gateway模块设计

1. 方案定位：把 MCP-Auth Gateway 做成「OAuth2.1 + LLM Agent API Gateway」

你原本的四个目标（身份可验证、能力可控、完整性可证明、可撤销与可审计）其实非常契合最近社区对 MCP 授权的共识：

- GitGuardian 在 *OAuth for MCP — Emerging Enterprise Patterns for Agent Authorization* 里提出：**OAuth 2.1 足够成熟，但需要通过“网关 + 短生命周期、按资源严格 scope 的 token”才能适应 agent 的非确定性调用链，并且形成清晰的审计边界**。GitGuardian Blog
- MCP 官方 Authorization 规范同样把 MCP Server 定位为 OAuth 资源服务器，建议使用 JWT + Resource Indicators 对每个 MCP Server、每个 scope 做精细控制。Model Context Protocol+1

因此，你的 MCP-Auth Gateway 最现实的定位是：

在 *User → Host → MCP Client → MCP Servers → 下游 API* 这条链路中，引入一个 **统一的 OAuth2.1 授权网关**，集中做：

- OAuth token 获取与验证；
- scope / Resource Indicator 控制；
- PoP / DPoP 绑定与请求级签名；
- 回调验签和 token 撤销；
- 工具 manifest（含 OAuth 元数据）的签名验证与策略检查。

这个定位与 GitGuardian 建议的「**gateway-based authorization**」几乎完全一致，只是你多加了 **Signed Manifest + 回调验签** 这两个更“协议化”的强化点，可作为你论文的创新点之一。GitGuardian Blog+1

---

2. 架构与信任边界（微调你现有 6.x.2）

组件保持不变，只是把与 OAuth 相关的职能写清楚一点：

- **User / Resource Owner**：在人机界面上完成 OAuth 授权、确认高风险操作（HITL）。
- **Host / LLM / MCP Client（OAuth Client 代理）**：
  - 负责与用户交互，以及把自然语言任务转成 MCP 调用；

- 自己不直接持久化长期 token，而是把授权流程交给 Gateway。

- **MCP-Auth Gateway（核心贡献）**：
  - 作为 OAuth2.1 里的“**授权中介/反向代理**”：统一验证外部 token，按策略换成短期、绑定特定 MCP Server 的下游 token；GitGuardian Blog
  - 充当 **Policy Enforcement Point（PEP）**：检查 profile / scope / capabilities，决定是否放行每一次工具调用；
  - 负责请求级签名、PoP/DPoP 证明、回调验签、token introspection 与 revocation 调用。IETF Datatracker+1
- **MCP Server（OAuth Resource Server）**：
  - 按 MCP Authorization 规范，对 JWT access token 做验证（签名、exp、aud/resource、scope 等）；Model Context Protocol+1
  - 只信任来自 Gateway 的流量（例如通过 mTLS、网络层 ACL 或者内部网络）。
- **AuthZ Server / IdP（OAuth2.1 / OIDC 提供方）**：
  - 发行短生命周期的访问 token（最好是 sender-constrained PoP token），并支持 introspection / revocation。IETF Datatracker+1
- **Manifest Authority / Registry**：
  - 为每个 MCP Server 发布 **Signed Manifest**，声明 tool_id、支持的 scope / resource、回调地址、服务器公钥等信息，用于供应链完整性与工具身份验证。Medium+1

信任边界：

- 用户 ↔️ Host：仍然视为“前端”，这里主要是 UI 与 HITL，不做强安全假设。
- Host ↔️ Gateway：Host 被视为“可能会被 prompt 污染但有基本身份”的客户端，Gateway 对其请求做强约束。OWASP LLM Top10 里把这种 **“excessive agency + prompt injection 下的插件滥用”** 视作核心风险，推荐使用中间代理来隔离 LLM 与关键系统。Checkmarx+1
- Gateway ↔️ MCP Servers ↔️ 外部资源：被视为“受保护资源域”，只接受通过 Gateway 控制的 OAuth token 与签名流量。

---

3. Gateway 内部模块（综合版）

结合你和文献里的思路，我建议最终 Gateway 拆成 **5 个模块**（前 4 个是你原来就有的，第 5 个是“现实部署必需”的审计模块）：

3.1 Manifest 验证模块（Manifest Verifier）

职责：

- 从 Registry 拉取 **Signed Manifest**，验证开发者签名 + 平台签名，校验证书链；Medium+1
- 解析字段：
  - `tool_id`
  - MCP 端点 URI（将作为 OAuth Resource Indicator / audience）GitGuardian Blog
  - 支持的 `scopes`、允许的 `redirect_uris`（用于 OAuth 客户端注册）IETF Datatracker
  - MCP Server 的签名公钥（用于回调验签）。
- 将解析结果缓存，供 **AuthZ&Policy** 和 **Callback Verifier** 使用。

现实依据：
OWASP LLM Top10 将“**Insecure LLM Supply Chain**”视作主要风险，明确建议对插件/扩展使用签名机制和 SBOM 追踪。Checkmarx GitGuardian 也披露了 Smithery.ai 宿主的路径遍历漏洞曾经暴露了数千个 MCP server 与 API Key，说明工具供应链确实可能被整体攻陷，这直接证明 **Signed Manifest + Registry** 的必要性。GitGuardian Blog

---

3.2 授权与策略模块（AuthZ & Policy Engine）

职责：

- 将每个 MCP Server 映射为 OAuth 里的 **Resource Server**，配合 Resource Indicators 将 token 严格绑定到单个 server（aud/resource 字段）。GitGuardian Blog+1
- 与 AuthZ Server 使用 OAuth2.1 **Authorization Code / Client Credentials** 等标准授权模式：
  - User-scoped：授权访问用户数据（文件、GitHub 仓库等）；
  - System-scoped：工具自身访问公共或组织资源。GitGuardian Blog
- 强制使用 OAuth 2.1 推荐的安全配置：
  - 授权码 + PKCE、移除 implicit flow、严格校验 redirect_uri（防止回调劫持）。IETF Datatracker+1
- 根据 manifest 中声明的能力 + 本地策略（可以参考你原来的 profile 概念，但不用在协议层暴露），决定：
  - 允许工具使用哪些 grant type / scopes；
  - access token 最大过期时间；是否允许 refresh token；
  - 需要 HITL 的风险级别。

现实依据：

- OAuth2.1 草案本身就强调 sender-constrained token、严格 redirect URI、移除隐式授权等实践来防止 token 滥用与重定向攻击。IETF Datatracker
- GitGuardian 和 Curity 都建议在 MCP 里使用 Resource Indicators + 短生命周期 token，并把 refresh token 当作“王冠上的珠宝”严加保护，这与你的“长生命周期 token 滥用”问题完全对齐。GitGuardian Blog+1

3.3 请求保护模块（Request Protection: PoP + 签名）

职责：

- 对 Host → Gateway 的 MCP 请求做规范化（canonicalization），计算请求体哈希（method, path, headers, body）。
- 根据 OAuth 2.1 的 **sender-constrained token** 建议，优先使用：
  - **DPoP**：每次请求附带一个由客户端密钥签名的 JWK 证明，token 绑定到该密钥；IETF Datatracker+1
  - 或者 mTLS PoP：通过双向 TLS，把 token 绑定到 TLS 证书。Okta Developer+1
- 对发往 MCP Server 的请求附加：
  - `mcp_authz.request_signature`（含 nonce, timestamp, body_hash）；
  - PoP / mTLS 相关的 header；
  - 最小必要 scope 的 access token。
- 在入口验证 Host 的签名（区分“合法 Host 被 prompt 污染”和“纯伪造请求”），在出口只允许 Gateway 自己签名后的请求流向 MCP Server。

现实依据：

- PoP / DPoP 已被 IETF 正式标准化，用于防止 Bearer token 被窃取后在别处重放。IBM、Okta 等厂商的最佳实践都建议对高价值 API 使用 sender-constrained token。IBM+3IETF Datatracker+3OAuth社区+3
- OWASP API Security Top10 将 **Broken Object Level Authorization / Excessive Data Exposure** 列为核心风险，建议通过细粒度 scope 和请求级检查防止 API 被滥用。OWASP Foundation+1 Gateway 在 MCP 里承担的就是这一责任。

3.4 回调与撤销模块（Callback & Revocation）

职责：

- 强制所有 MCP Server 的 callback / streaming 链路使用在 manifest 中注册的回调端点，并通过服务器私钥对回调 payload 签名：
  - `server_signature = Sign_k_server(hash(session_id, call_id, payload, nonce, timestamp))`
- Gateway 验证：
  - 回调是否来自允许的 IP/mTLS 对端；
  - `server_signature` 是否和 manifest 中的公钥匹配；
  - nonce / timestamp 是否在时效内，防止重放。
- 对高风险 C 类操作（例如发交易、删文件）：
  - 在每次调用前通过 introspection 查询 token 状态；
  - 或按一定频率自动 introspection。
- 提供 `authz.revoke` 接口，让用户/Host 可以一键撤销某个会话或工具的所有权限，Gateway 调用 AuthZ Server 的 revocation endpoint，同时更新本地缓存。IETF Datatracker+1

现实依据：

- OAuth2.1 明确建议使用 revocation / introspection 端点实现 token 生命周期控制。IETF Datatracker
- JWT 自身不易撤销，GitGuardian 文章就强调必须 **短生命周期 + 辅助撤销机制**，否则一旦泄露将持续可利用。GitGuardian Blog
- ITPro 针对 MCP 的安全分析指出，目前大量 MCP Server 配置不当，可能成为 Token 泄露与滥用的集中点，而没有统一回调与撤销逻辑的体系很难追踪问题。你的 Gateway 方案正好补上这块“统管”的空白。IT Pro

---

3.5 审计与监控模块（Audit & Monitoring）

职责：

- 记录结构化审计日志：包括 user / tool / MCP server / scopes / 参数摘要 / 决策（允许/拒绝）等信息；
- 提供可查询接口，支持事后取证与合规审计；
- 对异常行为（scope 拒绝激增、同一 token 短时间内大量调用）发出告警。

现实依据：

- GitGuardian 指出，gateway 模式的一个显著优点就是形成清晰的**“审计边界”**，所有授权决策都集中在一个地方有利于合规与事后分析。GitGuardian Blog
- OWASP LLM Top10 同样强调要对 LLM 与后端系统之间的交互进行全面日志和监控，以识别 prompt injection 与“Excessive Agency”。Checkmarx+1
- Solo 的 Agent Gateway、HashBlock 的“Secure API Gateway for AI Agents”都把审计和可观测性列为一等公民功能，说明这是现实部署的硬需求。Solo+1

---

4. 协议流程如何对齐四类攻击

下面这四条流程，其实就是你原来 6.x.5 的“现实化版本”，我只强调每一步在防什么攻击，以及可以引用哪些标准 / 案例。

4.1 工具注册 & Manifest 获取 —— 对抗「工具身份伪造」

1. 开发者向 Manifest Registry 提交工具 manifest，并用自己的私钥签名；平台可再加一层平台签名。Medium+1
2. Gateway 从 Registry 拉取 Signed Manifest，验证双重签名与证书链；
3. 将 `tool_id` 、 `resource` （MCP Server 地址）、 `allowed_scopes` 、 `server_pubkey` 等信息缓存。

防御点：

- 即便攻击者控制了某个 MCP Server 的宿主环境，若不能拿到对应私钥并在 Registry 通过签名验证，就无法伪装成原有工具；
- 这对 GitGuardian 报道的那类“托管平台一个路径遍历漏洞导致批量 MCP Server 与 API 密钥泄露”的场景是重要缓解手段。GitGuardian Blog

---

4.2 OAuth 授权 & token 签发 —— 对抗「权限滥用 + 长周期 Token 滥用」

1. 用户在 Host 中发起高权限工具初始化，Host 把请求转发给 Gateway；
2. Gateway 根据 Manifest 决定使用 **Authorization Code** 还是 **Client Credentials**，并将 Resource Indicator 指向对应 MCP Server；GitGuardian Blog+1
3. 授权页面由 AuthZ Server 提供，用户在浏览器中对特定 scope 授权；
4. Gateway 通过 PKCE 换取 access token（可选 refresh token），并立即：

- 缩短 access token 生命周期（分钟级）；
- 把 token 加密存储在 Gateway 内部，不暴露给 LLM。GitGuardian Blog+1

防御点：

- 利用 OAuth2.1 的 scope + Resource Indicator，实现每个工具、每个 MCP Server 独立的最小权限 token，避免“万能 token”被滥用；GitGuardian Blog+1
- 通过短生命周期 + 切断 token 与 LLM 上下文的接触，降低长周期 token 泄露后的风险；这正是 OAuth 社区和 GitGuardian 对 MCP 的核心建议。GitGuardian Blog+1

4.3 工具调用 & 请求保护 —— 对抗「权限滥用」

1. Host 通过 MCP Client 发起 `tools.call`，带上 `toolName` 和 arguments；
2. Gateway 对请求做 canonicalization，检查：
   - 调用是否在 manifest 能力、已授权 scopes 范围内；
   - 当前会话是否已经过 HITL 确认（对高风险操作）。OWASP Foundation+1
3. Gateway 生成 `request_signature`（包含 nonce, ts, body_hash）和 PoP/DPoP 证明，把 **最小 scope 的 PoP token** 注入 header；
4. MCP Server 验证 token（issuer、aud/resource、scope、exp）、PoP 证明和 `request_signature` 后才执行。

防御点：

- OWASP API Security 和 OWASP LLM Top10 都强调“最小权限 + 输出前验证”。Gateway 在这里把 **scope 检查 + 请求签名 + HITL** 集中到一点，限制 LLM 的 “Excessive Agency”。OWASP Foundation+2Checkmarx+2

4.4 回调 / 结果返回 —— 对抗「回调污染攻击」

1. MCP Server 在 Manifest 中声明固定的回调通道（例如 WebSocket 或反向流式通道），以及用于签名回调的公钥；Medium+1
2. 每个 callback 消息都包含 `server_signature`，对 (session_id, call_id, payload, nonce, ts) 哈希签名；
3. Gateway 验证：
   - 回调来源是否为注册的 MCP Server（mTLS / IP / DNS）；

- `server_signature` 是否正确；
- nonce / ts 是否在时效内。

4. 验证通过后，才把 callback 结果注入到 LLM 上下文。

防御点：

- 传统 OAuth 里“回调污染”主要出现在 redirect_uri 被滥用，这在 OAuth2.1 中通过严格 redirect_uri 注册和 mix-up 防御来缓解。IETF Datatracker
- MCP 的情况类似：如果不对回调路径和内容做签名与白名单约束，就可能被恶意 server 或 MITM 注入伪造结果。Gateway 的这一步把 OAuth2.1 的 redirect 安全思想扩展到了 MCP 回调通道。

4.5 撤销 & 审计 —— 对抗「长生命周期 Token 滥用」

1. 用户在 Host 的 UI 中撤销工具权限；
2. Host 调用 Gateway 的 `authz.revoke`，Gateway 再调用 AuthZ Server 的 revocation endpoint；IETF Datatracker
3. Gateway 更新本地缓存，一旦发现某 token 已撤销，在后续调用中拒绝转发，并记录审计信息；
4. 审计模块可以用于追踪“token 泄露后短时间内是否被滥用”、“某工具是否存在异常访问模式”等。GitGuardian Blog+1

---

5. 可行性论证：与现实实践的对齐

为了让这套方案在论文里显得“不是纸上谈兵”，你可以明确对齐几个现实案例：

1. **GitGuardian：OAuth for MCP + gateway 模式**
   - 文章直接指出：MCP 部署中，企业已经开始采用 **集中授权网关** 来验证 token、做上下文感知决策、把长生命周期 token 转换成下游短期断言，并形成审计边界，这与 MCP-Auth Gateway 的定位完全一致。GitGuardian Blog
2. **Curity：Design MCP Authorization to securely expose APIs**
   - Curity 建议在 MCP 中使用标准 OAuth / OIDC、Resource Indicators 和 JWT 验证，把 MCP Server 视为 Resource Server，并通过集中组件输出策略决策，与 AuthZ & Policy Engine 模块高度一致。Curity+1
3. **Solo Agent Gateway & HashBlock API Gateway for AI Agents**
   - Solo 的 Agent Gateway 明确支持 MCP / A2A 协议、外部认证（extAuthz）、细粒度流控和

OpenTelemetry 审计，把它定位为“**AI-native Gateway**，用于在企业环境中统一管理 agent 访问与策略”。Solo

- HashBlock 的“Building a Secure API Gateway for AI Agents with OAuth, Rate-Limits, and Audit Trails”提出使用 OAuth + 网关 + 审计的方式保护 LLM Agents 调用后端 API，这验证了你在 Gateway 里做 PoP、审计、限流等设计的实用价值。Medium

4. MCP 本身的安全争议

- The Verge 报道 Claude Desktop 通过 MCP 直接操作 GitHub、终端等工具，展示了 MCP 在现实产品中的高权限能力。theverge.com
- ITPro 等媒体指出当前 MCP 生态中普遍存在工具配置错误、缺乏企业级授权与审计的问题，强调需要更强的安全控制，这正是 MCP-Auth Gateway 想解决的痛点。IT Pro