ChatGPT

Authorization and Workflow Model for Agentic AI Agents

Introduction

AI agents with autonomy ("agentic agents") have emerged as powerful tools that can interpret goals, plan actions, and interact with various systems. Unlike traditional software, these agents do not follow rigid, predetermined code paths – they make real-time decisions, call external tools, and can even modify their behavior based on context and learned knowledge [1]. This flexibility enables complex workflows, but it also **introduces significant security and safety challenges**. A misbehaving or compromised agent could combine its wide-ranging capabilities in unintended ways, potentially causing data leaks, destructive actions, or other damage [2] [3]. In essence, an overly empowered agent can become *"a ticking time bomb with the blast radius of a small nuclear device"* if not properly controlled [4].

To address these risks, we propose a comprehensive **authorization and workflow control model** for agentic AI. The goal is to **limit the agent's "blast radius"** – i.e., the potential scope of harm – by enforcing strict guardrails on what the agent can do, when it can do it, and with which permissions. Our approach combines:

- **Controlled execution flow graphs** that predetermine allowable action sequences for tasks.
- **Fine-grained, dynamic authorization** using short-lived tokens and per-action permissions.
- **Simulated execution environments** (development sandboxes, CI evaluation pipelines, and production *digital twins*) to test and verify agent behavior safely.
- **Maximal observability and logging** to trace agent decisions and outcomes for accountability.

By integrating these elements, we aim to maintain the benefits of agentic flexibility *without* the unchecked risk, thus enabling confidence in deploying AI agents for real-world tasks.

Challenges with Unconstrained Agentic AI

Unpredictable and Over-Scoped Behavior: Agentic AI systems can access numerous tools and data sources, effectively expanding their attack surface across multiple domains [5]. If given broad permissions, an agent might overstep its intended role – either accidentally or due to malicious input – resulting in unintended actions. For example, a calendar-reading agent might, if over-privileged, attempt to modify or delete unrelated records, or an agent meant to send a few emails could spam thousands of messages if prompted incorrectly [2]. Traditional identity & access models (which often grant applications more privileges than needed) are ill-suited here, because an AI agent's *code and intent can morph at runtime*. In other words, **the union of all permissions given to an agent defines the worst-case actions it could take**, and an autonomous agent might inadvertently exercise any of them under the right (or wrong) conditions.

Prompt Injection and Contextual Exploits: AI agents rely on instructions (prompts) and context to decide actions. Attackers can manipulate these inputs (via prompt injection or context poisoning) to trick agents into violating policies [6] [7]. Since the agent itself lacks human common sense or built-in

ethics enforcement, it might execute harmful instructions unless external safeguards stop it [8]. This means **we cannot rely on the agent to self-police its actions**. As one industry expert noted, *"You can't reason with an LLM about whether it should delete a file. You have to design hard rules that prevent it from doing so."* [9].

Credential and Permission Sprawl: In enterprises, each agent may need access to numerous APIs, databases, and services, each requiring credentials or API keys. This can lead to **credential sprawl** – a proliferation of tokens and secrets, often long-lived and over-scoped [10] [11]. If an agent's token is stolen or misused, an attacker might leverage it to laterally move through systems just as the agent could, potentially causing widespread compromise [5] [12]. Moreover, many organizations lack visibility into what permissions each agent actually has; "shadow" agents or forgotten test agents with leftover permissions are common [13]. This makes it hard to trace or audit agent activities and enforce governance.

No Context Boundaries: Without careful design, an agent might carry sensitive data from one context into another inappropriate context (for instance, reading confidential data then inadvertently sending it out via an email or web request) [14]. If all capabilities are available at once, nothing stops an agent from combining them in dangerous ways. A notorious "toxic combination" cited in security is having an entity that can **(1) access untrusted inputs (e.g. browse the web), (2) handle sensitive data, and (3) perform privileged actions (e.g. send emails or modify records)** – all at the same time. No human user would ordinarily be allowed to do *all* of those without oversight, and an agent shouldn't either. **Our model ensures an agent can never hold all such powers simultaneously**.

In summary, **the fundamental challenge is to enforce least privilege and contextual limits on a non-deterministic, creative piece of software.** We must assume an agent will eventually try *anything* it has the permission to do – whether by accident or malicious prompting – and plan for containment [15] [16].

Controlled Workflow Execution Graphs

The first pillar of our approach is to define **controlled execution flow graphs** for the agent's tasks. Instead of allowing the agent to improvise arbitrary sequences of tool use, we specify a directed graph of possible action steps that the agent can take to accomplish a goal. Each node in this graph is an atomic action (e.g. "Read Calendar," "Fetch Web Page," "Send Email") with clearly defined input/output schemas and required permissions. Each edge defines an allowed transition (flow) from one action to another.

Key aspects of the workflow graphs include:

- **Explicit Stage Definition:** The task is broken into stages (or sub-tasks) that must happen in order. For example, Stage 1: read user's calendar; Stage 2: search the web for relevant information; Stage 3: update calendar or send an email summary. The agent can choose *how* to execute within a stage (or whether the stage is necessary based on runtime conditions), but it cannot skip or rearrange the stages outside the approved graph. This ensures the overall logic follows a vetted pattern.

- **Per-Stage Schemas:** Each stage has an expected input and output format. For instance, the calendar-reading stage expects a date range as input and produces a list of events as output. The web-browsing stage takes a URL or query and produces extracted data. By constraining data

formats, we reduce ambiguity and the chance of prompt injection hiding in unexpected input forms. It also helps in validating the agent's behavior at each step against the expected schema.

- **Allowed Tools and Actions:** The graph explicitly ties each step to the specific tool or API the agent should use. For example, "Search web" might map to a web search API endpoint, whereas "Send email" maps to an email-sending microservice. **The agent is limited to this universe of allowed actions** – it cannot call tools outside what the graph enumerates. This is akin to giving the agent a menu of safe, pre-approved operations.

By pre-defining execution flows, we **decouple the agent's high-level reasoning from the low-level execution**. The agent still has flexibility (it can decide which branch of a flow to take depending on context, or whether certain optional steps are needed), but it *cannot invent completely new action sequences*. Essentially, the agent's autonomy is sandboxed within a controlled workflow graph.

This concept mirrors software engineering practices like *finite state machines or flow charts*, and it aligns with emerging frameworks for AI agents. For example, Anthropic's **Model-Context Protocol (MCP)** and similar standards aim to formalize how an agent can interact with tools by defining available actions and required contexts [17]. In our model, the control flow graph serves as a policy that **binds the agent's possible paths of execution to a safe subset of behaviors**.

Fine-Grained, Dynamic Authorization per Action

Even with controlled flows, we must prevent an agent from abusing any single action. The solution is **fine-grained, time-bound authorization tokens** issued *per action and per instance of that action*. In practice, we introduce two proxy components around the agent:

- **1. An Identity Provider (IdP) for Agents:** This service issues short-lived tokens or credentials for specific actions. Whenever the agent is about to perform an action (as allowed by the workflow graph), it must request a token from the IdP for that action. The IdP will authenticate the agent's identity and verify that the action is permitted in the current context (correct stage of the workflow, agent has any necessary higher-level approval, etc.). The token is scoped *only to that action and resource* – for example, a token that allows "read from Calendar API for user X, valid for the next 60 seconds" or "HTTP GET to domain Y's API, one-time use." Crucially, the agent does **not** hold a blanket credential with broad access; it only obtains what it needs **just-in-time** and **just-enough** for each step [11] [18].

- **2. A Secure Execution Proxy:** This component receives the agent's action request along with the token, and it actually carries out the action on behalf of the agent (e.g., making the API call to the external service). The proxy validates the token and ensures the request matches the token's scope (no sneaky elevation of what is being done). If valid, it executes the call and returns results to the agent. If the agent attempts an action without a token or with the wrong token, the proxy denies it. This proxy can also sanitize inputs (to mitigate injection attacks further) and enforce output schemas.

By splitting responsibilities in this way, **the agent never directly possesses long-term credentials to sensitive systems**. Each token is ephemeral and tightly scoped. Even if an agent is compromised or tricked, the damage is limited to what that single token allows, which expires quickly. As noted in industry best practices, *"resource-level permissions reduce the blast radius if something goes wrong, ensuring agents can't overstep — even by accident."* [19]. Likewise, using **short-lived tokens** that force frequent re-authentication means any stolen credential has a very brief window of usefulness [20].

This dynamic authorization is analogous to the concept of *microsegmentation* or *just-in-time access* in Zero Trust security models. Instead of granting an AI agent static broad roles, we apply **the Principle of Least Privilege in real-time**: an agent gets only the minimal rights needed *for the current step, and nothing else* [18]. When it moves to the next step, it must request a new token for that step's privileges, and any prior token (and permission) is revoked or expires.

Sequential Permissioning Example: Consider the earlier example task with three steps – read calendar, browse web, update calendar & send email. Under our model: - When the agent starts **Step 1 (Read Calendar)**, it obtains a token that allows read-only access to the calendar service. The token has no internet privileges, no email-sending rights. The execution proxy will only allow calendar read operations with it. - For **Step 2 (Web Browse)**, the agent cannot reuse the calendar token. It must get a new token from the IdP that perhaps allows HTTP GET requests via a web proxy. The IdP confirms the agent is indeed now in the "Browse" stage (as per the workflow graph) and issues a token limited to web access. At this time, the agent no longer has valid calendar access (the previous token is expired or one-time use). So even if the agent were somehow manipulated to try a calendar operation during the browsing step, it would be blocked. - Finally, for **Step 3 (Update & Email)**, separate tokens are issued: one to update the calendar entry (with permission only to write a specific entry or to call a specific API route), and another to send an email (perhaps scoped to contacting a particular recipient or sending via a monitored email service). These are only acquired when the agent reaches that stage. They too expire quickly after use.

Through this approach, at **no point in time** does the agent hold a combination of privileges that would enable a catastrophic action (like exfiltrating data from a confidential source and exfiltrating it to an external site). Each token is like a single-key that opens one small door and then disappears. This method significantly contains the "blast radius" of agent actions: even if one step is exploited, it cannot directly escalate into a multi-step disaster without the agent explicitly and legitimately obtaining the next permissions (which our monitoring could catch if something anomalous was attempted).

Another benefit of this model is **centralized oversight and policy control**. Since the identity provider mediates every access, administrators can enforce global rules (e.g., time-of-day restrictions, approval requirements for certain actions, anomaly detection) before granting a token [21] [20]. For example, if an agent suddenly requests an unusual permission or a high-risk action outside of expected usage patterns, the system can flag or even halt it for review – effectively injecting a human-in-the-loop or an automated policy check when needed.

Modern authorization systems and identity frameworks are evolving towards this granular, context-aware approach. It aligns with the idea of **context-aware authorization**, where the decision to allow an action considers factors like the current task, the data sensitivity, and the agent's recent behavior [21] [22]. Our model implements context-awareness by tying permissions to the *specific stage in the workflow and task context*. It also resonates with the emerging standards such as OAuth 2.1 usage for AI agents (providing secure, automatic auth flows for non-human identities) [23] and proposals to treat AI agents as first-class identities in Identity and Access Management (IAM) systems [24] [25].

Simulated Environments and Digital Twins for Safe Testing

Even with strong workflow and authorization controls, it is crucial to **test and validate an agent's behavior in a safe environment** before and during deployment. We advocate a multi-stage execution environment strategy:

- **Development & Unit Testing Environment:** During development, each agent (or each new workflow added to an agent) should be run in a **simulated environment** with *surrogate dependencies*. This means all external tools or services the agent would call are replaced with mock or stub versions that mimic real responses but **do not have real-world side effects**. For example, a dummy calendar API that returns sample data, or a fake email service that logs the email content instead of actually sending it. The agent under development should not be able to tell it's not interacting with the real systems – from its perspective, everything is functioning normally. This way, developers can safely observe how the agent formulates its actions, ensure it adheres to the schemas, and that the control flow graph covers the necessary branches. It dramatically improves development speed and confidence because you can run the agent over and over, even in error scenarios, without any risk to actual data or systems.

- **CI Pipeline Evals (Continuous Integration Testing):** As agents are integrated into larger systems, we incorporate automated **evaluation tests (evals)** in a CI/CD pipeline. These evals consist of predefined scenarios or prompts fed to the agent in the simulated environment, checking that the agent's behavior matches expectations and does not violate any constraints. Essentially, this is an **autonomous agent test suite**. For instance, tests may verify that an agent *cannot* perform disallowed actions (negative tests for security) – e.g., try to make it fetch a file it shouldn't and ensure the request is blocked and logged. Other tests check that it accomplishes the goal through the intended sequence and produces correct results. By running these evals on each update, we prevent regressions and ensure that safety measures remain effective even as the agent's LLM or prompt templates evolve over time.

- **Staging/QA Environment:** Before going live, agents should be deployed in a staging environment that mirrors production as closely as possible (including connecting to real services but perhaps with non-production data). Here the agent operates with the full stack of controls (our identity proxy, execution proxy, etc.) against **non-critical data**. It's a final dress rehearsal to confirm everything works end-to-end in a realistic setting.

- **Production with Digital Twins and Simulation Mode:** In production use, we add a twist: the concept of a **digital twin** for critical actions. A *digital twin* in this context is a virtual replica of the environment or target resource where an agent's action can be simulated **without real-world consequences** [26]. For example, before an agent executes a high-impact action (like making a large financial transaction or deleting many records), the system could run that action in a sandboxed replica to see the outcome. If the result is acceptable, the real action proceeds; if not, it can halt or alert an operator. This approach of *"simulate before commit"* provides an extra safety net. It's akin to a "dry run" mode for agents, which is invaluable given their unpredictability. As Trend Micro's research notes in the context of cyber-defense, digital twin simulations allow safe testing of "what if" scenarios without endangering actual systems [26] – here we apply the same principle for any agent-initiated operation.

Additionally, production agents can be continually monitored by running **parallel evaluation tasks**: essentially, continuously or periodically feed the agent known test prompts or scenarios *in the background* (in a test mode) to ensure it's still behaving as expected. This is important because AI

behavior can drift over time due to model updates or even as the world/state changes. By **continuously running evals in production**, we can catch subtle issues early – for instance, after 1000 real executions, perhaps the agent starts generalizing in a wrong way. Continuous testing in a twin environment would flag the misbehavior before it causes serious trouble in the real environment.

- **Rolling Out with Caution:** When deploying new capabilities, one can run the agent in a shadow mode where it simulates actions and logs what it *would* have done, without actually performing them, until we are confident. Only then do we allow the agent to truly act. This phase-wise rollout ensures maximum confidence in the agent's compliance with policies.

The use of **digital twins and sandbox testing** also provides a trove of data for analysis. We can measure how often an agent attempted disallowed actions, how it responded to failures or denials, and improve its prompt or the control policies accordingly. It reinforces the principle that *"until you test it, every scope is Schrödinger's permission"* – you don't know if a permission is truly needed or if a policy truly holds, until you simulate and observe [27] [28] . Many organizations have found that by testing agents in sandbox conditions, they could **strip away 80-90% of permissions with no loss of functionality** [29] , revealing just how over-scoped initial assumptions might have been.

In sum, a layered testing strategy from development to production ensures that **agents operate under known conditions and their safety measures actually work**. If an agent ever deviates or tries something outside its bounds, it's caught in a safe environment first. This dramatically reduces the chance of a nasty surprise in live operation.

Observability, Logging, and Explainability

A fundamental requirement for trust in autonomous systems is **observability** – the ability to understand and trace what the agent is doing and why. Our model emphasizes robust logging and tracing at every layer:

- **Action Logs:** Every action the agent attempts (authorized or denied) is logged with detail. Because all tool usage goes through the execution proxy, we can record: the action name, parameters used, time, the outcome, and which token (identity) was used. This provides a complete audit trail of the agent's activities [30] . In case of an incident, these logs are invaluable for forensic analysis and root-cause determination. They also help in compliance, answering questions like "who (which agent) accessed this data and when?" in a human-readable way.

- **Decision Trace (Reasoning Logs):** In addition to external actions, we log the agent's internal reasoning to the extent possible. For instance, the prompts and responses from the LLM, any chain-of-thought the agent produces, and the decisions it makes at branch points in the workflow. This is important for **explainability** – if the agent does something unexpected, we need to trace back and see the series of prompts or intermediate conclusions that led to it [31] . Storing this in a graph database (as hinted by the speaker with an "M graph DB") can enable querying and analyzing the decision paths.

- **Monitoring & Anomaly Detection:** With comprehensive logs, we can apply monitoring tools to detect anomalies. For example, if an agent suddenly calls an API it never did before, or starts outputting much larger data than usual, those can trigger alerts. This ties into the idea of a "black box recorder" for AI agents, which some have called for [32] – you want a continuous record of agent behavior to notice subtle drifts or emerging issues. Observability also means capturing metrics: number of actions per hour, error rates, distribution of tool usage, etc., to

establish baselines of normal operation. Any deviation can be a sign of trouble (whether a technical bug or a security issue).

- **Provenance and Data Tracking:** If an agent reads data from one source and later uses it elsewhere, our system can tag the data (or logs) to trace that flow. This helps ensure that, say, confidential information read in step 1 isn't inadvertently included in a public-facing output in step 3. By having each action log include what data was read or written (when feasible, via identifiers or hashes), we add **provenance tracking**. In case of a leak or error, we can answer "where did this piece of information come from and where did it go?".

All these logs and traces should be accessible for developers and auditors through appropriate tools. We envision something like an **"agent control center"** dashboard where one can review an agent's recent runs, inspect each action (with links to logs), and even replay a sequence in simulation if needed. This level of transparency is key to building trust in agentic systems – it addresses the *"lack of traceability"* problem noted in enterprise AI deployments [33]. Rather than the agent being an opaque decision-maker, we make its operation as transparent as a traditional system (if not more, since we log *everything* it does, not just errors).

Moreover, the presence of thorough logging itself has a security benefit: it imposes accountability. If agents know (or rather, are configured such) that every move is monitored, there's no "operating in darkness." As an extension, **explainability tools** (e.g., capturing the chain-of-thought) can be integrated to provide higher-level explanations for why the agent chose a certain path. This can help in refining the workflows and prompts over time for safer behavior.

Discussion: Balancing Flexibility and Control

The proposed model is comprehensive, but it does introduce complexity. There are practical considerations and challenges:

- **Performance and Latency:** Requiring the agent to obtain a token for each action and possibly simulate actions before real execution can add overhead. We need to ensure the identity provider and execution proxy are highly available and fast, so they don't bottleneck the agent unduly. Techniques like caching certain low-risk authorizations, or combining multiple related actions under one token when safe to do so, might be employed to streamline performance. However, security is often a trade-off with convenience, and we err on the side of safety here. Short-lived tokens and checks can be optimized, but not eliminated, in high-stakes environments.

- **Integration with Existing Systems:** Implementing this model in an enterprise means integrating with existing Identity and Access Management (IAM) systems and tool APIs. The concept of issuing scoped tokens per action might leverage existing standards like OAuth/OIDC with fine-scoped grants, or newer standards like OAuth 2.1 and DCR (Dynamic Client Registration) for agents [23]. We may need to build a layer on top of traditional IAM to handle the dynamic context (since most IAM assume static roles/permissions). Some vendors are already moving this direction, advocating that each AI agent be treated as a unique identity (service account) with very granular scopes [25] [34]. Our model extends that to *each action* of the agent. This likely requires custom development today, but it aligns well with Zero Trust principles being discussed for agent security [35].

- **Defining the Workflow Graphs:** For every new task or agent, someone has to design the allowed flow graph and enumerate the actions, schemas, and permissions. This is a new kind of design activity – somewhat akin to writing a program specification or an API contract. It requires collaboration between developers (who understand what the agent *should* do) and security engineers (who impose what it *must not* do). Tools or languages for describing these graphs could evolve (perhaps an extension of OpenAPI/Swagger for action sequences, or policy-as-code frameworks). Policy-as-code is indeed recommended to enforce such rules centrally [36]. Over time, libraries of common workflows might emerge so that not every graph is from scratch.

- **Maintaining Least Privilege:** Agents' tasks might evolve, which means the workflow and permission scopes should be revisited regularly. Our logging of unused permissions can help here: if an agent never uses a particular permission in any of its runs, that scope can likely be removed – a process of continuous **scope reduction** [37] [29]. Regular audits of agent permissions are essential, just as one would audit human users or service accounts. The difference is that with agents, those permissions are often ephemeral in our model; what we audit is the template of what *could* be issued. Still, principles like "start with nothing and add incrementally" apply [38] – by default an agent should begin with no access until a policy grants it step by step.

- **Fail-Safe Mechanisms:** If our control infrastructure fails (e.g., the token service goes down), the agent should ideally fail *closed* (i.e., do nothing) rather than default to an open, unsafe mode. Designing for high reliability in the control plane is crucial; otherwise, a downtime could paralyze all agent operations. Caching some decisions or having an offline mode is possible but must be done carefully to not inadvertently grant permanent permissions.

Despite these challenges, the direction is clear: **to safely harness agentic AI, we must impose structure and limits externally rather than expecting the agent to inherently be safe**. This approach echoes the consensus emerging in industry. As summarized by one framework: *"Agents aren't people, and they can't be governed as if they were. Assign ownership. Scope least privilege. Prefer federation over static secrets. Monitor actions. Retire fast."* [39]. Our model embodies these principles by design – each agent action is owned and approved through identity tokens, least privilege is enforced per step, static long-term secrets are replaced by federated tokens, every action is monitored, and credentials are short-lived (retired quickly).

Conclusion

Agentic AI systems offer unprecedented automation and adaptability, but without robust guardrails, they pose equally unprecedented risks. In this paper, we presented a holistic authorization and workflow model that enables **trustworthy and controlled execution of AI agents**. By constraining agents to predefined workflow graphs and issuing them ephemeral, scoped permissions for each action, we significantly reduce the potential damage from errant behavior or compromise. Our model's emphasis on simulation (digital twins, sandbox testing) and continuous monitoring further ensures that agents behave as expected and that any deviation is caught early in a safe manner.

This approach transforms the agent from a wild card element into a governed entity akin to a well-audited microservice – albeit one that is creative and autonomous within the bounds we set. It brings the best practices of software engineering and security (like least privilege, testing, auditing, Zero Trust) into the realm of AI agents. Early evidence suggests that such strict scoping does not hinder functionality: many agents can accomplish their tasks with only a fraction of the permissions they would

have otherwise been given [37]. What we sacrifice in a bit of development overhead and complexity, we gain in **confidence and safety**, which is a worthy trade-off when deploying AI in mission-critical roles.

Moving forward, organizations implementing agentic AI should consider adopting similar models. It will involve cultural shifts (treating AI "employees" with the same rigor as human ones in terms of identity and access) and new tooling. The payoff is resilient AI systems that can be trusted not to "break everything" at machine speed [40]. We envision a future where AI agents can be *freely creative in problem-solving, yet fundamentally incapable of breaching the safety constraints* set by their human operators. Achieving this will unlock the true potential of agentic AI, allowing it to speed the business without widening the blast radius [41].

Sources:

- Eric Olden, "Over-scoped agents: The permission sprawl that will end you," *Strata Blog*, Oct 2025. [4] [42]
- Anirudh Murali, "Safeguarding the Enterprise AI Evolution: Best Practices for Agentic AI Workflows," *ISACA Industry News*, 2025. [5] [20]
- Graham Neray (CEO of Oso) et al., "Best Practices of Authorizing AI Agents," *Oso Blog*, 2023. [18] [30]
- WorkOS, "Securing AI Agents: A guide to authentication, authorization, and defense," Jun 2025. [19] [43]
- Strata Identity, "The least privilege playbook for agents" and "The Sandbox: your safe space for scope surgery," *Agentic Identity* series, 2025. [44] [29]
- Trend Micro, "Using Agentic AI & Digital Twin for Cyber Resilience," May 2025. [26]

---

[1] [3] [6] [7] [8] [19] [21] [43] Securing AI agents: A guide to authentication, authorization, and defense — WorkOS
https://workos.com/blog/securing-ai-agents

[2] [9] [14] [16] [17] [18] [22] [23] [30] Best Practices of Authorizing AI Agents
https://www.osohq.com/learn/best-practices-of-authorizing-ai-agents

[4] [13] [15] [27] [28] [29] [32] [36] [37] [38] [40] [42] [44] Over-scoped agents: The permission sprawl that will end you | Strata.io
https://www.strata.io/blog/agentic-identity/over-scoped-agents/

[5] [10] [11] [12] [20] [24] [25] [31] [33] [34] [35] Industry News 2025 Safeguarding the Enterprise AI Evolution Best Practices for Agentic AI Workflows
https://www.isaca.org/resources/news-and-trends/industry-news/2025/safeguarding-the-enterprise-ai-evolution-best-practices-for-agentic-ai-workflows

[26] Using Agentic AI & Digital Twin for Cyber Resilience | Trend Micro
https://www.trendmicro.com/en/research/25/e/ai-digital-twin-cyber-resilience.html

[39] [41] Agentic Access Management Framework | Secure AI Access
https://www.oasis.security/blog/agentic-access-management-framework