



Kent Academic Repository

Gupta, Shivangi, Arief, Budi and de Lemos, Rogério (2026) *Agentic AI Gone Rogue: Addressing the Limitations of Static Access Control*. In: 2026 International Workshop on Security and Artificial Intelligence (SECAI 2026). . (In press)

Downloaded from

<https://kar.kent.ac.uk/116436/> The University of Kent's Academic Repository KAR

The version of record is available from

This document version

Author's Accepted Manuscript

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in **Title of Journal**, Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our [Take Down policy](https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies) (available from <https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies>).

Agentic AI Gone Rogue: Addressing the Limitations of Static Access Control

Shivangi Gupta^[0009–0007–8240–8091], Budi Arief^[0000–0002–1830–1587], and
Rogério de Lemos^[0000–0002–0281–6308]

School of Computing, University of Kent, Canterbury, UK
`{sg106,b.arief,r.delemos}@kent.ac.uk`

Abstract. As agentic AI systems become increasingly capable of acting autonomously on behalf of users, they reduce users’ workload and improve the efficiency of achieving goals. However, they also introduce security challenges, such as privilege escalation due to broad access permissions granted to agents, and the lack of complete delegation chain verification at the server side. To address the lack of delegation chain verification, we introduce a decentralised delegation and verification approach that uses a signed JSON Web Token (JWT). The proposed approach performs continuous trust checks on every delegation before the agent reaches the server for execution. Agents’ authenticity is established through signed agent cards, whereas verification and authorisation are managed through a combination of tokens and a gateway integrated with a policy engine that enforces the permissions granted by the user. This approach ensures that agents are authorised only to perform user-approved tasks and can exercise only the permissions explicitly granted by the user for those tasks. Our approach aims to motivate a shift from the current methods (such as providing agents with access tokens without verifying what the user actually approves and delegates, which rely on predefined scopes or permissions that can be overly broad or restrictive), to fine-grained user-approved permissions. To evaluate the proposed solution, the approach was tested against ten different threats that are commonly found in agentic AI deployments. Based on the results, the proposed approach shows promising potential for verifying delegated permissions and ensuring that delegation is limited to permissions that are explicitly approved by the user. Such features are currently missing in existing frameworks, which tend to verify delegation requests only once they reach the resource server, which can lead to partial chain verification, and users’ actual requirements being missed.

Keywords: Agentic AI · Trusted Delegation · Continuous Verification · Dynamic Authorisation

1 Introduction

Computer-based systems typically rely on static, predefined steps that require a high level of human interaction and oversight. The emergence of artificial

intelligence (AI) changed that, allowing such systems to become smarter, and to require less human involvement. The recent development of *agentic AI* – in which AI agents are no longer limited to serve just as chatbots [24,48] – sees AI agents becoming goal-oriented and more autonomous by decomposing bigger goals into smaller sub-tasks. For instance, by leveraging large language models (LLMs), AI agents can make decisions towards the goal fulfilment using their own reasoning [48]. In agentic AI systems, agents work as first-class entities with their own identities, responsible for calling the Application Programming Interfaces (APIs), and working on behalf of the user to re-plan the strategy based on the given goal, thus becoming capable of adapting to a changing context.

Although these developments tend to benefit human users (e.g., by reducing human users’ workload), they also introduce new security vulnerabilities, such as privilege escalation and memory poisoning [33,51]. Specific to the work presented in this paper, the transition to agentic AI systems has posed challenges to traditional security authorisation frameworks [42], such as OAuth [25]. In these frameworks, the user must grant access to third-party applications (clients) to act on their behalf, without sharing their credentials with those applications. Based on the user’s approval, the *authorisation server (AS)* generates an access token, allowing the client to access resources on the *resource server (RS)* [5,25].

In comparison, agentic AI requires an approach that can dynamically authenticate and authorise agents automatically and seamlessly. This autonomy raises important trust and verification challenges [42], e.g., how to authenticate and authorise short-lived agents securely across multiple trust boundaries and how to verify the execution of sensitive operations in the absence of human oversight.

To deal with the challenges associated with securing the agentic flow, we propose an approach that combines JSON Web Token (JWT) [31] and JSON Web Key Set (JWKS) [30], together with decentralised delegation, signed agent cards, a gateway, a policy engine, and continuous trust verification. This combination is chosen because JWT and JWKS are widely adopted for token representation and public key verification, respectively. The agent cards are lightweight JSON-based files that are easy to host at an endpoint to advertise agents’ skills and capabilities [52]. The agent cards in our approach are signed by a trusted developer company that creates an agent, therefore, proving the authenticity and integrity of the agent card (unsigned agent cards are currently observed as a security gap by OWASP in their documentation [10]). The gateway and policy engine are used to verify the complete chain from the user to the agent (since verifying the complete chain is not possible directly at the server), and then allowing access based on what the user authorises. The choice of combining a gateway with a policy engine is motivated by recent research on the security of agentic AI systems [13].

In our approach (detailed in Section 5), agents are authenticated using signed agent cards. Authorisation is enforced through JWTs and a gateway integrated with a policy engine. The proposed approach fills the research gap in verifying the delegation chain from user to agent (including multi-hop systems). With this approach, implementers can identify what was initially delegated and authorised

by the user, the scopes and resources requested by the agents down the chain are restricted based on the user’s approved delegated permissions.

This paper provides two key contributions: (i) An analysis of limitations of existing authorisation mechanisms, such as OAuth, in validating the delegation chain from user to agent; (ii) A proposal of an alternative approach for authorisation and authentication to validate the delegation chain from user to agent.

2 Related Work

The execution flow of an AI agent has changed from reactive to goal-oriented architecture, where agents may use LLMs for reasoning and planning, memory to ensure consistency, and tools to fetch data [35]. The change from a single agent system to a multi-agent system requires secure coordination and communication. This communication between agents is achieved using the concept of delegation that was studied prior to agentic AI through taxonomies characterising delegation along dimensions such as direct versus indirect sub-delegation, capability versus responsibility, and grant versus transfer semantics [43]. For efficient communication, companies introduced several protocols such as Agent2Agent (A2A), Agent Communication Protocol (ACP), and Agent Network Protocol (ANP) [35]. However, each protocol has its own pros and cons. For example, A2A and ACP introduced the concept of inter-domain communication [35], but they still rely on existing authentication and authorisation frameworks, which are centralised and created for humans. ANP introduced decentralised identifiers (DIDs) for authentication [18,23], where authorisation of low-risk tasks is handled by agents, and high-risk tasks are authorised by humans [18]. However, ANP is still evolving, as the concept of DID is still emerging for agents [15,42], leaving a research gap in authorisation, especially for multi-hop implementations.

2.1 Authorisation for Agentic AI Systems

Current authorisation and authentication mechanisms assume that the identities and permissions of the executing parties remain stable throughout the session. In this case, the authentication and authorisation mechanisms were often implemented through bearer tokens [32], such as OAuth [25], OpenIDC [32,41], and cookies [42], assuming the entity does not change the identity, scopes or context during the session [42,48].

With agentic AI systems, this approach is no longer appropriate, since agents are generally short-lived or can be spawned dynamically [42]. As a result, zero-trust architecture (ZTA) [42] emerged as a potential solution by ensuring that every request is continuously authenticated and authorised (“do not trust, verify continuously” [42]). However, the real-world implementation of ZTA still depends on some centralised parties, such as *identity providers* (IdPs) and *authorisation servers* (AS). This structure is efficient for human-to-machine interaction, but inadequate for machine-to-machine interaction, because the centralised parties

involved in this structure can introduce latency, a single point of failure, and other cross-boundary issues [40,42].

Existing work on access control for agentic AI systems focuses mainly on restricting the agent’s access to user’s sensitive information, without actually validating the user’s requirement. A study by Kim et al. [34] found that an organisation employing OAuth-based access controls – especially, Role Based Access Control (RBAC) – would need to restrict the agent scope. However, existing authorisation mechanisms such as OAuth, OpenID Connect, RBAC, XACML, etc. were introduced for humans; OAuth and OpenID Connect require user consent and manual client registration, rely on static and coarse-grained scopes, and do not allow tokens to be dynamically adapted according to context.

Several other strands of research have shifted the focus for existing systems to adapt to the decentralised mechanisms, such as DIDs, and Verifiable Credentials (VCs) [42,45,48], but existing systems were still built for humans, hence incorporating DIDs and VCs can be complex and expensive [29,37].

2.2 Limitations of OAuth

The OAuth framework was introduced to address authorisation challenges in traditional client-server architectures. OAuth defines separate roles for resource owners (users), clients (third-party applications acting on behalf of users), AS and RS [24]. By issuing access tokens to clients, OAuth allows applications to access protected resources without requiring clients to store or directly handle user credentials. When using OAuth, resource owners delegate part of their permissions to the clients to work on their behalf. It functions effectively when humans act as the executors. However, the autonomous behaviour of AI agents introduces security challenges that fall outside the scope of the existing framework [57]. Research has identified additional security risks in agentic AI systems that cannot be addressed with existing static frameworks [57]. For example, He et al. [26] highlighted vulnerabilities in AI agents that cannot be handled by the current framework, while Ren et al. [47] demonstrated how inter-domain agents coordinate and execute malicious tasks bypassing static controls.

In addition to the above limitations, OAuth also presents limitations with respect to agentic AI systems. OAuth flow being centralised around AS creates an issue if the agents want to communicate across domains, i.e., outside a trust boundary. Cross-domain implementations have been suggested, such as Dynamic Client Registration(DCR), but they are still theoretical and increase the overhead [15].

OAuth does not support a multi-hop delegation chain beyond two hops, e.g., user→orchestrator→agentA and so on. Due to this, RS only sees the partial chain, not the entire delegation chain to verify “who delegated the task to whom”, and “whether the agent is actually working on behalf of the user”. In other words, there is no verifiable, non-repudiable, auditable and traceable implementation for capturing delegation chains [15].

OAuth does not constrain agents’ depth, that is, the ability of an agent to spawn many sub-agents, which is an integral part of agentic AI systems. This is

Table 1. Baseline comparison between existing approaches and our approach

Approach	User Anchoring	Offline Attenuation	Param-Level Constraints	Revocation	Auditability
Macaroons [16]	Anchored to the root HMAC-secret holder, not a human approval approach	Holder appends caveats without contacting the issuer	Caveats are predicates restricting macaroon’s authority	Requires third-party caveats or short TTL	Relies on third-party caveats
Biscuit [20]	Root-key holder, not human-bound	Allows offline attenuation	Datalog checks present	Has native unique revocation IDs per token/attenuated chain	No agent-aware audit primitives [49]
SPKI/SDSI [53]	Key-to-key delegation, no task-level human binding	Possible via certificate-chain tuple reduction without contacting the root	Authorisation is a free-form S-expression; can express constraints but no structured schema	RFC 2693 treats Certificate Revocation List (CRL) as optional, not mandatory	Not specified in RFC 2693
Agentic JWT [24]	Binds action to verifiable user intent	Not specified	Relies on AS to get a least-privilege access token (optionally bound to an intent token)	Not specified	Chained delegation assertion and per-agent identity hash give partial provenance
HDP draft [27]	Proposed token signing by the human principal at issuance	Not specified as per IETF draft	Not specified as HDP tokens are audit input not enforcement	Proposes short-lived, session-bound tokens instead of a revocation registry	Offline-verifiable, signed hop-by-hop chain-of-custody
AAT draft [40]	Proposed as Root Issuer, not documented as a human-signature token (leaving it as a deployment responsibility)	Allows offline attenuation	Adds param-level constraints	Revocation of individual AATs or derived tokens is outside the scope of the draft	Cryptographic parent to child relation is verifiable, but no stated audit-log feature
Our approach	Root token signed by user on approval (restricting further delegation to user’s requirements)	Allows offline attenuation by constraining child token to have less permissions than root token	Allows param-level constraints by restricting token minting with user approved values	Supports short TTL tokens and allow revocation if any discrepancy is found	Creates signed logs at every verification step

a limitation in OAuth extensions being proposed that ensures max-delegation-depth [38]. Although OAuth is a widely accepted authorisation framework, as highlighted above, it cannot be directly applied to agentic AI systems without affecting agent’s potential capabilities. Moreover, the current research work relies on central authority [15,24] and leaves a gap for multi-hop flows [15]. Some recent IETF drafts for human-anchored delegation propose to solve the issue of over-permission by minting user-restricted tokens. However, the proposals leave the secure generation and verification of tokens to implementers [14,27,40].

To address agentic AI systems, existing authorisation frameworks are being extended. For example, OAuth is being extended to [28]: (i) introduce the delegated authorisation; (ii) limit the access to the protected resource; (iii) allow dynamic registration of the agents as clients; (iv) allow dynamic discovery of endpoints. However, existing frameworks cannot yet process: the actual context (i.e., they rely on the request made by the agent to process the token) [28], delegation chain beyond two parties [15], and decentralised authorisation and authentication of AI agents [15,42].

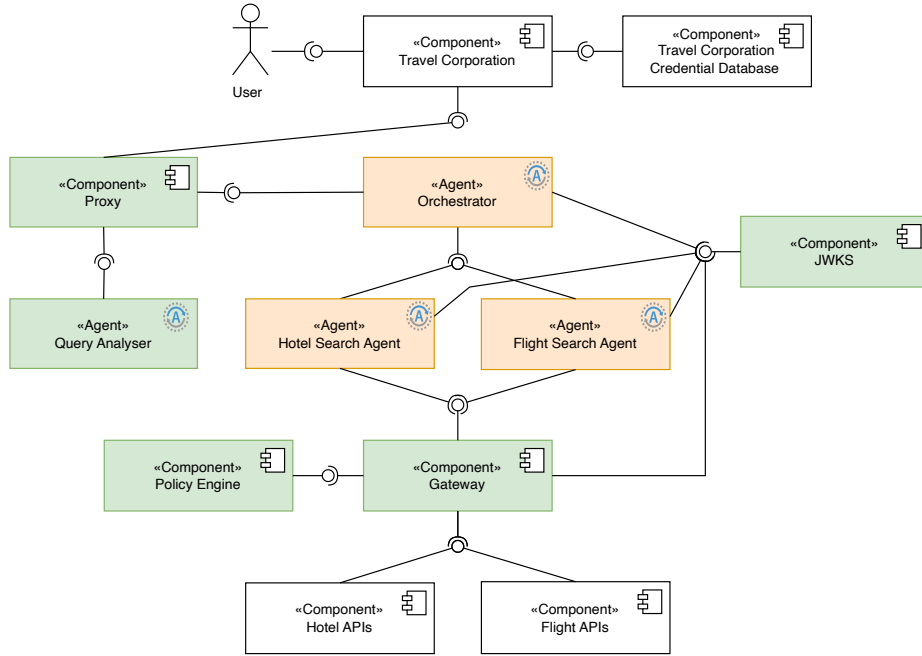


Fig. 1. Architecture diagram of authorisation in agentic AI flow of Travel Corporation application (green boxes highlight authorisation components, orange boxes highlight agents, while white boxes highlight authentication and resource server components)

These drawbacks are not merely theoretical weaknesses, but represent a significant research gap that necessitates a new approach capable of supporting dynamic and continuous authentication and authorisation of agents in agentic AI systems (especially for multi-hop implementations). Our approach aims to fill some of these gaps by proposing a method to securely authenticate, authorise and delegate the task/sub-task restricted to the user’s assigned task. The agent is delegated only the permissions, or a subset of permissions, necessary to fulfil the user’s task. Table 1 highlights the baseline capability comparison between existing works and our approach that proposes to strengthen the authorisation for agentic AI systems.

3 Case Study

To highlight the key differences between human-to-machine and machine-to-machine interactions, we picked an example of a travel booking system as a case study, and we call it the “Travel Corporation” system [34,54]. The Travel Corporation (as shown in Figure 1) has an application that helps users search and book flights and hotels for various destinations. The application provides data about various flights and hotels depending on the users’ requirements.

The white boxes in Figure 1 represent the non-agentic AI components of the Travel Corporation application. The application first authenticates the user and then allows the user to manually search, filter and book the required flights and hotels. The application displays the various hotel and flight results fetched from the Hotel and Flight APIs, using which the user can manually select the required information for booking. The application requires a lot of human involvement to perform tasks, while (non-agentic) AI agents can work as chatbots or assistants [48]. However, with advanced autonomy, agents can execute an end-to-end workflow on behalf of the user, reducing the number of tasks that the user has to perform manually.

With agentic AI, the above tasks can be performed by agents, requiring the user only to submit a natural-language query to the application. The orange boxes in Figure 1 highlight the agentic AI flow of the system. The authenticated user submits the task in a natural language, and the orchestrator agent reasons about the task and delegates the task/sub-tasks further to the respective sub-agents. However, autonomy can introduce security issues. For example, the agent may be tricked by an attacker to follow the latter’s instruction, which can result in scope escalation, goal drift, wrong tool usage, etc. Therefore, a security solution is required that can record the delegation chain from user to sub-agent, along with the user’s consent for a task. Section 5 and the green boxes in Figure 1 introduce our proposed solution for a secure agentic AI flow.

3.1 Threat Model

We assume that attackers mainly want to escalate beyond their delegated authority (i.e., to act beyond what is authorised by the user), and/or to replay or forge tokens. For attackers to achieve these goals, we also assume that they are capable of: (i) controlling the sub-agent (e.g., flight search agent); (ii) controlling the orchestrator agent; (iii) monitoring the traffic; (iv) expanding the scope, permissions, and delegated values; and (v) tampering with the delegation chain from user to sub-agent. As such, this paper identifies the following ten threats:

1. **Scope escalation:** The sub-agent expands the scope beyond what was delegated to it initially. For example, a flight search agent (sub-agent) may maliciously expand the scope to execute when the sub-agent was only delegated a scope as read by the orchestrator agent.
2. **Token replay:** The attacker captures a valid token and reuses it for other (potentially malicious) tasks.
3. **Using expired token:** The attacker or malicious agent/sub-agent tries to access the RS using an expired token.
4. **Executing high-risk operation without human oversight:** The sub-agent tries to execute a high-risk task (e.g., payment) without user approval.
5. **Forged user authority:** The attacker tries to delegate tasks to the orchestrator agent using an authenticated username, e.g., Eve (an attacker) tries to delegate a task to the orchestrator agent using Bob’s username.
6. **Sub-agent uses revoked token:** The user changed the task, revoked the permissions, but the sub-agent requests access using the revoked permission.

7. **Sub-agent uses wrong tool:** The sub-agent uses a different tool than what was delegated to it by the orchestrator agent, e.g., a flight search agent uses the `search_hotels` tool instead of the `search_flight` tool for the flight search sub-task.
8. **Params value violated:** The sub-agent requests access to a value different from what was approved by the user. For example, the sub-agent requests results for `max_price = £1,000`, when it was only authorised for `£300`.
9. **Delegated unapproved sub-task:** The malicious orchestrator agent creates a sub-task from the task that the user never approved.
10. **The sub-agent uses different “*delegation-chain-token*”:** The sub-agent (such as a flight agent) tries to use a different “*delegation-chain-token*” for minting an “*execution-token*” (e.g., the flight agent tries to create an “*execution-token*” using the “*delegation-chain-token*” that is created for the hotel agent for accessing hotel information). Note: this threat is pertinent to our approach, since the threat considers a sub-agent in a multi-hop system, which – under an attacker’s influence – would request information from an RS for a sub-task that was not delegated to the sub-agent.

4 Methodology

This section outlines the specific prompts used, the selection of models, the data collection process, and the evaluation methodology used to assess the importance of fine-grained, continuous verification for agents in agentic AI systems.

To build the prototype and evaluate our approach (outlined later in Section 5), we selected “qwen3.5:35b” model [3] running on “NVIDIA A100 80GB PCIe”, “Streamlit” [4] for frontend and “AutoGen” [1] for building agents. The system is implemented in Python 3. We selected the “qwen3.5:35b” model for two main reasons: (i) to reduce the token consumption pricing, and (ii) to make resource constraint devices (such as edge devices) compatible with our approach.

To evaluate the system, we used approximately 20 different prompts, since the selected prompts cover the requirements that users commonly provide to AI systems [8,9]. The prompts were grouped into “simple-direct” and “complex-indirect” prompts. For instance, a simple-direct prompt could be “I want flight booking from London to Glasgow for £380”, while complex-indirect prompts involved specifying the requirements indirectly to the system such as “Plan me a relaxing vacation from London to Glasgow for £2747 with economy class with 4 days stay”. In complex-indirect prompts, we did not specify separate budgets for hotel and flight booking. The decision to divide the budget for booking a hotel and flight was left to the “Query Analyser” agent. In case of replanning, we just changed the origin or destination in some prompts, while for other prompts we changed the budget, hotel stay days, and flight preferences (from economy to business or vice versa).

For agents to fetch the hotel and flight details, we used Model Context Protocol (MCP) [6]. MCP is a lightweight protocol that bridges LLMs (such as GPT-4o) to external data sources and tools, acting as a standardised adapter

to allow LLMs to search files, query databases, and control applications without needing custom, vendor-specific API. For our implementation here, we used a Python-based MCP library, along with synthetic data, which was created manually for the tools after observing the actual data from various websites.

Building on the threat model presented in Section 3.1, we highlight our approach here for evaluating threats using the recent OWASP’s “Agentic Security Initiative” [7,10,11] and Anthropic’s “Zero Trust for AI Agents” documentation [12], which mention different exploits that happen due to weak governance or missing user approval for the task. The OWASP highlights the latest top 10 highest-impact threats [7], however our proposed approach covers eight threats (Agent Goal Hijack, Tool Misuse and Exploitation, Identity and Privilege Abuse, Insecure Inter-Agent Communication, Cascading Failures, Human-Agent Trust Exploitation, Rogue Agents, and partially Agentic Supply Chain Vulnerabilities). The remaining two threats (Memory & Context Poisoning, and Unexpected Code Execution (RCE)) are currently out of scope for the paper. We evaluate whether the resource requested by the agent or sub-agent (in a multi-hop environment) aligns with the user’s requirement, and the task is approved and verified by the user before the agent requests the resource from the RS. These issues are highlighted by OWASP and Anthropic as some of the reasons for current threats in their documentations.

5 Proposed Solution

AI agents are used on everyday tasks to execute a wide range of system flows, such as sending emails, searching information, calling APIs, and initiating payments on behalf of users [27]. When a user authorises an agent, which can further delegate the task to other sub-agents, the user’s authorisation becomes disconnected from the action, i.e., there remains no proof or record of the authorisation chain. This creates accountability, auditability and safety issues [27]. Existing agentic AI systems authenticate devices, network endpoints, and other system entities, but they are unable to capture the human user who initiated the authorisation or delegation chain, making it difficult to determine the rationale behind an agent’s behaviour [14]. Moreover, agents often operate across multiple platforms using user credentials, thereby increasing the risk of impersonation attacks. To address these issues, our solution assigns a unique identity to each agent, constrains agent actions to those explicitly authorised by the user, and captures the complete delegation chain from the user to all agents and sub-agents.

5.1 Architecture of the Proposed Solution

Figure 1 shows the architecture of the proposed authorisation solution for agentic AI flows, which introduces an approach in which an authenticated human user signs a root token using their own keys. An agent (represented as orange boxes in Figure 1) signs the delegation or execution tokens (minted based on the root

Algorithm 1 User Approved Delegation Chain

Input: Takes the user query in natural language after the user is successfully logged in
Output: Displays results based on user query

```

1: subtasks  $\leftarrow$  QueryAnalyzer(user_query)
2: root-delegation-token  $\leftarrow$  Sign(user_priv_key, {claims and subtasks to create the root-delegation-token})
3: Verify root-delegation-token and user signature on the root-delegation-token
4: for each  $s \in$  subtasks do agent  $\leftarrow$  DiscoverAgent( $s$ ) AND delegation-chain-token  $\leftarrow$  orchestrator/parent agent create signed delegation-chain-token for delegating sub-task to a sub-agent
5:   if  $\neg$  AcceptDelegation(deleg, root, agent) then return Deny
6:   end if
7:   if  $s.risk = high$  then WaitForUserApproval( $s$ )
8:   end if
9:   execution-token  $\leftarrow$  sub-agent create signed execution-token for executing a sub-task
10:  Evaluate execution-token, delegation-chain-token and root-delegation-token and get decision with the reason
11:  if decision  $\neq$  Allow then RevokeExecJTI(exec.jti) return Deny(reason)
12:  end if
13:  result[ $s$ ]  $\leftarrow$  MCPServer[ $s.mcp\_url$ ].call( $s.tool$ , exec.params)
14: end for
15: return result

```

token) using their respective keys. The tokens and delegation chain depict the constraints under which an agent is authorised to act on the user’s behalf.

Continuing with the travel corporation case study, the user logs into the application and sends a query, which is then fetched by a trusted proxy that forwards it to an internal trusted agent “Query Analyser”. The query analyser has access to an LLM and the list of available tools. Based on the user query, the query analyser generates the sub-tasks, a short task description, and the maximum delegation depth (as shown by line 1 of Algorithm 1). The generated parameters are then forwarded to a trusted proxy, which later adds the JWT header, remaining claims in the payload (such as issue time, expiry time, etc.), and verifies the values generated by the query analyser.

For verifying the claims in the sub-tasks, the system uses the GLiNER2 model [50,56]. The GLiNER2 model was chosen because it satisfies our requirements to reliably extract entities and the respective values without hallucinating new entity types or values that do not exist [55]. Additionally, GLiNER2 runs on CPU, avoiding the GPU dependency of LLM-based models such as GPT-4o, Claude 3.5 Sonnet, and Claude Opus [55].

The model fully runs within a trusted proxy, which is important because the user’s query becomes the basis for a cryptographically signed delegation chain, ruling out the need to depend on LLM-based models. Finally, the GLiNER2 extracts the same entity values for the same query, whereas LLM-based models rely on token sampling and can return different results each time the same query is given to them. Alternatives such as spaCy require per-entity fine-tuning [2,36], DeBERTa zero-shot increases CPU latency as labels increase, e.g., 6.8 times slower than GLiNER2 at 20 labels [55], and the original GLiNER [55] would require three separate models to cover the Named Entity Recognition (NER), classification, and extraction tasks that GLiNER2 handles together [55].

Appendix 1 explains the key concepts used in this solution, including the terminology regarding actors (users and agents), as well as other components

used. The rest of this section outlines the token structure being used in our proposed solution:

1. *“root-delegation-token”*: represents the human who actually authorised the actions and delegated the task. The JWT highlights the authorised/approved actions, the agent to whom the user has delegated the task to, and the approved tools the agent is allowed to use. The *“root-delegation-token”* contains the 15 items of payload listed in Appendix 2. The `approved_subtasks` is a dictionary that stores key-value pairs such as `sub_task id`, `tool` (required for sub-task), `mcp_server_url` (MCP server that exposes the tool), `scope` (specific for sub-task), `risk` (specific for sub-task, such as low, medium, high), `human_oversight` (human review required or not, for instance through “human-in-the-loop” verification), `description` (short description of specific sub-task), and `params` (specific to the sub-task). In our approach, `iss` in *“root-delegation-token”* represents the application URL (i.e. application domain) that holds the user’s public keys.
2. *“delegation-chain-token”*: is used when an agent wants to further delegate a task to other sub-agents, such as when the orchestrator agent further delegates a sub-task to a flight agent. This JWT highlights the agent who delegated the sub-task, to whom the sub-task was delegated to, the approved actions (which are subsets of *“root-delegation-token”*), and the parent *“root-delegation-token”*. The token is signed by the agent delegating the sub-task to another agent, i.e., signed by the parent agent. This token contains 21 claims, including `jti`, `token_type`, `params`, `iss`, and `human_oversight`. In our approach, `params` in the *“delegation-chain-token”* captures the `params` claim mentioned in the *“root-delegation-token”* under approved subtasks, and `iss` represents the developer company’s URL (i.e. developer-domain) that holds the agent’s public keys (for instance, orchestrator agent’s public keys).
3. *“execution-token”*: The token is created and signed by a sub-agent that needs to connect to an RS. The *“execution-token”* is minted based on the *“root-delegation-token”* and the *“delegation-chain-token”*. It records useful information such as the tool the agent is using (a subset of approved tools), and the actions the agent will perform (a subset of approved actions). The *“execution-token”* contains the following claims: `jti`, `token_type`, `iss`, `sub`, `sub-task`, `aud`, `scope`, `risk`, `human_oversight`, `agent_id`, `purpose`, `tool`, `delegation_jti`, `root_jti`, `iat`, `exp`, `params`, `mcp_server_url`.

This token structure helps us to understand how an agent behaves, and under whose authority. Together, these tokens provide the structure and foundation for accountability in agentic AI systems, especially in multi-hop implementations.

5.2 Implementation

Each agent is assigned an agent card containing information such as agent name, skills, capabilities, developer organisation name, endpoint URL, description, input and output schema, scopes and developer signature. The developer’s signature on the agent card prevents the card from being tampered with. The

corresponding public keys are published as `https://<developer-domain>/.well-known/developer/jwks.json`. Once the card is signed, it is published at a `/.well-known` URI location for discovery (`/.well-known/agent-card.json`).

Each agent and user are assigned a public and private key pair. The private key is stored in a secure location such as Key Management System (KMS) or hardware-backed locations (e.g., passkeys for users that ensure keys remain secure with the user [22,46]), while the public key for the agents is published via `https://<developer-domain>/.well-known/jwks/agents`. Users' public keys are published via `https://<application-domain>/.well-known/jwks/users/<username>`. The users are assigned the key pair when they first create an account with the application. The key rotation occurs only if the user changes the credentials or reports any key leaks.

In the context of the case study introduced in Section 3, let us consider the same “complex-indirect” prompt as an example query sent by the user. The GLiNER2 model verifies that the **origin**, **destination**, **nights**, **price-per-night**, etc., are the same as requested by the user, and the **scope**, **risk**, **tool**, and **MCP server URL** are valid. For example, the **MCP server URL** is not hallucinated, **tool** fetched is present at the MCP server, **scope** is not low for high-risk task, and the **params** claims generated have key and value pairs as required by the tool's schema – such as **origin:London**, **destination:Glasgow**, **max_price:£300**, **travel_class:economy** [6].

After verifying and generating the “*root-delegation-token*”, the JWT is displayed to the user for approval or rejection. Once the user approves and signs the “*root-delegation-token*”, the approval is logged, and the token is subsequently forwarded by the proxy to the orchestrator agent (shown by line 2 in Algorithm 1 and Appendix 2).

After receiving the JWT, the orchestrator agent first verifies the token using the user's public keys exposed by the JWKS endpoint. Next, the orchestrator agent verifies that the **delegated-to** and **delegated-to-agent-id** claim's values match respective values present in its agent card, **current_delegation_depth** < **max_delegation_depth**, and the token has not yet expired. Once verified, the orchestrator creates a signed log that records the verification status. The orchestrator then discovers the agents based on the approved subtasks (Algorithm 1, lines 3–4). The orchestrator first verifies the developer's signature on the agent cards, and then it creates the “*delegation-chain-token*” for the selected agents (Algorithm 1, line 4). The created token is then forwarded to the discovered agents along with the “*root-delegation-token*”.

When the sub-agent (such as a flight agent) receives a “*delegation-chain-token*”, the sub-agent first verifies the signatures on the “*delegation-chain-token*” and “*root-delegation-token*” and then verifies the agent card of the orchestrator agent. If verified, the agent then verifies every claims and their respective values present in the “*delegation-chain-token*” against the “*root-delegation-token*”, i.e., the values should be a subset of or equal to or less than “*root-delegation-token*” (Algorithm 1, lines 5–8). If verified, the agent first creates a signing log and then creates the signed “*execution-token*” (Algorithm 1, line 9).

Table 2. Threats validated and stopped by the proposed approach (Legend: **A** = Agent, **G** = Gateway, **AS** = Authorisation Server, **RS** = Resource Server, **●**= full, **○**= no, **◐**= partial)

Threat	Stopped without the proposed approach	Location (A/G) where the threat is stopped using the proposed approach, and how
1	◐ AS can grant broad scopes in access tokens	● G (using policy engine) : denies the request to access the tool, if the scope is mismatched with what was initially authorised by the user.
2	◐ AS stops replay, but the check is implemented for single hop (with drafts and proposals coming up for agents and multi-hops)	● A : when the sub-agent receives a <i>delegation-chain-token</i> , the sub-agent verifies the token ID against token IDs seen before and rejects any token that is replayed. G (using policy engine) : verifies token IDs of all received tokens against token IDs seen before and rejects any replayed tokens.
3	◐ AS/RS stop request made using expired tokens, but they are implemented for single hop (with drafts and proposals coming up for agents)	● A : when orchestrator agent and sub-agent receive <i>root-delegation-token</i> and <i>delegation-chain-token</i> , the agents verify issue/expiry time against current time and reject expired tokens. G (using policy engine) : verifies token IDs of all received tokens against token IDs seen before.
4	○ RS does not verify human oversight (with drafts and proposals coming up for agents)	● G (using policy engine) : denies access to the sub-agent if the sub-task requires human oversight and the sub-agent requests access to the RS without human approval.
5	◐ RS verifies tokens against AS's public keys and client's public keys using Demonstrating Proof-of-Possession (DPOP) (with drafts and proposals coming up for user signed tokens)	◐ A : When the orchestrator agent and sub-agent receive a <i>root-delegation-token</i> , the agents verify the signature using the public key hosted using the JWKS endpoint and reject the token if the signature is not verified. G (using policy engine) : verifies the user's signature on the token using the public key and rejects the token if the signature is not verified.
6	◐ AS prevents use of revoked tokens until single hop (with drafts and proposals coming up for agents)	● G (using policy engine) : verifies the complete chain and verifies whether the <i>root-delegation-token</i> is revoked or not by the user (based on which the <i>execution-token</i> is minted) and rejects any request that uses a revoked token.
7	◐ RS verifies the access token's aud claim and provides access if valid, but RS does not verify if the user approved its use for a specific task	● G (using policy engine) : verifies each claim of the <i>execution-token</i> (minted by the agent) with the <i>root-delegation-token</i> (approved and signed by the user) and rejects any request that uses a tool not approved by the user for a particular sub-task.
8	◐ AS/RS grants access based on request (comparing the requested value against pre-defined boundaries); does not verify what is requested by the user	● G (using policy engine) : verifies the requested value from the sub-agent in <i>execution-token</i> with what was delegated by the orchestrator agent and what was initially approved by the user in <i>root-delegation-token</i> ; rejects the request if the value is greater than what was approved by the user.
9	○ AS/RS does not verify if the requested sub-task, and tools are the same as what the user needs (with drafts and proposals coming up for agents)	● A : when the sub-agent receives <i>delegation-chain-token</i> and <i>root-delegation-token</i> , the sub-agent verifies both tokens and rejects tokens with unapproved values. G (using policy engine) : verifies <i>execution-token</i> and <i>delegation-chain-token</i> with the user-approved <i>root-delegation-token</i> and rejects the tokens that have unapproved values.
10	○ AS/RS does not verify delegation chains (drafts and proposals are coming up for agents) from user to agent	● G (using policy engine) : verifies claims in <i>execution-token</i> with claims in <i>delegation-chain-token</i> and <i>root-delegation-token</i> , such as <code>agent_id</code> , <code>delegated_to</code> , <code>delegation_jti</code> and <code>root_jti</code> .

The “*execution-token*”, along with the “*delegation-chain-token*” and the “*root-delegation-token*”, is then forwarded to the MCP server (when the tool is called), as shown by line 10 in Algorithm 1. The gateway present at the MCP server first verifies the signature on each token, then verifies the agent card, and verifies that the tokens are not revoked/expired/replayed, and finally verifies the claims and `params` in the “*execution-token*” and “*delegation-chain-token*” against the “*root-delegation-token*” using the policy engine. If everything is verified, the agent is then allowed access to the RS, based on the user’s authorised claims (Algorithm 1, line 10–15 and Appendix 2).

5.3 Validation

The system is evaluated against the ten threats mentioned in Section 3.1. The approach uses *updated proxy and gateway*, since traditional API gateways and proxies were not built for agentic protocols and recording information about the context [13,44]. Our approach evaluates whether the request to access a resource at RS aligns with user requirements and approval. Current implementations authorise access using OAuth access tokens, without considering whether the user agrees to the access or not (e.g., if a user asked for a London to Glasgow trip, but an attacker made an agent request a trip from London to Paris, current implementations would allow that). Thus, our approach needs to verify the chain and constraints from user to sub-agent. To evaluate and simulate the ten threats, the JWTs were crafted with values other than the ones approved by the user.

Scope escalation: The sub-agent mints an “*execution-token*” with `scope: execute` using the “*delegation-chain-token*” granting only `scope: read`. **Token replay:** The orchestrator and flight agent replay an old unexpired token. In the first case, the orchestrator agent replays an old token to flight agent, and in the second case, the flight agent replays an old token for requesting access to a resource. **Using expired token:** The orchestrator and flight agent use an expired token. In the first case, the orchestrator agent delegates a sub-task using an expired “*delegation-chain-token*” to the flight agent, and in the second case, the flight agent request access to a resource using an expired “*execution-token*”. **Executing high-risk operation without human oversight:** The sub-agent (flight agent) mints an “*execution-token*” without user approval for a high-risk task. **Forged user authority:** The attacker tries to create a “*root-delegation-token*” using their own keys. For example, in this attack, an attacker *Eve* attempts to sign a “*root-delegation-token*” minted by *Bob* using its own attacker’s private key. The threat does not mitigate an attack in which Bob’s private keys are stolen (out of scope for the paper). **Sub-agent uses revoked token:** The user revoked “*root-delegation-token*”, but the sub-agent (flight agent) requests access using the revoked token. **Sub-agent uses wrong tool:** The “*delegation-chain-token*” allows the flight agent to use the `search_flight` tool, but the flight agent mints the “*execution-token*” for the `search_hotels` tool using the same delegation chain. **Params value violated:** The sub-agent (such as hotel-search agent) requests results with `max_price = £1,000` when the user approved `max_price=£300`. **Delegated unapproved sub-task:** The malicious orchestrator agent creates a “*delegation-chain-token*” with a sub-task, such as find-flights, but delegates `tools`, `params` etc. that the user never approved. **The sub-agent uses a different delegation-chain-token:** In this attack, the flight agent tries to create an “*execution-token*” using the “*delegation-chain-token*” that is created for the hotel-search agent for accessing hotel information.

With the proposed approach, all the ten threats were caught before or at the gateway and access was denied before the agent could access the RS. Table 2 summarises the location and the checks that stop threats using our approach. For example, “Threat 1” is mitigated because the gateway, using the policy engine, verifies whether the `scope` in the “*execution-token*” is a subset of the `scope`

mentioned in the “*delegation-chain-token*” (subset check on actions). Similarly “Threat 2” is caught and mitigated since every token is assigned a unique ID, which is stored after verification. The gateway rejects any JWT seen before.

6 Discussion

The paper highlights the importance of continuous trust verification and proof of delegated permissions from the user to the agent before the agent requests access to the resource on the resource server.

Implications of this research The proposed approach provides a solution for tracing and verifying trust at each step before the agent accesses the resource server on the user’s behalf. The solution can be used in addition to existing protocols. In the event of a cyber incident, the proposed approach enables a reviewer to reconstruct the entire delegation chain. Furthermore, when an agent is granted an OAuth access token, which may provide broad access, the reviewer can verify that the permissions ultimately exercised remain consistent with the user’s original authorisation. This is achieved when the agent exchanges the access token for a narrower token using OAuth extensions such as Token Exchange, Rich Authorization Requests (RAR) [17], or transaction tokens.

The solution helps solve some of the problems that systems are currently facing, including user consent. Some of the cyber incidents reported below are examples of such gaps: (i) “It took nine seconds” - Claude AI agent deletes company’s entire database [21]; (ii) The Vercel Breach - OAuth Supply Chain Attack Exposes the Hidden Risk in Platform Environment Variables [19]; (iii) Crypto Theft by AI agent - Grok agent decoded the Morse code into plain English and treated the instruction as a valid transaction [39]. However, our solution will incur additional costs, such as increased latency (in terms of token creation, signing, and verification at multi-hops) and additional components (such as proxy, gateway, policy engine).

For instance in case of the “simple-direct” prompts the “Query analyser” reported approximately 83415.68 ms latency (the latency also includes the time the agent took to understand the user query in natural language, i.e., depends on how long and clear the user query is), GLiNER2 model took 9387.72 ms, sub-agents reported 2 to 5 ms for token creation and signing and 4 to 6 ms for token verification. The end-to-end flow latency (root token→orchestrator agent→sub-agent→gateway (using policy engine)) reported was 108797.42 ms to 154568.11 ms. For “complex-indirect” prompts the “Query analyser” reported approximately 95071.46 ms to 121823.13 ms latency depending on the length and clarity of user query, GLiNER2 model took 10723.32 ms to 12617.30 ms, sub-agents reported 3 to 6 ms for token creation and signing and 4 to 8 ms for token verification. The end-to-end flow latency reported was 174568.2 ms to 232244.46 ms. However, our prototype is deployed within our environment and the measured latencies does not include network traffic and factors such as

complex user queries. In a larger and distributed deployment, the total end-to-end latency can depend on various other factors such as heavy network traffic, handling other sessions etc.

Limitations The paper currently uses the GLiNER2 model to verify claims generated by the query analyser agent, which verifies the claims based on the defined entities. Since the user’s query can be complex and include various requirements, therefore, there can be scenarios where GLiNER2 model may be unable to capture all the requirements. In addition, the paper currently authenticates the agents using signed agent cards published at a well-known URI, however, there can be possible implementations where agents without having agent cards can be spawned at runtime for a particular sub-task. Additionally, our approach currently relies on manually written policies, which require policies to be written beforehand. Finally, currently, it is not possible to handle dynamic policy updates while the agent is processing the query. For instance, if certain permissions are invalidated while policies are updated, the agents will not be able to fulfil the new permission, and certain checks will need to be introduced after the required data is fetched.

7 Conclusion and Future Work

Agentic AI represents a shift from a conversational chatbot to an AI system consisting of agents that can work autonomously. The paper proposed an approach that can help improve security by moving beyond the concept of statically assigning pre-defined permissions. The paper further introduces the concept of continuous trust verification, based on the actions and behaviour approved and delegated by the user. The proposed approach is not intended to replace existing protocols or frameworks such as MCP, A2A, and OAuth. It is proposed to provide a solution to tackle the current issue of accountability and delegation for agents. Using the proposed approach, a reviewer, such as a security analyst, can determine which action was delegated by the user, who authorised the action, what was initially approved by the user, which agent received the delegation, any subsequent delegations to other agents, the tools, scopes, and permissions propagated through the chain in the case of multi-hop delegations. In the proposed approach, the MCP server does not require OAuth-based access tokens, unlike the Gmail or GitLab MCP servers (which use OAuth-based access tokens to secure the user’s personal and sensitive information). However, the paper currently uses the GLiNER2 model to verify the claims generated by the agent.

Future work will focus on exploring ways to introduce methods that can handle complex user queries and scopes with precision. While the approach highlights a way to record the delegation from user to agent (even in a multi-hop scenario), future research will focus on applying the approach for ephemeral agents that can be spawned for a particular sub-task without having agent cards. Furthermore, research is required to create automatic policies that can verify and authorise an agent based on changing requirements.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Appendix 1 Glossary of Key Concepts

The proposed solution introduces the following key concepts on actors (users and agents) and other components for securing the multi-hop agentic AI system:

- *User*: refers to the human actor who gives a task in natural language.
- *Query analyser agent*: receives the task from the proxy and creates “*root-delegation-token*”.
- *Orchestrator agent*: receives the user-signed “*root-delegation-token*” via the proxy, and creates “*delegation-chain-token*” for each sub-agent after discovering the sub-agents, based on the task and approved sub-tasks.
- *Sub-agents* (such as Flight Search Agent): receive both “*root-delegation-token*” and “*delegation-chain-token*”, and create an “*execution-token*” for executing the sub-task that requires access to a tool and the MCP server.
- *Proxy*: is a trusted component between the user, query analyser and orchestrator agent. It passes the task given by the user to the query analyser agent, verifies the claims and their respective values generated by the query analyser agent using GLiNER2 model and passes the user-signed “*root-delegation-token*” to the orchestrator agent.
- *JWKS (JSON Web Key Set)*: is a JSON data structure that contains public keys for verifying JSON Web Tokens (JWTs).
- *Policy Engine*: returns “allow” or “deny” decision to access the MCP server.
- *Gateway*: is a trusted component that verifies the tokens and the complete chain from the user to the sub-agents and allows/denies the agent access based on the decision given by the policy engine.
- *MCP Servers and Tools* (such as Hotel APIs, Flight APIs): are the components that host the different tools to help agents perform various actions, such as fetching information from the resource server.

Appendix 2 Structure of “root-delegation-token”

Below is an example structure of the “*root-delegation-token*”. This token outlines the sub-tasks and the permission request(s) for further delegation(s), which are displayed to the user, who will then approve or deny the delegation(s).

JWT Header: alg : RS256 typ : JWT kid : user-alice-7b1dc2	sub : alice aud : Orchestrator delegated_to : Orchestrator delegated_to_agent_id : agent-orchestrator-0a4b20 iat : 1778152667 (~2026-05-07 16:47:46) exp : 1778154167 (~2026-05-07 17:12:46) task_description : Plan a relaxing vacation from London to Glasgow for 2747 pounds with economy class and a 4-day stay. scopes : read, execute, reasoning total_budget : 2747.0
JWT Payload: jti : 32438fe4-c5e8-46e8-826a-255cea156bab token_type : root-delegation-token iss : test-corp.com	

```

max_delegation_depth : 5
current_delegation_depth : 0

Approved Subtasks (4):
1. find-flights - tool : search_flights -
    mcp_server_url : mcp.flights.internal -
    scope : read - risk : low -
    human_oversight : False -
    description : Search for economy flights
        from London to Glasgow within budget -
    params : - origin : london -
        destination : glasgow - max_price : 300 -
        travel_class : economy
2. find-hotels - tool : search_hotels -
    mcp_server_url : mcp.hotels.internal -
    scope : read - risk : low -
    human_oversight : False -
    description : Search for hotels in
        Glasgow for 3 nights within budget -
    params : - area : glasgow - nights : 3 -
        max_price_per_night : 815

```

References

1. AutoGen - A framework for building AI agents and applications. <https://microsoft.github.io/autogen/stable/index.html>
2. Linguistic Features. <https://spacy.io/usage/linguistic-features>
3. qwen3.5. <https://ollama.com/library/qwen3.5>
4. Streamlit documentation. <https://docs.streamlit.io/>
5. What is OAuth? <https://www.microsoft.com/en-us/security/business/security-101/what-is-oauth/>
6. What is the Model Context Protocol (MCP)? <https://modelcontextprotocol.io/docs/getting-started/intro>
7. OWASP Top 10 for Agentic Applications for 2026. <https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/> (2025)
8. 12 AI Prompts for Travel Planning (Group Travel and Solo Travel). <https://www.joinmytrip.com/blog/en/ai-prompts-for-travel/> (2026)
9. 30+ Best AI Prompts for Cheap Flights. <https://insidea.com/blog/ai/prompts/best-ai-prompts-for-cheap-flights> (2026)
10. AIUC-1: Crosswalks OWASP Top 10 For Agentic Applications. <https://genai.owasp.org/resource/aiuc-1-crosswalks-owasp-top-10-for-agentic-applications/> (2026)
11. State of Agentic AI Security and Governance 2.01. <https://genai.owasp.org/resource/state-of-agentic-ai-security-and-governance/> (2026)
12. Zero Trust for AI agents. <https://claude.com/blog/zero-trust-for-ai-agents> (2026)
13. A Linux Foundation Project: agentgateway. <https://agentgateway.dev/> (2026)
14. Beyer, B.W.: Architecture for Human-Anchored Agent Identity, Delegation, and Provenance. <https://datatracker.ietf.org/doc/draft-beyer-agent-identity-architecture/> (2026)
15. Bhushan, B., Pappu, K., Mittal, A.: A Conceptual Framework for Authentication in Agentic AI Ecosystems: Protocol Analysis and Taxonomy. In: 2025 International Conference on Computer and Applications (ICCA). pp. 1–8 (2025)
16. Birgisson, A., Politz, J.G., Erlingsson, U., et al.: Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In: NDSS (2014)
17. Cecchetti, S.: Cedar Profile for OAuth 2.0 Rich Authorization Requests. Internet-Draft draft-cecchetti-oauth-rar-cedar-02, IETF (Feb 2024), <https://datatracker.ietf.org/doc/html/draft-cecchetti-oauth-rar-cedar-02>, Work in Progress
18. Chang, G., Lin, E., Yuan, C., et al.: Agent network protocol technical white paper. arXiv:2508.00007 (2025)
19. Columbus, L.: Vercel breach exposes the OAuth gap most security teams cannot detect, scope or contain. <https://venturebeat.com/security/vercel-breach-exposes-the-oauth-gap-most-security-teams-cannot-detect-scope-or-contain> (Apr 21, 2026)

20. Couprie, G., et al.: Biscuit, a bearer token with offline attenuation & decentralized verification. <https://doc.biscuitsec.org/reference/specifications.html>
21. Cuthbertson, A.: It took nine seconds': Claude AI agent deletes company's entire database. <https://www.independent.co.uk/tech/claude-ai-agent-deletes-startup-anthropic-b2966176.html> (28 April 2026)
22. Deshpande, V., Khade, A.V., J, H., Addanki, G.K., Deenadayalan, S.: A Passkey Based Recovery Mechanism for Blockchain Hardware Wallets. In: 2024 IEEE/ACM 17th Int'l Conf. on Utility and Cloud Computing (UCC). pp. 276–281 (2024)
23. Ehtesham, A., Singh, A., et al.: A survey of agent interoperability protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP). arXiv:2505.02279 (2025)
24. Goswami, A.: Agentic JWT: A Secure Delegation Protocol for Autonomous AI Agents. arXiv:2509.13597 (2025)
25. Hardt, D.: The OAuth 2.0 Authorization Framework. <https://datatracker.ietf.org/doc/html/rfc6749> (2012)
26. He, Y., Wang, E., Rong, Y., et al.: Security of AI Agents. In: 2025 IEEE/ACM International Workshop on Responsible AI Engineering (RAIE). pp. 45–52 (2025)
27. Helixar: Human Delegation Provenance Protocol (HDP): Cryptographic Chain-of-Custody for Agentic AI Systems. <https://datatracker.ietf.org/doc/draft-helixar-hdp-agentic-delegation/> (2026)
28. Helou, M.E., Troiani, C., Ryder, B., et al.: Delegated Authorization for Agents Constrained to Semantic Task-to-Scope Matching. arXiv:2510.26702 (2025)
29. Huang, K., Narajala, V.S., et al.: A Novel Zero-Trust Identity Framework for Agentic AI: Decentralized Authentication and Fine-Grained Access Control. In: 2026 Int'l Conf. on AI x Data and Knowledge Eng. (AIDKE). pp. 98–101. IEEE (2026)
30. Jones, M.: JSON Web Key (JWK). <https://datatracker.ietf.org/doc/html/rfc7517> (2015)
31. Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT). <https://datatracker.ietf.org/doc/html/rfc7519> (2015)
32. Jones, M., Hardt, D.: The OAuth 2.0 Authorization Framework: Bearer Token Usage. <https://datatracker.ietf.org/doc/html/rfc6750> (2012)
33. Khalid, O., Farooqi, A.U.H., Bilal, M.: Agentic AI: a review, applications and open research challenges. *Computing* **108**(6), 80 (2026)
34. Kim, S., Kim, H.: A Case Study on Delegating Critical Tasks to Agentic AI and Prototype Access Control Methods. In: 2025 Cyber Awareness and Research Symposium (CARS). pp. 1–5 (2025)
35. Krishnan, N.K.: Beyond Context Sharing: A Unified Agent Communication Protocol (ACP) for Secure, Federated, and Autonomous Agent-to-Agent (A2A) Orchestration. arXiv:2602.15055 (2026)
36. Kumar, M., Singh, P., Kashtriya, P.: Enhanced biomedical named entity recognition using SpaCy and BERT models. *Procedia Comp. Sci.* **258**, 1954–1961 (2025)
37. Kumarasamy, S.: Verifiable Credentials: A Deep Dive for the Agentic AI Era. <https://shankarkumarasamy.blog/2025/02/28/verifiable-credentials-a-deep-dive-for-the-agentic-ai-era/> (2025)
38. Li, R., Wang, H., Liu, C., Li, T.: OAuth 2.0 Delegated Authorization. <https://datatracker.ietf.org/doc/draft-li-oauth-delegated-authorization/> (2026)
39. Liam 'Akiba' Wright: How one trader used morse code to trick Grok into sending them billions of crypto tokens from its verified wallet. <https://cryptorank.io/news/feed/c5874-how-one-trader-exploited-grok-and-morse-code-to-tri>

- ck-ai-agent-into-sending-billions-of-crypto-tokens-from-a-verified-wallet (May 4, 2026)
40. Niyikiza, N.A.: Attenuating Authorization Tokens for Agentic Delegation Chains. <https://datatracker.ietf.org/doc/draft-niyikiza-oauth-attenuating-agent-tokens/> (2026)
 41. OpenID Foundation: What is OpenID Connect. <https://openid.net/developers/how-connect-works/>
 42. Palavali, D.R.: Agentic AI for Self-Sovereign Identity: A Decentralized Zero Trust Framework for Autonomous Microservices. *International Journal of Computational Mathematical Ideas (IJCMI)* **17**(1), 17347–17360 (2025)
 43. Pham, Q., Reid, J., McCullagh, A., Dawson, E.: On a taxonomy of delegation. In: *Emerging Challenges for Security, Privacy and Trust – IFIP Advances in Information and Communication Technology*. vol. 297, pp. 353–363. Springer (2009)
 44. Posta, C.: Why Do We Need a New Gateway for AI Agents? <https://www.solo.io/blog/why-do-we-need-a-new-gateway-for-ai-agents> (2025)
 45. Ragupathy, V.: Secure Agent-Based Architectures for Decentralized AI Identity Management: The DAIS Framework. In: *IEEE 5th Int’l Conf. on AI in Cybersec. (ICAIC)*. pp. 1–6 (2026)
 46. Reitinger, T., Pernul, G.: No Password, No Problem? A Large-Scale Field Study of Passkey Adoption and Usage. In: *2026 IEEE Symposium on Security and Privacy (SP)*. pp. 3607–3626 (2026)
 47. Ren, Q., Xie, S., Wei, L., et al.: When autonomy goes rogue: Preparing for risks of multi-agent collusion in social systems. *arXiv:2507.14660* (2025)
 48. Santos, O.: A New Identity Framework for AI Agents. <https://community.cisco.com/t5/security-blogs/a-new-identity-framework-for-ai-agents/ba-p/5294337> (2025)
 49. Stefanescu, S., Aciobanitei, I.: Lineage: A UMA2.0 Protocol Extension for Multi-Hop LLM Agent Delegation. *IEEE Access* **14**, 104574–104591 (2026)
 50. Stepanov, I., Shtopko, M., Vodianytskyi, D., Lukashov, O.: The million-label ner: Breaking scale barriers with gliner bi-encoder. *arXiv:2602.18487* (2026)
 51. Sunil, B.D., Sinha, I., Maheshwari, P., et al.: Memory Poisoning Attack and Defense on Memory Based LLM-Agents. *arXiv:2601.05504* (2026)
 52. Surapaneni, R., Jha, M., Vakoc, M., Segal, T.: Announcing the Agent2Agent Protocol (A2A). <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interopability/> (2025)
 53. Ylonen, T., et al.: SPKI Certificate Theory. RFC 2693 (Sep 1999). <https://doi.org/10.17487/RFC2693>, <https://www.rfc-editor.org/info/rfc2693>
 54. Yu, F., Jay, C., Yantian, H., et al.: Safeguarding AI Agents: An In-Depth Look at A2A Protocol Risks and Mitigations. <https://live.paloaltonetworks.com/t5/community-blogs/safeguarding-ai-agents-an-in-depth-look-at-a2a-protocol-risks/ba-p/1235996> (2025)
 55. Zaratiana, U., Pasternak, G., Boyd, O., Hurn-Maloney, G., Lewis, A.: GLiNER2: An Efficient Multi-Task Information Extraction System with Schema-Driven Interface. *arXiv:2507.18546* (2025)
 56. Zaratiana, U., Tomeh, N., Holat, P., Charnois, T.: GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer. In: *Procs. 2024 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. pp. 5364–5376 (2024)
 57. Zouari, J.: Toward Agentic IAM: A Probabilistic Authorization Framework for Least Privilege AI Workflows. In: *Proceedings of the IEEE/ACM 12th Int’l Conference on Big Data Computing, Applications and Technologies*. pp. 1–6 (2025)