ViewLegacy.

MCP時代の業務システム接続における最小権限設計

技術白書

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年6月5日

本書は、AIエージェントをMCPおよびツール呼び出しインターフェース経由で業務システムに接続する際に、データベース、スキーマ、CRUD操作、認証情報に対する過剰な権限を渡さないための設計を評価する。

1. エグゼクティブサマリー

MCPは、AIエージェントが外部ツールを発見し、呼び出すことを容易にする。この相互運用性は有用だが、セキュリティ上の問いは「エージェントがシステムに到達できるか」から、「エージェントがどの操作を発見し、呼び出し、どのパラメータを与え、何を観測できるか」に変わる。

本書は5つの公開パターンを比較する。直接SQLツール、汎用的なテーブル単位CRUDツール、広範な内部APIカタログ、ロールでフィルタされたAPIまたはツールカタログ、そしてMCPのツール発見を認可範囲でフィルタするエージェント定義のCapabilityベースアクセスである。

結論は限定的かつバランスの取れたものにする。最小権限は実行時の認可だけでなく、ツール発見、スキーマ公開、入力パラメータ、出力フィールド、行数制限、ログ、監査可視性にも適用されるべきである。Capabilityベースアクセスは、広範なデータベースアクセスやテーブルアクセスではなく、明示的な業務操作を公開することで、露出する権限を減らせる。

ただし、すべてのリスクを取り除くわけではない。分析、大規模BI、柔軟なアドホッククエリが必要な用途では常に最適とは限らない。特に適しているのは、読み取り中心で、操作を明示的な業務Capabilityとして表現できる、レガシー業務システムへの運用アクセスである。

2. 背景

AIエージェントは、顧客ステータス、注文履歴、在庫状況、請求ステータス、サポート文脈、社内ワークフロー状態など、運用データへのアクセスを必要とし始めている。多くの組織では、そのデータは自律的なエージェント実行環境を前提としていないレガシーデータベースや業務アプリケーションに残っている。

MCPは、クライアントがツールを発見し、外部システムを呼び出す方法を標準化する。MCPのツールモデルでは、クライアントは`tools/list`で利用可能なツールを取得し、`tools/call`で名前付きツールに構造化引数を渡して実行できる。ツール定義には名前、説明、スキーマが含まれる。これは相互運用性に役立つが、カタログ自体が機密性の高い業務Capability、エンティティ名、ワークフロー、スキーマ、運用構造を漏らす可能性がある。

レガシーデータベースには、顧客、注文、在庫、会計、従業員データが同じ環境に含まれることが多い。生SQLアクセス、スキーマ全体のCRUD API、フィルタされていないOpenAPIやMCP定義を公開すると、実行前の発見段階で最小権限に反することがある。

プロンプトインジェクションとツール誤用は、この境界をさらに重要にする。最小権限はプロンプトでモデルに依頼するだけでは不十分であり、モデルの外側で強制されなければならない。

3. 比較する公開パターン（1/5）

パターン1: データベースまたはSQLツールを直接公開する

AIエージェントは、`run_sql`のような生データベースまたはSQLツールを発見できる。実行できるクエリは、主にデータベース認証情報、データベースロール、SQLパーサ、ラッパー側の制限によって決まる。

発見可能な情報には、テーブル名、スキーマ説明、クエリ例、データベースエラー、イントロスペクション機能が含まれ得る。認可は通常、データベースロールとツールラッパーで実施される。

エージェントが侵害、誤誘導、または悪意ある指示を受けた場合、攻撃者は広範なテーブル列挙、無関係な業務領域をまたぐJOIN、行数制限の回避、認証情報が許す場合の書き込み操作を試みられる。読み取り専用のデータベースロールであっても、特定の業務タスクに必要な範囲を大きく超えるデータを公開し得る。

3. 比較する公開パターン（2/5）

パターン2: 汎用的なテーブル単位CRUDツール

AIエージェントは、`list_customers`、`get_order`、`update_invoice`、`delete_order`のような、テーブルまたはリソースを反映したツールを発見できる。エージェントがデータベース認証情報や生SQLアクセスを持たなくても、ツールカタログはテーブル形状の業務構造を公開する。

呼び出し可能な表面は、特定の業務ワークフローより広い。認可は一般にAPI層で、エンドポイント、ロール、オブジェクト所有権、リソースポリシーによって実施される。ツール定義が発見前にフィルタされる場合も、されない場合もある。

エージェントが侵害または誤誘導されると、無関係なテーブル操作を呼ぶ、不要なフィールドを要求する、レコードを列挙する、別のアプリケーション文脈のために公開された更新・削除ツールを使う、といった事態が起きる。

3. 比較する公開パターン（3/5）

パターン3: 広範な内部APIカタログ

AIエージェントは、MCPツールまたはOpenAPI由来の関数として公開された、多数の内部APIを発見できる。この方式は生のデータベース認証情報を避けられるが、内部システム、エンティティ名、ワークフロー、管理機能の詳細な地図を公開し得る。

呼び出し可能な表面はAPIキーまたはユーザーIDに依存する。多くのデプロイでは、発見範囲が実行範囲より広い。つまり、エージェントは後で`403 Forbidden`になるエンドポイントも見ることができる。

APIが適切に設計されていれば生SQLよりよいが、広範なカタログはモデルを混乱させ、偶発的な誤用を増やす可能性がある。エージェントが機密ワークフローを探すよう悪意ある指示を受けた場合、実行がブロックされても、カタログ自体が偵察情報になる。

3. 比較する公開パターン（4/5）

パターン4: ロールでフィルタされたAPIまたはツールカタログ

AIエージェントは、呼び出し元のロール、ユーザーID、テナント、APIキーのスコープに関連するツールまたはエンドポイントだけを発見できる。発見時のフィルタリングは不要な可視性を減らし、実行時認可は許可範囲外の呼び出しを拒否する。

これは広範なカタログに比べて意味のある改善である。ツール説明の公開範囲を制限し、偶発的な誤用を減らし、`tools/list`を認可境界の一部にする。

ただし、設計は正しいロール定義、厳密なカタログフィルタリング、独立した実行時チェックに依存する。発見フィルタリングと実行認可がずれると、隠されているはずのツールが呼び出し可能になったり、見えているツールが想定以上の情報を返したりする。

3. 比較する公開パターン（5/5）

パターン5: エージェント定義のCapabilityベースアクセス

このパターンでは、AIエージェントは生SQL、スキーマアクセス、広範なテーブルCRUD、データベース認証情報ではなく、名前付きの業務Capabilityを発見し呼び出す。例は`get_customer_summary`、`search_orders`、`check_inventory`、`get_invoice_status`である。

選択された操作はREST APIとMCPで公開される。APIキーは許可されたCapabilityにスコープされる。OpenAPIとMCPツール定義は、呼び出し元の認可範囲に基づいてフィルタされる。クラウドゲートウェイはルーティング、ID、認可、可観測性を扱うが、生SQLやデータベース認証情報を知るべきではない。

オンプレミスエージェントは、SQL、データベース認証情報、検証、実行ポリシー、プリペアドSQL実行、出力フィルタリング、行数制限、業務上の意味を所有する。`capability.yaml`が存在し得る操作を定義する。初期の読み取り専用スコープでは、破壊的操作は設計上スコープ外である。

3. 図: 広範なMCPツール公開

```
[AI Agent / MCP Client]
        |
        | tools/list returns broad catalog
        v
[MCP Server]
        |-- run_sql ---------------------------+
        |-- list_customers --------------------|
        |-- list_employee_records -------------|--> [(Legacy Database)]
        |-- delete_order ----------------------|
        |-- export_accounting_data ------------+
        ^
        |
  prompt injection can steer discovery and calls
```

広範な公開では、呼び出し元が実行を試す前に、ツール発見だけで機密性の高い操作が見える可能性がある。`tools/call`をブロックすることは必要だが、フィルタされていないカタログからのメタデータ露出は取り除けない。

3. 図: 最小権限の発見と実行

ゲートウェイは、エージェントが発見・呼び出しできる対象を狭める。その後、エージェント側の実行境界がパラメータを検証し、データが呼び出し元に返る前に結果をフィルタする。

3. 図: 認可スコープ例

```
sales_partner key
  |-- get_customer_summary
  +-- search_orders

support_agent key
  |-- get_customer_summary
  |-- search_orders
  |-- get_invoice_status
  +-- check_inventory

internal_admin key
  |-- get_customer_summary
  |-- search_orders
  |-- get_invoice_status
  |-- check_inventory
  +-- list_employee_records

No scoped key exposes run_sql or delete_order in the read-only service scope.
```

このモデルでは、ツール説明をセキュリティ上重要な内容として扱う。パートナー用キーは、意図的な業務公開でない限り、従業員レコード用ツールが存在することを知るべきではない。

3. 公開パターン比較

パターン	発見可能な表面	呼び出し可能な表面	スキーマ・業務プロセス露出	認可モデル	主なリスク
直接SQLツール	生SQLツール、スキーマメモ、例、エラー	DBロールとラッパーが許すSQL	高: テーブル、列、JOIN、エラー	DBロールとラッパーチェック	プロンプトインジェクションがDB誤用になる
汎用CRUDツール	テーブル・リソース形状のツール	APIキーごとの広範なテーブル操作	中から高: リソース名とフィールド名	APIエンドポイントとオブジェクトチェック	CRUD表面が業務上の必要を超える
広範なAPIカタログ	多数の内部APIと説明	APIキーまたはユーザー次第	高: 内部ワークフローと管理機能	APIゲートウェイとサービス認可	カタログが運用構造を漏らす
ロールフィルタ済みカタログ	ロール単位のツール・エンドポイント	ロール単位の操作	中: 許可された業務ツールのみ	発見フィルタと実行時認可	スコープずれ、広すぎるロール
CapabilityベースMCP	呼び出し元単位の業務Capability	明示的なCapability呼び出しのみ	低め: スキーマでなく業務操作	APIキースコープ、発見フィルタ、実行時認可、エージェントポリシー	粗いCapabilityが広範アクセスを再現する

4. 評価基準: 発見と実行

評価軸	直接SQLツール	汎用CRUDツール	広範なAPIカタログ	ロールフィルタ済みカタログ	CapabilityベースMCP
ツール発見の最小化	弱い	弱から中	弱い	強め	強め
実行時の最小権限	DBロールが非常に狭い場合を除き弱い	エンドポイントが狭ければ中	API次第で中	強め	Capabilityが狭ければ強め
スキーマ・業務プロセス露出	高	中から高	高	中	低め
パラメータ単位制約	弱から中	中	中	中	強め: Capability別検証
出力allow-list	外部または後処理が多い	エンドポイント別	サービス別	サービス別	エージェントが強制
行数・バイト数・タイムアウト	ラッパーとDB次第	API次第	API次第	API次第	実行ポリシー境界
APIキー・ID単位認可	DBロールまたはラッパー	APIキー・ユーザー	APIキー・ユーザー	ロール・キースコープ	Capabilityスコープキーと実行時チェック

4. 評価基準: セキュリティと運用

評価軸	直接SQLツール	汎用CRUDツール	広範なAPIカタログ	ロールフィルタ済みカタログ	CapabilityベースMCP
プロンプトインジェクション影響範囲	高	中から高	中から高	低め	正しく実装されれば低め
業務操作単位の監査性	弱い: クエリ・セッションログ	中: テーブル・リソースログ	中: エンドポイントログ	強め	強め: Capability名、呼び出し元、状態
失効とキー ローテーション	DB認証情報またはラッパーキー	APIキー	APIキー	スコープ済みAPIキー	CapabilityスコープAPIキー
運用複雑性	初期は低いがリスク高	中	中から高	中から高	中から高
レガシー適合性	露出が危険	APIモデリングが必要	広すぎることが多い	適切にスコープすれば良い	読み取り中心の運用に強い
MCP・ツール呼び出し適合性	低い	使えるが広い	使えるがノイズが多い	より良い	明示的な名前付きツールに適合
侵害されたエージェントの失敗モード	DB権限が露出	広範なリソース呼び出し	広範なAPI偵察	スコープ内誤用	スコープ内誤用と残存データリスク

5. 評価方法

評価は例示的な脅威モデルシミュレーションであり、形式的な証明ではない。完全なMCPサーバーは実装せず、本番データベースにも接続しない。

モデルは`src.zip`配下に置く。

- `evaluation-model.json`は架空の業務システム、ツール、ロール、公開パターン、シナリオ、定性的な結果を定義する。
- `evaluate-scenarios.mjs`はそのモデルから`scenario-outcomes.md`を生成する。
- `scenario-outcomes.md`は、本書で使うロール別発見結果とシナリオ結果を記録する。

架空のツールには、`get_customer_summary`、`search_orders`、`check_inventory`、`get_invoice_status`、`list_employee_records`、`run_sql`、`delete_order`を含めた。ロールは`sales_partner`、`support_agent`、`internal_admin`、意図的に危険な`over_permissioned_key`である。

この方法は権限と影響範囲を評価するものであり、性能や形式的な悪用可能性を測るものではない。

5. 評価シナリオ

ID	シナリオ
S1	sales partnerが `tools/list` を呼び出す。
S2	support agentが `tools/list` を呼び出す。
S3	侵害されたエージェントが管理者専用ツールを発見しようとする。
S4	プロンプトインジェクションが無関係な業務ツールの呼び出しを求める。
S5	プロンプトインジェクションが利用可能な全ツールの開示を求める。
S6	プロンプトインジェクションが生SQLツールの呼び出しを求める。
S7	正当なユーザーが許可済み業務Capabilityを有効なパラメータで呼び出す。
S8	ユーザーが許可上限を超える行数を要求する。
S9	ユーザーが出力allow-list外のフィールドを要求する。
S10	APIキーが全ツールへ過剰に権限付与されている。

定性的なラベルとして、`High risk`、`Medium risk`、`Lower risk`、`Blocked by design`、`Blocked if correctly implemented`、`Depends on implementation`、`Allowed`、`Out of scope` を用いる。

6. 評価結果: ロール別ツール発見

ロール / APIキー	期待されるツール	発見すべきでないツール	フィルタしない場合のリスク
`sales_partner`	`get_customer_summary`, `search_orders`	`check_inventory`, `get_invoice_status`, `list_employee_records`, `run_sql`, `delete_order`	パートナーがサポート、HR、生SQL、書き込みCapabilityの存在を知る。
`support_agent`	`get_customer_summary`, `search_orders`, `get_invoice_status`, `check_inventory`	`list_employee_records`, `run_sql`, `delete_order`	サポート文脈で管理者、生SQL、破壊的ツールが見える。
`internal_admin`	`get_customer_summary`, `search_orders`, `get_invoice_status`, `check_inventory`, `list_employee_records`	`run_sql`, `delete_order`	管理者スコープが生DBまたは破壊的権限へ広がる。
`over_permissioned_key`	全ツール	なし	最小権限は実質的に失敗し、侵害されたエージェントが全カタログを発見・呼び出しできる。

発見フィルタリングが重要なのは、ツール名と説明が、実行が後で拒否される場合でも業務構造を開示し得るためである。

6. 評価結果: シナリオ結果

シナリオ	直接SQLツール	汎用CRUDツール	広範なAPIカタログ	ロールフィルタ済みカタログ	CapabilityベースMCP
S1: sales partnerが `tools/list`	High risk	High risk	High risk	Lower risk	Lower risk
S2: support agentが `tools/list`	High risk	High risk	High risk	Lower risk	Lower risk
S3: 管理者専用ツールを発見	High risk	High risk	High risk	Blocked if correctly implemented	Blocked if correctly implemented
S4: 注入が無関係なツールを呼ぶ	High risk	Medium risk	Medium risk	Blocked if correctly implemented	Blocked if correctly implemented
S5: 注入が全ツールを開示要求	High risk	High risk	High risk	Lower risk	Lower risk
S6: 注入が生SQLを呼ぶ	High risk	Blocked by design	Depends on implementation	Blocked if correctly implemented	Blocked by design
S7: 正当な許可済みCapability	Allowed	Allowed	Allowed	Allowed	Allowed
S8: 行数上限を超える要求	High risk	Depends on implementation	Depends on implementation	Depends on implementation	Blocked if correctly implemented
S9: 未許可フィールド要求	High risk	Medium risk	Medium risk	Depends on implementation	Blocked if correctly implemented
S10: 過剰権限キー	High risk	High risk	High risk	High risk	High risk

6. 評価結果: リスク比較

リスク領域	直接SQLツール	汎用CRUDツール	広範なAPIカタログ	ロールフィルタ済みカタログ	CapabilityベースMCP
発見可能な権限	高	高	高	低め	低め
呼び出し可能な権限	高	中から高	中から高	スコープ済みなら低め	Capabilityが狭ければ低め
機密性失敗	高	中から高	中から高	中	低め、ただし消えない
完全性失敗	書き込み許可時は高	書き込みツール公開時は高	書き込みAPI公開時は高	書き込みフィルタ時は低め	読み取り専用スコープでは低め
プロンプトインジェクション影響	高	中から高	中から高	低め	低め
スキーマ偵察	高	中	中から高	低め	低め
監査の曖昧さ	高	中	中	低め	低め
キー過剰権限の影響	高	高	高	高	高

最小権限のMCP設計では、発見と実行の両方をフィルタする必要がある。`tools/call`だけをフィルタするとカタログメタデータが露出する。`tools/list`だけをフィルタしても、隠されたツールが名前指定で呼べるなら不十分である。

6. 評価結果: リスクの移動

リスク	広範なMCP公開での所在	最小権限CapabilityベースMCPでの所在	残存管理策
生DB権限	AIランタイム、SQLツール、スキーマプロンプト	オンプレミスエージェントとローカルDBロール	ホスト堅牢化、シークレット管理、DB最小権限
ツール偵察	全カタログと説明	呼び出し元でフィルタされたCapabilityカタログ	説明文をセキュリティ上重要な内容としてレビュー
未認可操作	エージェントが無関係なツールを見て呼ぶ	ゲートウェイとエージェントの認可境界	発見フィルタと実行時認可
過剰出力	テーブル、フィールド、広範なAPIレスポンス	Capabilityの出力allow-list	フィールド、行数、バイト数のレビュー
プロンプトインジェクション	モデルが広範な呼び出しへ誘導される	モデルは付与済みCapabilityだけ誘導できる	外部強制、監視、必要に応じた人の確認
悪い設定	広いAPIキーが多数のAPIを公開	過剰権限キーが多数のCapabilityを公開	キースコープ、ローテーション、アクセスレビュー
機密ログ	プロンプト、SQL、パラメータ、API結果	ゲートウェイとエージェントログ	生データ、SQL、認証情報、トークン、スタックトレースを避ける

リスクは消えない。MCPクライアントやモデル層から、ゲートウェイ認可、Capability定義、エージェント実行ポリシー、運用へ移動する。

6. 評価結果: 解釈

評価が支持する主張は限定的である。CapabilityベースMCPアクセスは、発見フィルタリング、実行時認可、パラメータ検証、出力allow-list、行数制限、ログ規律が正しく実装されている場合に、露出する権限を減らせる。

これは、このアーキテクチャが絶対的に安全であることを示すものではない。広すぎるCapabilityはデータを漏らし得る。単一の過剰権限APIキーは慎重なツール設計を無効化し得る。オンプレミスエージェントホストが侵害されれば、データベースパスとローカル認証情報が露出し得る。不適切なログは機密業務データを漏らし得る。

主な利点はガバナンスの明確さである。「エージェントにデータベースアクセスを許すか」ではなく、「この呼び出し元に、特定の業務Capability、特定のパラメータ、特定の出力フィールド、明示的な制限で発見・呼び出しを許すか」をレビューできる。

7. 考察（1/2）

`tools/list`のフィルタリングは`tools/call`のフィルタリングと同じくらい重要である。ツール説明は、実行がブロックされる場合でも業務構造を漏らし得る。`list_employee_records`、`refund_invoice`、`delete_order`のようなカタログは、そのワークフローが存在することをモデルとユーザーに伝える。

広範なAPIカタログはAIエージェントを圧倒することもある。モデルが見る無関係なツールが多いほど、誤ったツールを選ぶ機会や、悪意ある指示に従って意図しないワークフローへ進む機会が増える。

Capability定義は重要なセキュリティ境界になる。Capabilityは業務操作を表すのに十分狭くなければならず、データベースアクセスを再現するほど広くてはならない。多数のフィールドと高い行数上限を持つ`export_all_customers`のようなCapabilityは、実質的には最小権限ではない。

読み取り専用スコープは破壊的リスクを大きく下げる。ただし、機密性リスクは取り除かない。

7. 考察（2/2）

MCPは相互運用性を高めるが、相互運用性は安全な公開を自動的に意味しない。プロトコルはツールと呼び出しを記述できるが、どのツールをどの呼び出し元へ公開すべきかは業務側の設計判断である。

分析ワークロードでは、統制されたデータウェアハウス、BIセマンティックレイヤー、または管理された分析環境の方が適する場合がある。特に、柔軟なアドホック探索、多数のテーブルをまたぐ広範なJOIN、バッチ分析、大量の履歴データを使う機械学習、成熟した人間向け探索ガバナンスが必要な場合である。

運用AIエージェントでは、明示的な業務Capabilityの方が、生データベースアクセスや汎用CRUDツールより統制しやすいことが多い。運用エージェントが必要とするのは、顧客サマリー、注文検索、請求ステータス、在庫可用性、チケット文脈のような、境界のある問いへの境界のある答えである。

このアーキテクチャは、システムが制約を強制する代わりにモデルの自己制限を信頼した場合に失敗する。

8. 実務上の示唆（1/2）

レガシーシステムへAIエージェントを接続する企業は、読み取り専用MCPツールから始めるべきである。ツールはデータベーステーブルではなく、業務意図を中心に定義する。ツールは、データベース構造ではなく、呼び出し元に許可された操作を表すべきである。

呼び出し元のIDまたはAPIキースコープに基づいてツール発見をフィルタする。実行時にも認可を再度強制する。呼び出し元は`tools/list`で返されなかったツールでも、名前を指定して`tools/call`を試みられるため、実行時チェックは必要である。

入力検証、実行ポリシー、プリペアドSQL、行数制限、バイト数制限、タイムアウト、出力allow-listを使う。生SQLツール、生スキーマアクセス、広範なCRUDツール、データベース認証情報をAIランタイムやゲートウェイへ公開しない。

CapabilityスコープのAPIキーを使う。アクセス要件が変わった場合は、キーをローテーションまたは失効する。

8. 実務上の示唆（2/2）

機密業務データをログに残さない。ゲートウェイログは、第二のデータストアにならずに運用に役立つべきである。Capabilityまたはツール名、呼び出し元ID、ステータス、エラーコード、リクエストID、レイテンシを監視する。

プロバイダークラウド側のログには、生SQL、データベース接続文字列、認証情報、Authorizationヘッダー、APIキー、トークン、完全なパラメータ値、完全な結果セット、スタックトレース、データベースドライバ詳細を残さない。

MCPツール説明をセキュリティ上重要な内容としてレビューする。説明文は、内部インシデントメモ、秘密のワークフロー名、管理機能、認証情報、スキーマ詳細を漏らさずに、モデルが許可された操作を選ぶ助けになるべきである。

パートナー、サポート、内部管理者のアクセスプロファイルを分ける。最も安全なカタログは、呼び出し元が実際に必要とするカタログである。

9. 制限事項と今後の課題（1/2）

この評価は例示的であり、完全なセキュリティ監査、正式な脅威モデル、ペネトレーションテスト、レッドチーム評価の代替ではない。実験は完全なMCPサーバーを実装せず、すべてのMCPクライアントを対象にしない。

比較は、すべてのデータベースエンジン、ドライバ、ORM、APIゲートウェイ、エージェントフレームワーク、IDプロバイダ、デプロイトポロジ、ログ基盤を網羅しない。Capability定義とツール説明がレビューされ、正しく実装されていることを前提とする。

本書は主に読み取り中心アクセスに焦点を当てる。書き込み操作には、追加の制御、承認フロー、冪等性、トランザクション設計、変更ログ、ロールバック戦略、より強い監査が必要である。

人間の運用プロセス、インシデント対応、アクセスレビュー、性能やレイテンシのベンチマークは十分に評価していない。

9. 制限事項と今後の課題（2/2）

今後の課題として、実際のMCPクライアント、実世界のプロンプトインジェクションコーパス、複数のデータベースエンジン、異なるAPIゲートウェイ、レッドチームテスト、正式な脅威モデリング、書き込み操作の安全性を評価する必要がある。

さらに、`capability.yaml`、OpenAPI説明、MCPツール定義、ゲートウェイ認可ポリシー、エージェント側実行ポリシーの間で、Capabilityカタログが時間とともにどのようにレビューされ、ドリフトがどう検出されるかも検討すべきである。

CapabilityベースMCP公開は、ツール定義が広すぎてデータベースアクセスを事実上再現する場合や、成熟したガバナンスのもとで人間の分析者が探索的アクセスを必要とする場合には最適ではない。

したがって結論は条件付きである。この設計は、Capabilityモデル、認可層、エージェント実装、運用制御が正しい場合に、露出する権限を減らし、監査性を高められる。

10. 参考文献

1. Model Context Protocol, "Tools", https://modelcontextprotocol.io/specification/2025-06-18/server/tools
2. Model Context Protocol, "Authorization", https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization
3. OWASP GenAI Security Project, "OWASP Top 10 for Large Language Model Applications", https://owasp.org/www-project-top-10-for-large-language-model-applications/
4. OWASP API Security Project, https://owasp.org/www-project-api-security/
5. NIST, "SP 800-207: Zero Trust Architecture", https://www.nist.gov/publications/zero-trust-architecture-0
6. OpenAPI Initiative, "API Endpoints", https://learn.openapis.org/specification/paths.html
7. PostgreSQL Documentation, "Row Security Policies", https://www.postgresql.org/docs/17/ddl-rowsecurity.html

これらの参考文献は、プロトコル用語、認可の枠組み、LLMおよびAPIセキュリティリスク、ゼロトラスト原則、APIカタログ用語、データベースの行レベルセキュリティの背景を確認するために用いた。本書の例示的評価を認証または保証するものではない。

最後に

本分析は、最小権限MCP設計が絶対的に安全であるとは主張しない。

示しているのは、レガシーシステムへの運用AIアクセスに対する、より狭く監査しやすい権限モデルである。この設計は、露出する権限を減らし、発見可能なツールを制限し、呼び出し可能な表面を狭め、プロンプトインジェクションの影響範囲を減らし、認可境界を明示し、業務操作単位の監査性を高め得る。

これらの効果は、正しいCapability設計、ツールフィルタリング、実行時認可、出力制御、ログ規律、キー管理、オンプレミスエージェント環境の運用に依存する。

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年6月5日