agency A's document processor to read shared resources
permit(
  principal == Agent::"did:web:agency-a.gov:agents:doc-processor",
  action == Action::"read",
  resource
) when {
  resource in Namespace::"gov.shared.documents"
};
```

3. Verifiable Credential Delegation

When Agent A delegates to Agent B, Agent A issues a signed delegation credential (W3C Verifiable Credential). Agent B presents this credential along with its DID. The receiving system verifies both the credential signature and the delegation scope, preventing privilege escalation.

Recommendation to NIST: We recommend NIST standardize a minimal agent credential format based on W3C Verifiable Credentials that carries the agent's GAID, authorized scope, expiry, and the delegating principal's signature. This enables cross-organizational authorization without custom integration.

Section 3: Tool Disclosure and Security

Response to Question 3a — Tool Disclosure and Attestation

Question (as construed): How should agent tools be disclosed and attested?

Declarative Tool Disclosure

OSSA requires that **every tool an agent may call is declared in its manifest before deployment**. There is no implicit or runtime-only tool set. This is the security equivalent of a bill of materials for software — but for agent capabilities.

```yaml
tools:
  - name: search_documents
    description: "Search federal document repository by keyword"
    protocol: mcp
    input_schema:
      type: object
      properties:
        query: { type: string, maxLength: 500 }
        classification: { type: string, enum: [UNCLASSIFIED, CUI, SECRET] }
      required: [query]
    output_schema:
      type: object
      properties:
        results: { type: array, items: { $ref: "#/components/schemas/Document" } }
    rate_limit: 30/minute
    sensitivity: medium
    required_capability: read_documents
  - name: generate_summary
    description: "Generate a summary of a document"
    protocol: mcp
    input_schema:
      type: object
      properties:
        document_id: { type: string }
        max_length: { type: integer, maximum: 2000 }
      required: [document_id]
    rate_limit: 10/minute
    sensitivity: low
    required_capability: write_summaries
```

Attestation via Signed Manifests:

OSSA supports manifest signing through the `x-signature` block. Consumers that require verified trust only accept agents whose manifest signature can be cryptographically verified:

```yaml
x-signature:
  algorithm: Ed25519
  publicKey: "z6MkhaXg..."
  signature: "z3FXQjeZ..."
```

```
  signedAt: "2026-03-08T12:00:00Z"
  signedBy: "did:web:agency.gov:admins:security-team"
```

Trust tiers (see Exhibit C) define the verification level:

Tier	Name	Verification	Use Case
0	Community	No verification	Open-source experimentation
1	Self-Signed	Schema validation passes	Internal development
2	Verified	Manifest signature verifies against known key	Cross-team deployment
3	Organizational	Signed by organization's root key + audit trail	Federal production

Recommendation to NIST: We recommend NIST require that all AI agent tools be declared in a machine-readable manifest before deployment, with the manifest signed by an accountable entity. Tool-calling protocols (e.g., MCP) should require mutual authentication in federal contexts.

Response to Question 3b — Tool Security and Rate Limiting

Question (as construed): How should tool invocation be secured and constrained?

OSSA addresses tool security through three enforced layers:

Layer 1: Cedar Pre-Authorization (see Section 2a)

Every tool call is evaluated against Cedar policies before execution. Unauthorized calls never reach the tool server.

Layer 2: Rate Limits in Manifest

Each tool declares a `rate_limit` that is enforced by the platform — not advisory. Our production Cedar policies include rate limit enforcement:

```
// Prevent agents from exceeding tool rate limits
@id("mcp-rate-limit-exceed")
forbid(
  principal,
  action == BlueflyCompliance::Action::"callTool",
  resource is BlueflyCompliance::MCPTool
) when {
  context has rate_limit_consumed &&
  context.rate_limit_consumed >= resource.rate_limit
};
```

Layer 3: Input Schema Validation

Tool parameters are defined with JSON Schema. Invalid, oversized, or out-of-scope inputs are rejected at the gateway before reaching the tool implementation.

Layer 4: Resource Budgets

The OSSA `token-efficiency` extension enforces hard limits on resource consumption per session:

```
extensions:
  token-efficiency:
    budgets:
      max_tokens_per_call: 4096
      max_calls_per_session: 100
      max_total_session_tokens: 50000
    optimization:
      cache_enabled: true
      compression: true
      redundancy_elimination: true
```

Our production Cedar policies enforce these budgets:

```
// Enforce OSSA max execution time
@id("enforce-agent-max-execution-time")
```

```
  forbid(
    principal is BlueflyCompliance::Agent,
    action == BlueflyCompliance::Action::"execute",
    resource
  ) when {
    principal.max_execution_time_seconds.is_some() &&
    context has elapsed_seconds &&
    context.elapsed_seconds >= principal.max_execution_time_seconds.unwrap()
  };

  // Enforce maximum concurrent agents (safety guardrail)
  @id("enforce-max-agents")
  forbid(
    principal,
    action == BlueflyCompliance::Action::"execute",
    resource
  ) when {
    context has agent_count &&
    context.agent_count >= 10
  };
```

Recommendation to NIST: We recommend NIST define rate-limit enforcement as mandatory (not advisory) and require mutual TLS or equivalent authentication for tool-calling protocols in federal AI deployments. Resource budgets should be declared in the agent specification, not left to runtime configuration.

Section 4: Governance, Observability, and Threat Mitigation

Response to Question 4a — Resource and Token Governance

Question (as construed): How should agent resource consumption be governed?

The Cost and Safety Imperative

Ungoverned AI agents can consume unbounded resources — making thousands of API calls, generating millions of tokens, or running for hours without checkpoints. In federal contexts, this creates both cost overruns and safety risks: an agent consuming excessive resources may be malfunctioning, compromised, or executing an unintended action loop.

OSSA Token-Efficiency Extension

OSSA's `token-efficiency` extension enforces resource budgets at the specification level — not as runtime configuration that can be overridden, but as part of the agent's immutable manifest:

- **Budgets:** `max_tokens_per_call`, `max_calls_per_session`, `max_total_session_tokens` cap consumption at multiple granularities
- **Optimization:** Declared techniques (caching, compression, redundancy elimination) allow deployers to align with cost and performance goals
- **Attribution:** Every agent invocation is logged with `agent_gaid`, `principal_did`, and `token_count`, enabling cost attribution to the originating human principal

Cedar enforcement in production:

```
// Require human approval when spawning more than 5 concurrent agents
@id("require-approval-five-agents")
forbid(
  principal,
  action == BlueflyCompliance::Action::"execute",
  resource
) when {
  context has agent_count &&
  context.agent_count > 5 &&
  context has approval_for_agents &&
  context.approval_for_agents == false
};
```

This policy ensures that resource-intensive operations (spawning many agents) require explicit human approval — implementing the "human in the loop" principle at the policy level.

Recommendation to NIST: We recommend NIST mandate resource budgets in agent specifications for federal deployments. Every invocation must be attributable to a human principal for audit and cost accounting. Budget enforcement should be in the authorization layer, not advisory.

Response to Question 4b — Observability and Audit

Question (as construed): What observability and audit requirements should apply to agent operations?

Structured Audit Logging

OSSA implementations produce structured audit logs for every agent execution. The minimum audit record includes:

Field	Description	Example
`timestamp`	ISO 8601 execution time	`2026-03-08T15:30:42.123Z`
`agent_gaid`	Agent's persistent DID	`did:web:agency.gov:agents:doc-processor`
`tool_name`	Tool that was invoked	`search_documents`
`principal_did`	Originating human principal	`did:web:agency.gov:users:jsmith`
`cedar_decision`	Authorization decision	`permit` or `deny`
`policy_id`	Cedar policy that made the decision	`mcp-tool-analyzer-deny-high`
`duration_ms`	Execution time in milliseconds	`1,247`
`token_count`	Tokens consumed	`3,891`
`exit_status`	Success or failure	`success`

Full chain traceability: The audit log captures the complete chain from human user through orchestrator agent through worker agents to tool calls. Every link in the chain is recorded with DIDs and Cedar policy decisions. This enables incident investigators to reconstruct exactly what happened, who authorized it, and which policies applied.

Real-time monitoring: Our production deployment streams audit events to TimescaleDB for time-series analysis and Grafana for dashboarding. Anomaly detection (e.g., unusual token consumption patterns, repeated authorization denials) triggers automated alerts.

Incident response integration: When failures or security events are detected, automated GitLab issues are created with full context — agent ID, execution trace, policy decisions, and resource consumption. Our production system achieves sub-minute mean-time-to-detect (MTTD) for regression-class failures.

Recommendation to NIST: We recommend NIST require structured audit logs for all agent tool invocations in federal deployments. The audit record must include the authorizing policy decision (not just the action), the full principal chain, and resource consumption. Audit data should be exportable to SIEM systems in standardized formats.

Response to Question 4d — Threat Mitigation

Question (as construed): What threats should agent security frameworks mitigate and how?

OSSA Threat Matrix

The following threat matrix maps common AI agent threats to OSSA controls and NIST SP 800-53 Rev. 5 control families:

Threat	Attack Vector	OSSA Mitigation	Cedar Policy	NIST 800-53
Identity Spoofing	Agent impersonates another agent	GAID + DID resolution + x-signature verification	`deny-unverified-agents`	IA-5, SC-8
Privilege Escalation	Agent acquires capabilities beyond its tier	Cedar pre-authorization + tier/capability bounds	`mcp-tool-analyzer-deny-high`	AC-6, AC-3
Unbounded Resource Use	Agent consumes excessive tokens/API calls	token-efficiency budgets + rate limits	`enforce-agent-max-tool-calls`	SC-6
Tool Abuse	Agent calls unauthorized or dangerous tools	Declarative tool disclosure + pre-authorization + MCP auth	`forbid-force-push`	AC-3, CM-7

Threat	Attack Vector	OSSA Mitigation	Cedar Policy	NIST 800-53
Revocation Lag	Compromised agent continues operating after revocation	DUADP real-time events + Cedar runtime check	`deny-revoked-agents`	IA-5(2)
Delegation Abuse	Agent delegates with excessive scope	Signed delegation credentials + scope checks	`validate-delegation-scope`	AC-2, AC-6(1)
Manifest Tampering	Attacker modifies agent manifest	x-signature cryptographic verification	`deny-unsigned-manifests`	SI-7, SC-8
Lateral Movement	Compromised agent accesses other namespaces	Namespace isolation + Cedar resource policies	`enforce-namespace-boundary`	AC-4, SC-7
Audit Evasion	Agent operates without logging	Mandatory audit logging at authorization layer	(architectural)	AU-2, AU-3
Data Exfiltration	Agent sends data to unauthorized endpoints	Tool output schema validation + egress policies	`restrict-external-endpoints`	SC-7, AC-4

Separation of Duties — A Real-World Example:

One of our most important Cedar policies enforces separation of duties for policy authoring itself — ensuring that the security controls cannot be subverted by a single actor:

```
// Separation of duties: policy authors cannot approve their own policy changes
forbid(
  principal is User,
  action == Action::"mr_approve",
  resource is MergeRequest
) when {
  resource.source_branch like "policy/*" &&
  principal == resource.author
};

// Only authorized policy approvers can merge policy changes
```

```
forbid(
  principal is User,
  action == Action::"mr_merge",
  resource is MergeRequest
) when {
  resource.target_branch == "main" &&
  resource.source_branch like "policy/*"
} unless {
  principal in Group::"policy_approvers"
};
```

This is not theoretical — this policy is enforced in our GitLab CI/CD pipeline today, protecting the integrity of the very Cedar policies that govern agent authorization.

Recommendation to NIST: We recommend NIST align federal agent security requirements with a threat model that includes identity spoofing, privilege escalation, resource abuse, revocation lag, and audit evasion. Controls should be machine-readable (e.g., OSCAL) to enable automated compliance verification.

NIST Framework Alignment

NIST AI Risk Management Framework (AI RMF) 1.0

OSSA maps to all four functions of the NIST AI RMF 1.0:

AI RMF Function	OSSA Component	Implementation
GOVERN	Cedar policies + governance extension	229 production policies enforcing agent behavior, separation of duties, deployment rules
MAP	DUADP registry + discovery protocol	Federated agent discovery across organizations, capability mapping, risk categorization
MEASURE	token-efficiency extension + observability	Resource consumption tracking, structured audit logs, anomaly detection, quality scoring
MANAGE	Fleet Manager + revocation + lifecycle	Agent lifecycle management, real-time revocation, incident response automation

NIST SP 800-53 Rev. 5 Control Mapping

OSSA controls map to the following NIST SP 800-53 Rev. 5 control families:

Control Family	OSSA Implementation
AC (Access Control)	Cedar pre-authorization, tier-based capabilities, namespace isolation
AU (Audit)	Structured audit logging with principal chain, Cedar decision recording
IA (Identification & Auth)	W3C DID identity, x-signature verification, trust tiers
SC (System & Comms Protection)	MCP mutual authentication, rate limiting, resource budgets
CM (Configuration Management)	Declarative manifests as immutable configuration, tool disclosure
SI (System & Information Integrity)	Manifest signing, schema validation, input validation
CA (Assessment & Authorization)	Automated compliance engine, continuous policy evaluation

OSCAL Compatibility

OSSA control implementations can be expressed in NIST's Open Security Controls Assessment Language (OSCAL) format, enabling automated compliance assessment. We maintain an OSCAL component definition that maps OSSA controls to SP 800-53 requirements.

Production Evidence and Live Reference Implementations

OSSA is not a proposal — it is running in production. We invite NIST reviewers to verify these live systems:

Live Systems

System	URL	What It Demonstrates
OSSA Website	openstandardagents.org	Specification documentation, interactive builder, playground
DUADP Protocol v0.2.0	duadp.org	Discovery protocol specification, NIST alignment page
DUADP Discovery	discover.duadp.org	Live federated agent discovery across DUADP nodes
Agent Marketplace	marketplace.blueflyagents.com	Browse OSSA-compliant agents with verified manifests
Compliance API	compliance.blueflyagents.com/api/v1/cedar/policies	16 Cedar policy files (229 policies) served via REST API
Agent Builder	openstandardagents.org/builder	Create, validate, and export OSSA agents
Schema Explorer	openstandardagents.org/schema	Interactive OSSA JSON Schema browser

Open Source Repositories

Repository	License	Description
OSSA Specification	MIT	Core specification, JSON Schema, CLI tooling
Cedar Policies	MIT	2,892 lines of agent authorization policies
DUADP Protocol	MIT	Discovery protocol specification and reference node
OSSA CLI	MIT	`npx @bluefly/openstandardagents@0.4.8` — init, validate, export

Scale Metrics

- **16 Cedar policy files** covering agent guardrails, resource limits, roles, branch protection, code review, deployment rules, MCP security, plugin governance, and more
- **2,892 lines of Cedar policy code** — formally verified, production-enforced
- **229 individual Cedar policies** served via the compliance API
- **932 static pages** generated for the OSSA website
- **17 MCP tools** in the DUADP discovery protocol
- **Platform exporters** for Docker, Kubernetes, CrewAI, LangChain, and npm

---

Recommendations to NIST

Based on our experience building and deploying OSSA, we respectfully offer the following recommendations to the CAISI initiative:

Identity

1. **Mandate persistent, globally resolvable agent identifiers** conformant to W3C DID Core 1.0. Agent identity must survive restarts, migrations, and redeployments.
2. **Require cryptographic binding** between agent identity and agent manifest. Self-attestation is insufficient for federal contexts.
3. **Standardize revocation propagation** with sub-minute latency requirements. Revocation events should use established protocols (AsyncAPI, WebSocket, SSE).

Authorization

1. **Adopt formally verified policy languages** (e.g., Cedar) for high-assurance agent authorization. Policy languages must be provably terminating, decidable, and consistent.
2. **Mandate pre-authorization** for all tool execution by federal AI agents. "Run first, audit later" is incompatible with federal security requirements.
3. **Require full principal chain recording** in authorization decisions. Every agent action must be traceable to the originating human principal.

Interoperability

1. **Standardize agent manifests** as machine-readable, schema-validated declarations of identity, capabilities, tools, and constraints. We offer the OSSA manifest format as a starting point.

2. **Define a minimal agent credential format** based on W3C Verifiable Credentials for cross-organizational delegation and trust.
3. **Adopt federated discovery protocols** that do not depend on centralized registries. We offer DUADP as a reference implementation.

Governance

1. **Mandate resource budgets** in agent specifications. Token consumption, API call limits, and execution time constraints should be declared, not configured ad hoc.
2. **Require structured, machine-readable audit logs** exportable to SIEM systems. Audit records must include the authorization decision, not just the action.
3. **Map agent security controls to OSCAL** for automated compliance assessment against NIST SP 800-53.

Conclusion

The AI agent ecosystem is at an inflection point. Without standards, every deployment is a bespoke security problem. With the right standards, AI agents can be as auditable, interoperable, and controllable as any other software component in a federal system.

OSSA provides a concrete, production-tested answer to the questions this RFI poses. It is not perfect — no first-generation standard is — but it is real, it is open, and it is running. We have built the specification, the tooling, the policies, the discovery protocol, and the reference implementations. We have deployed them. We have used them to govern real AI agents in production.

We invite NIST to review OSSA v0.4 as a reference implementation for the CAISI initiative. We are ready to collaborate on aligning the specification with federal requirements, contributing to working groups, and supporting the development of national standards for AI agent security.

The infrastructure exists. The policies are enforced. The agents are governed. We look forward to building the future of secure AI together.

Respectfully submitted, **BlueFly.io Platform Team** *On behalf of the OSSA open-source community* info@blueflyagents.com | openstandardagents.org

---

Exhibits

Exhibit A — OSSA v0.4 Manifest Schema Excerpt

```yaml
apiVersion: ossa/v0.4
kind: Agent
metadata:
  name: federal-document-processor
  namespace: gov.agency.documents
  labels:
    compliance: nist-800-53
    classification: CUI
identity:
  did: "did:web:agency.gov:agents:doc-processor"
  trustTier: organizational
  x-signature:
    algorithm: Ed25519
    publicKey: "z6MkhaXg..."
    signature: "z3FXQjeZ..."
role: |
  Process, classify, and summarize federal documents according to
  agency retention policies and classification guidelines.
llm:
  primary: claude-sonnet-4-5
  fallback: gpt-4o
  constraints:
    max_tokens_per_call: 4096
    temperature: 0.3
tools:
  - name: search_documents
    protocol: mcp
    sensitivity: medium
    rate_limit: 30/minute
  - name: classify_document
    protocol: mcp
    sensitivity: high
    rate_limit: 10/minute
  - name: generate_summary
    protocol: mcp
    sensitivity: low
    rate_limit: 20/minute
security:
  tier: tier_2_write_limited
  capabilities: [read_documents, write_summaries, classify]
  constraints:
```

```
      max_calls_per_session: 100
      max_total_session_tokens: 50000
      allowed_namespaces: ["gov.agency.documents"]
  autonomy:
    level: supervised
    approval_required: true
    human_oversight: always
  extensions:
    token-efficiency:
      budgets:
        max_tokens_per_call: 4096
        max_total_session_tokens: 50000
      optimization:
        cache_enabled: true
    governance:
      cedar_policy_set: "gov-agency-document-processing"
      oscal_component: "./ossa-oscal-component.json"
      audit_level: comprehensive
```

Exhibit B — DUADP Protocol Operations

Operation	Method	Endpoint	Description
Discover Agents	GET	`/discover`	Federated agent search by capability, namespace, or keyword
Register Agent	POST	`/registry/agents`	Register an OSSA agent with a DUADP node
Resolve DID	GET	`/resolve/{did}`	Resolve agent DID to manifest and endpoints
Revoke Agent	PUT	`/registry/revoke/{agentId}`	Immediately revoke an agent's registration
Federation Sync	POST	`/federation/sync`	Synchronize agent metadata across DUADP nodes
Health Check	GET	`/health`	Node health and federation status

Operation	Method	Endpoint	Description
Subscribe Events	WS	`/events`	Real-time agent lifecycle events (register, revoke, update)

Protocol standards: OpenAPI 3.1 (REST), AsyncAPI 3.0 (events), WebFinger (discovery), W3C DID Core (identity resolution)

Exhibit C — OSSA Trust Tier Table

Tier	Name	Verification Method	Cedar Policy Attribute	Federal Use Case
0	Community	None (schema validation only)	`principal.trust_tier == 0`	Sandbox, experimentation
1	Self-Signed	Agent manifest passes schema validation	`principal.trust_tier == 1`	Internal development, testing
2	Verified	Manifest signature verifies against registered public key	`principal.trust_tier == 2`	Cross-team, staging
3	Organizational	Signed by organization's root key + full audit trail	`principal.trust_tier == 3`	Federal production, FedRAMP

Exhibit D — Complete Threat Matrix

See Section 4d above for the full threat matrix mapping OSSA controls to NIST SP 800-53 Rev. 5 control families.

Exhibit E — Cedar Policy Inventory

Policy File	Policies	Lines	Domain
agent-guardrails.cedar	19	272	Worktree, commit, branch, rebase, file protection
agent-resource-limits.cedar	14	185	Max agents, tool calls, execution time, concurrent requests
agent-roles.cedar	30	293	Analyzer, executor, reviewer, orchestrator

Open Standard Agents

OSSA is an open standard for AI agents. The vendor-neutral specification was created by Thomas Scola, founder of Bluefly.io.

Docs

Getting Started

Specification

Full Docs

Roadmap

Schema

Resources

Examples

Playground

Blog

Design Guide

Community

Discord

GitLab

GitHub

npm

PROJECT STATUS

OSSA is an early-stage open standard for AI agent interoperability. Vendor-neutral, Apache 2.0 licensed, and designed for production deployments. Contributions, feedback, and adoption stories welcome.

MCP Native | Docker | K8s | OpenTelemetry | YAML Manifest