Applied Human Factors and Ergonomics (AHFE2025), Vol. 199, 2025, 1221–1230
https://doi.org/10.54941/ahfe1006933

Secure Authentication Design for AI Agents

Anna Topol[1], Elizabeth Koumpan[2], Laurentiu Gabriel Ghergu[3], and Grzegorz Jurek[4]

[1]IBM Research, Yorktown Heights NY 10598-0218, USA
[2]IBM Consulting, Ottawa, ON K1G 4K9, Canada
[3]IBM Consulting, Bucharest, B 060201, Romania
[4]IBM Consulting, Krakow, 12 30-150, Poland

ABSTRACT

In the financial industry, artificial intelligence (AI) agents are increasingly adopted to drive higher productivity and economic performance. These solutions require access to critical enterprise systems, such as ERPs, trading platforms, or other solutions, where they need to authenticate and execute actions on behalf of their users. This brings specific security challenges regarding how to reliably authenticate the agents to these critical systems. In this paper, we will examine common anti-patterns in designing authentication mechanisms for agents in function-calling use cases. Additionally, we will present the best solution for implementing authentication and explore two alternative security solutions depending on the capabilities of the external system. Finally, we will provide an example architecture that uses the MCP protocol to authenticate the agent.

Keywords: MCP, Agent security, Financial industry, Enterprise systems, Agent architecture, Security patterns

INTRODUCTION

In the financial industry, agents are becoming pervasive as artificial intelligence expands its applicability to various financial use cases (Weforum, 2025).

However, financial institutions face increasing challenges with security when it is applied to agents, where agents needs to be authenticated in the company core systems, and retrieval augmented generation in particular, as the risks compound (Rooseveltinstitute, 2024).

Regulatory non-compliance concerns, adversarial manipulation, and accountability gaps increase the pressure on the IT division to strengthen the security of its agents.

Specific challenges exist in this space as the agent will usually not authenticate itself but rather perform actions on behalf of its users, or on behalf of another agent, or systems can make decisions and take actions independently without explicit human approval for each action. Impersonation is not a technique that can be easily adopted when it comes to critical IT systems (Medium, 2025).

In this paper we review common security anti-patterns, and provide some recommendations.

Let's analyse communication flow in the diagram below.

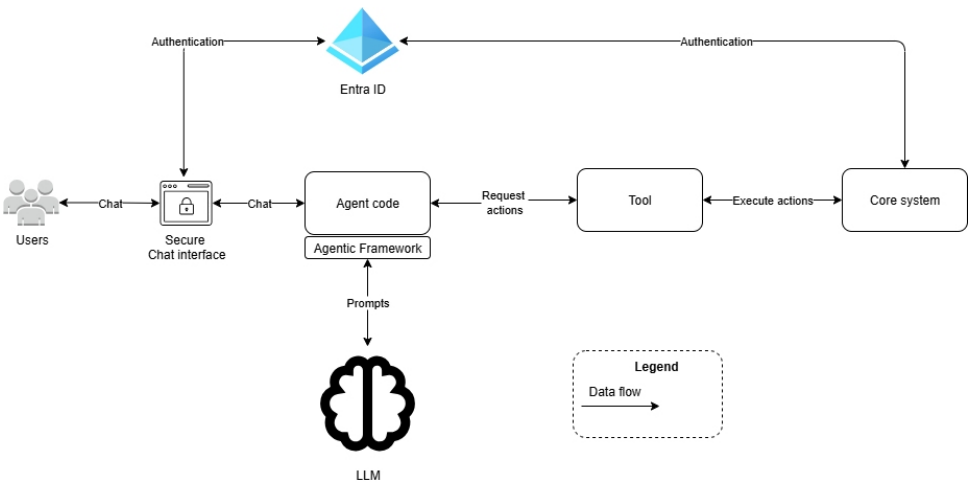

Figure 1: Conceptual model of generic use case for agents (IBM IP owned).

In the above pattern, users converse with an agent through a web application that authenticates them based on a user repository, such as Active Directory (Entra ID, Verify), an LDAP server, or other identity & access management service (Medium, 2025).

The Agent code is responsible for internal orchestration between the Large Language Model (LLM) and existing tools that the model can use. The Agent code relies on an Agentic Framework (e.g., Langchain) to interact with LLM.

The Agent code will use a specialized prompt to inject the list of available tools into the model's context. The LLM model, which is also fine-tuned for tool usage, can reply with a request about which tool it needs and what parameters to send to that tool to fetch additional information from the core banking company system or execute specific actions on behalf of the user in this system.

How do we ensure the Agent code **cannot retrieve or modify information** that the currently logged-in user doesn't have permission to access, even though the Agent itself needs system access to function? If the Agent uses its own credentials or system-level access, it could bypass user-level permissions and access data the user shouldn't see.

When proposing a design for this problem, the architect may utilize a set of the following anti-patterns to address this security challenge:

- the system operates with full autonomy
- acts on behalf of the user
- operates on behalf of an agent.

Each model introduces unique vulnerabilities when implemented incorrectly.

Anti-Patterns When Implementing Security in Agent-Led Retrieval System

1) Use of tokens in the prompts that the LLM is receiving by putting them into context

In this approach, the front-end (assuming a secure chat interface) will authenticate the user and obtain a JWT token from the user repository, then it will inject this token into the context of the LLM.

A special prompt is used to instruct the LLM how to use the tool, requiring the LLM to include the authentication token in its response. This way, the Agent code using a regular expression will be able to parse the response and send the request to the tool, including the JWT token that is used for authentication.

However, in this approach, the LLM is managing the credentials, which is a bad practice as its output may be non-deterministic. It may hallucinate or use the incorrect JWT when authenticating the user, leading to security vulnerabilities, which could result in unauthorized access or authentication failures.

2) Adding the prompts as tool parameters by the LLM

In this scenario, the LLM is constructing the list of input parameters for the tool, and the JWT token is included as one of these parameters.

This is an anti-pattern as the LLM should not be directly involved in the inclusion of the authentication token in the list of parameters for the tool (Github, 2024) If the prompt or tool call is logged, cached, or transmitted insecurely, the JWT can be intercepted and used by unauthorized entities, leading to session hijacking, privilege escalation, or data breaches.

3) Letting the LLM choose which token or credentials should be used

In this scenario, the LLM is empowered to decide which authentication token to use based on a specific user scenario.

The action is performed using a service account that has elevated permissions, where the connectivity to the external system is established using a series of service accounts, some with read-only permissions and others with write permissions. Allowing the LLM to make a decision based on the user scenario, may lead to a compromised token being used, an unauthorized token being leaked, or a violation of security policies, as the model may not have the necessary guardrails to make a safe decision.

4) Determining the user identity from the input text, where the LLM is used to determine the user's identity based on their interactions.

For example, the user declares his name as Bob Smith. Hence, the agent believes that the identity has been provided, and it's executing requests towards the external system using this service principal. This is catastrophic security vulnerability, as anyone can claim to be anyone, there is no password verification, so session, multi- factor authentication.

Identity must always be properly established before agent conversation

5) Not propagating the user identity downstream (IBM, 2025).

The chosen authentication solution should create audit logs that accurately identify the actor executing the action on the external system.

To ensure an external system is informed of the full chain of command when an action originates from a user, is executed by an agent, and then performed on the external system, the authentication token used by the agent must carry information representing this delegation.

Using tokens that do not propagate the user identity towards the external system but rather only propagate the identity of the agent may not be sufficient, as multiple users can use the agent at the same time. The audit logs on the external system side must correctly identify both the actor performing the action and the context in which the action is being performed.

6) Service account has full administrative rights on the core system and can impersonate users (Cloud.google, 2025).

In this scenario, a service account with full administrative permissions is used to authenticate the agent on the external system. This account will then impersonate users and perform actions on their behalf.

This is an anti-pattern, as the agent should have the least amount of privileges when executing actions on behalf of users in external systems. The users must provide specific and limited access credentials to the agents that represent them.

7) Fetching all the external system data, then filtering in the agent

If the external system does not have sufficient security mechanisms, extracting all the data and then filtering it within the agent application may not be considered the best security practice.

A better approach is to ensure that only the necessary data is extracted from the remote system, allowing security to be implemented either in a separate gateway component or within the external system itself.

Filtering logic done inside the agent is not recommended. Relying solely on agent instructions for security checks is non-deterministic, making the system vulnerable to security issues, whereas using variables and filters provides more precise and secure control.

IMPLEMENTING AUTHORIZATION AND AUTHENTICATION WHEN USING TOOLS

We recommend to leverage the OAuth2 (Hardt and Jones, 2012) standard and generate authentication tokens that are then managed directly by the Agent code.

A high-level overview of the architecture is presented in Figure 2 below.

The user will open the graphical user interface, invoke the agent, and be redirected to the login page of the identity provider (IdP). On a successful login, the identity provider provides an authentication token (usually a JWT token), the agent code will use when receiving requests from the LLM to leverage the tool.

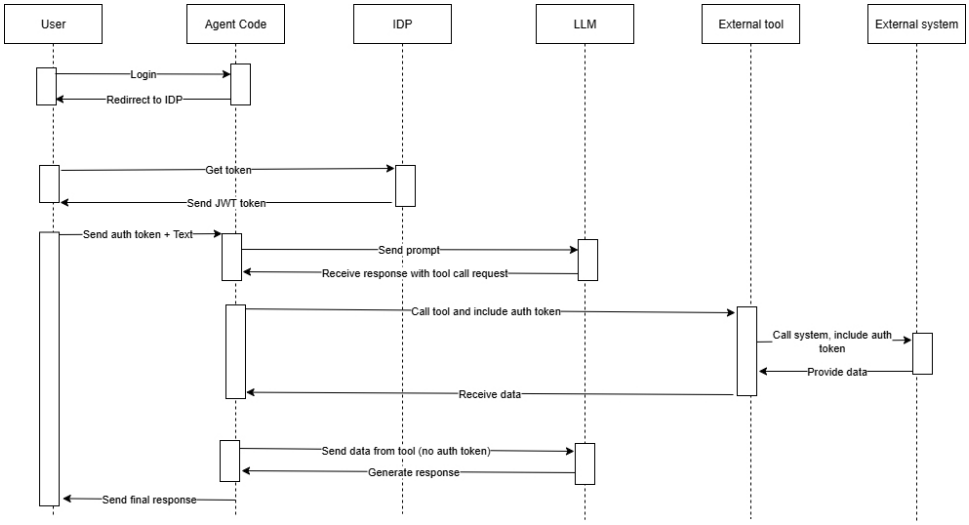

Figure 2: Authentication flow between agent and external system (IBM data flow design).

The LLM will not see or use this authentication token when it needs to access the external system using the tool. Instead, the agent code will send just the user text as a prompt to the LLM and load the information about the existing tools in the context of that prompt. The LLM fine-tuned for tool usage, can decide to extract data from the external system by calling the tool.

The Agent code will use regular expressions to analyse the LLM request and extract parameters necessary to call the tool, except for the authentication token. The Agent code will call the external tool by including the authentication token, which was stored in the user session on the server side.

Once the tool retrieves the needed data elements, they will be provided to the LLM for generating the final answer without giving any information regarding the authentication token. Finally, the response will be sent to the user.

One concern here is the situation where the tool may allow the execution of destructive actions on external systems, such as enabling the LLM to delete a table it receives as a parameter. In this situation, the LLM may provide the incorrect table name, and the tool may perform a destructive action on behalf of the user because the authentication token has sufficient permissions.

We need to implement guardrails, f.e. bringing 'human in the loop', where the agent asks the user to confirm that the action about to be performed is correct (AWS Documentation, 2025).

For such actions that are considered risky by the programmer, an additional step may be displayed to the user directly by the agent to confirm that the action proposed by the LLM is correct. If the user confirms, then the agent code will execute the tool call with the specified LLM parameters on the external system.

In such situations, there may be additional challenges, depending on the external system's capabilities

Integration With External Systems That Offer No Interface (Linkedin, 2025)

In some cases, the external system may not provide an API interface that the tool can leverage.

In this scenario, one option is to leverage Robotic Process Automation to execute the login procedure and specific actions in external systems. That way, we can integrate with this solution even if no API interface is available.

An architecture overview is presented in Figure 3 below.

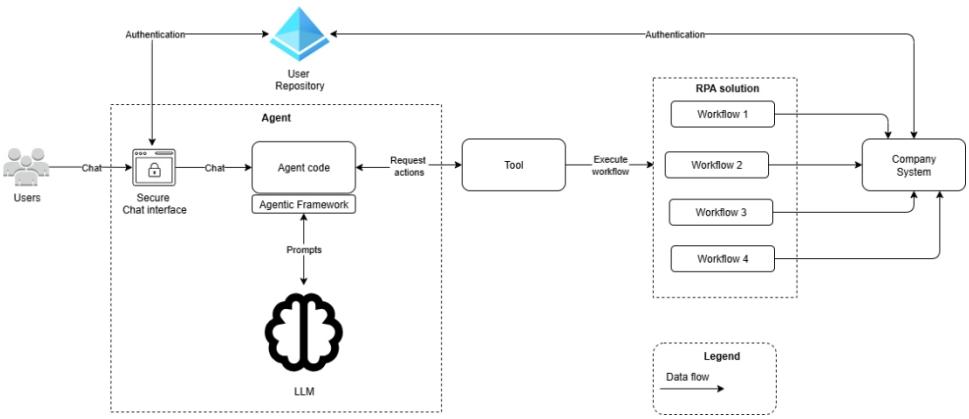

Figure 3: Use of RPA to integrate agents with external systems (IBM).

In this scenario, the tool will be used to execute a specific workflow on the company system. These workflows can trigger the execution of business processes or authenticate the user to the system.

The authentication token will be propagated from the Secure Chat interface to the Tool in the same way as described previously, then the workflow may map a specific claim from the JWT token (like the user email) to a local repository that maps the user to a given authentication method which the RPA bot will use as part of a specific flow to login on the Company system.

Integration With External Systems With Legacy Protocols (Moesif, 2025)

In a different scenario, the company system may use an authentication protocol that is not supported by the company that is delivering the agent. In this scenario, we can use a middleware to implement the authentication protocol conversion. This will allow both the agent and the company system to leverage the authentication protocol, which is compliant with their company standards.

A high-level architecture overview is presented in Figure 4 below.

The middleware is a software component of Company 1 (SaaS provider), which is mapping the authentication tokens issued for users when connecting to the front-end of the agent application to API keys, which are required to connect to the external system located under the control of Company 2.

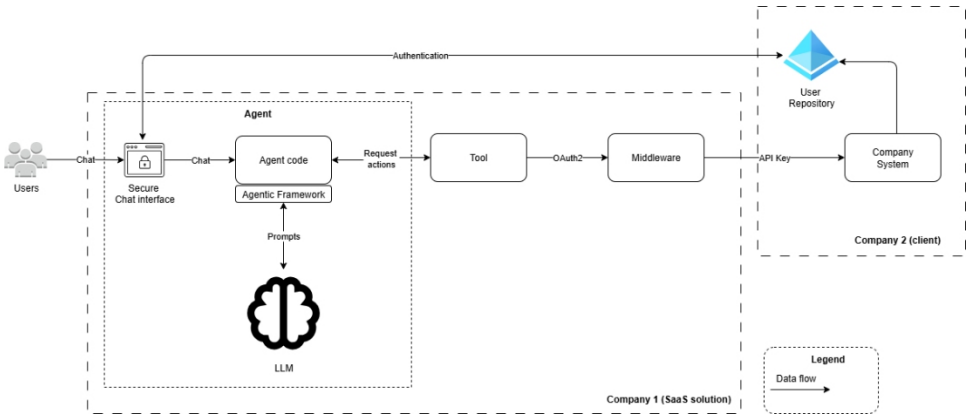

Figure 4: SaaS agent with authentication protocol conversion (IBM).

The middleware can use the claims from the JWT token, like the user's email address, to propagate the identity into the Company System. As the JWT tokens are signed, the middleware can trust the claim to be valid.

Hence, the proposed solution will enable SaaS providers of agentic solutions to connect to external systems that do not comply with their internal corporate standards in terms of authentication and authorization.

Example Security Implementation With MCP Protocol (Medium, 2025)

The real-life implementation of the pattern described above can be achieved with the help of the MCP protocol. Let's assume we plan to implement an agent assistant that responds to queries related to invoice status for a client using multiple types of applications.

The scenario may look straightforward, but complexity arises when you realize that the structure of business units and the authorization model differ between various systems. For example, SAP has a fixed model composed of companies and plants, while EBS(e-business suite) allows the structure of business units in a more flexible way, adjusted for client preferences.

In such a scenario the proposed architecture should be built with MCP server that manages AI Tools, Token Manager.

MCP Server is a module that:

- Enables tools that external agents can use
- Manages the accesses, can request identity information, and pass it to the token manager to obtain a token
- Stores the tokens and passes them to the tool when needed
- Logs and monitors the accesses, ensuring traceability

Token manager is a custom module that can:

- Understand the structure of business units in the subjected systems
- Able to retrieve the authorization information from the system based on a given user identifier
- Create or obtain tokens that allow tools to execute actions in the systems

- It can be short-lived tokens created on request (preferred way)
- Or standard accounts segregated by the business unit and role

• Pass token to integration tools to let them connect directly to the external system to execute tasks with limited privileges.

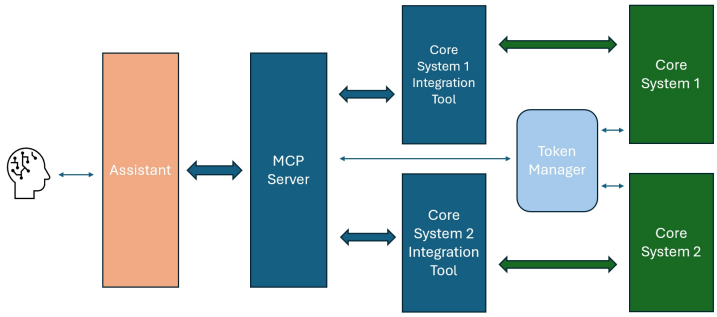

Figure 5: MCP architecture (IBM).

Types of Tokens, Token Life Span

To ensure secure operations, only short-lived tokens should be passed to the AI tool. These tokens should remain valid only for the duration necessary to complete the requested operation. Ideally, such tokens would be generated directly within the enterprise application; however, this is often not feasible, as most core systems do not support this capability. To overcome this limitation, a token manager can be used to generate short-lived tokens, which are then managed by the token manager module.

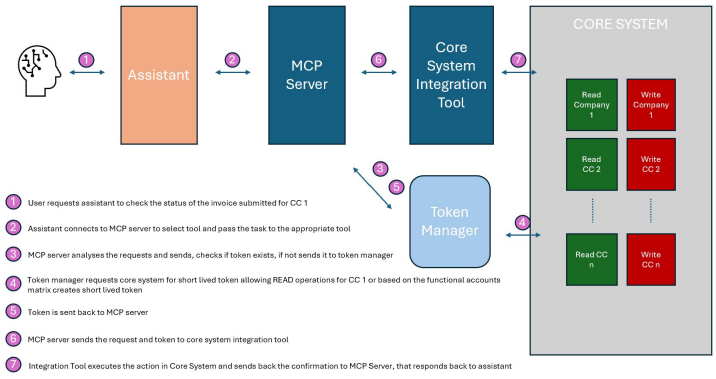

Figure 6: Token lifespan (IBM).

In this approach, the token manager relies on a matrix of functional accounts, each configured with access to specific business units and

features. While this method requires the creation and maintenance of many functional accounts within the core enterprise system, it offers a key security benefit: agents are never granted broad system access, and credentials are not distributed across multiple tools or agents. Instead, access is centrally controlled by the token manager module, ensuring auditability and traceability.

CONCLUSION

Agents, cross–agent collaboration is becoming pervasive in the financial industry, and many other industries. This is raising authentication and authorization challenges as these solutions have to connect to critical company systems. Various anti-patterns can be found in the industry, given that implementing security for this new technology is not very straightforward.

Depending on the connectivity needs for various enterprise solutions worldwide, it may be necessary to utilize Robotic Process Automation to integrate with legacy systems or leverage middleware solutions to modify the authentication protocol and comply with company policies.

For tool access, we recommend to use the MCP protocol, which supports OAuth2 as a standard protocol for authenticating users to external company systems. The MCP Server can be combined with a Token Manager component to generate short-lived tokens which are based on a matrix of functional accounts, each configured with specific business units and features.

While security-efficient AI-driven automation of routine IAM tasks and AI-optimized threat intelligence initiatives are promising, the efficacy of such approaches has yet to be quantified at scale. Authentication methods continue to mature.

With MCP and similar frameworks like LangChain or AutoGen, AI agents are no longer just able to answer questions or provide information – they're actually executing actions quickly. And this shift is making it dangerously easy to give those services too much power and control, without fully understanding the implications (Echohq, 2025)

- **Autonomous systems** require the most restrictive controls with kill switches and bounded authority
- **User delegation** must never store credentials and should use short-lived, scoped tokens
- **Agent identities** need least-privilege service accounts with strong authentication
- **All models** benefit from defense in depth, comprehensive logging, and continuous monitoring
- **Context matters**: Choose security controls based on the autonomy model and risk profile

Security, observability, and auditability have to evolve alongside Security frameworks. Because when something goes wrong, it won't be the model's fault, but it'll be the system around it.

ACKNOWLEDGMENT

The authors would like to acknowledge to Jean Stephane Payraudeau, Sridhar Muppidi, Jose A Rodriguesz, for their support.

REFERENCES

Artificial Intelligence in Financial Services. https://reports.weforum.org/docs/WEF_Artificial_Intelligence_in_Financial_Services_2025.pdf

Agent Authentication in AI Systems. https://medium.com/@lahirugmg/agent-authentication-in-ai-systems-a67008e47a09

Agent Authentication in AI Systems. https://medium.com/@lahirugmg/agent-authentication-in-ai-systems-a67008e47a09

Expert Advice on Integrating APIs with Legacy Systems in 2025 - https://www.moesif.com/blog/monitoring/Expert-Advice-on-Integrating-APIs-with-Legacy-Systems-in-2025/

Hardt, D., & Jones, M. (2012). *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force (IETF). [RFC 6749]. https://doi.org/10.17487/RFC6749

Implement safeguards for your application by associating a guardrail with your agent. https://docs.aws.amazon.com/bedrock/latest/userguide/agents-guardrail.html

Identity propagation and distributed security. https://www.ibm.com/docs/en/cics-ts/5.6.0?topic=securing-identity-propagation-distributed-security

Model Context Protocol (MCP) real world use cases, adoptions and comparison to functional calling. https://medium.com/@laowang_journey/model-context-protocol-mcp-real-world-use-cases-adoptions-and-comparison-to-functional-calling-9320b775845c

Passing an Authorization Token to the tool & avoiding the LLM. #534 - https://github.com/langchain-ai/langserve/discussions/534

Service account impersonation. https://cloud.google.com/iam/docs/service-account-impersonation

System Integration Without APIs: Creative Approaches That Deliver. https://www.linkedin.com/pulse/system-integration-without-apis-creative-approaches-deliver-vema-wtwhc/

The Risks of Generative AI Agents to Financial Services - https://rooseveltinstitute.org/publications/the-risks-of-generative-ai-agents-to-financial-services/

The 6 hidden risks in deploying with MCP. https://www.echohq.com/gated/hidden-risks-of-mcp