ViewLegacy.

暴走するAIエージェントの被害範囲を制限するアーキテクチャ設計

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8月3日

本書は、AIエージェントが誤った判断をしたとき、周囲のシステムが被害を限定的かつ監査可能な範囲に抑えるには、どのようなアーキテクチャが必要かを検討する。

1. エグゼクティブサマリー（1/2）

AIエージェントの挙動が、すべてのプロンプト、コンテキスト、障害、再試行、実行環境の侵害において正しいとは仮定できない。本書でいう「暴走」には、呼び出しの反復、誤ったツールやパラメータ、未許可操作の試行、過剰取得、プロンプトインジェクションに起因する誤用、意図しない書き込み、複数ツールを連鎖させた誤用を含む。モデルが文字どおり自律的または悪意を持つという意味ではない。

本書は、DB認証情報の直接付与、無制限SQL実行、広範なCRUDまたは内部API、フィルタされない大規模MCPツールカタログ、ロールでフィルタしたツールと実行時認可、ゲートウェイとオンプレミス側のポリシー enforcementを組み合わせたケイパビリティベース実行の6方式を比較する。

主要な結論は、プロンプトやモデル alignment は有用な指針だが、決定論的なセキュリティ境界ではないということである。

1. エグゼクティブサマリー（2/2）

ケイパビリティベースの封じ込めは、生のシステムアクセスを定義済み業務操作に置き換え、モデル外部で認可、検証、資源制限、出力フィルタ、レート制御、監査を行うことで、実行可能な面を狭めうる。最も強い構成は、呼び出し元別のツール発見、ゲートウェイでの呼び出し単位認可、独立したオンプレミス allow-list、入力と出力の契約、プリペアドステートメント、読み取り専用DBアカウント、行数・バイト数・時間の制限を組み合わせる。

ただし結論には限界がある。読み取り専用は破壊的リスクを大幅に下げるが、機密性、推論、費用、可用性のリスクは残る。オンプレミスホストの侵害、過剰権限キー、広すぎるケイパビリティ、payloadを含むログ、安全でない書き込み操作は境界を弱める。

広範な分析には統制されたDWHやBIセマンティックレイヤーが適する場合があり、特権管理者や低遅延ローカル自動化には別方式が必要になりうる。封じ込めはリスクを減らすが、取り除かない。

2. 背景（1/2）: エージェント連携が変えるリスク

従来のアプリケーションは、主として開発者が記述した経路を通る。AIエージェントはツールとパラメータを動的に選び、出力を次の呼び出しへ連鎖させ、エラー後に再試行できる。正しいendpointを誤った顧客に使う、許可されたenumを不適切な業務状態で使う、有用な検索を数百回繰り返すなど、技術的に妥当でも運用上不適切な要求が生じる。

ツール選択は、曖昧なユーザー意図、誤推論、侵害されたコンテキスト、直接のプロンプトインジェクション、取得コンテンツに埋め込まれた命令の影響を受けうる。OWASPは、プロンプトインジェクションをモデル挙動を意図しない形で変える入力として説明し、RAGやfine-tuningでも完全には緩和できないとしている [1]。Excessive Agencyは、過剰な機能、権限、自律性を損害の根本要因として扱う [2]。

2. 背景（2/2）: 広範なツールは誤判断を増幅する

広範なツールは、多数の判断を一つの権限付与に圧縮する。DB認証情報はアカウントの全権限、raw SQLは言語、汎用CRUDはオブジェクトとmethod、大規模MCPカタログは多数の操作とschemaを公開する。結果として、確率的なモデルが事実上のポリシー判断点になる。

レガシーシステムには、操作単位の認可、一貫したvalidation、上限付きpagination、構造化監査、呼び出し元別rate limitがない場合がある。1件の請求状態を必要とする取引先に対して、DB権限モデルは粗すぎることがある。

MCPツールはモデルが発見・選択できるよう設計されている。仕様は同時に、serverへ入力検証、アクセス制御、rate limit、出力sanitizeを求め、機微操作の確認も推奨する [3]。これらは周囲のシステムが実装すべき制御であり、ツール説明やsystem promptでは代替できない。

3. 比較対象の封じ込めアーキテクチャ（1/6）

パターン1: AIエージェントへDB認証情報を直接付与

エージェント実行環境がconnection stringなどのDB認証情報を持つ。アカウントが許可するschema metadataとデータを発見し、native queryを送信し、sessionを維持できる。パラメータと出力の自由度は主にDB engineとアカウント権限で決まる。

- **権限と書き込み:** DBアカウントに付与された全read/write権限
- **発見と呼び出し:** schema全体のobjectとnative DB操作
- **レート制御と監査:** 通常はDB側。共有アカウントでは呼び出し元が集約される
- **想定被害範囲:** 付与済みDB権限全体
- **モデル依存:** 最大。適切なquery作成と停止をエージェントに強く依存する

読み取り専用credentialでも、行数、バイト数、join、lock pressure、query cost、機微列は自動的に制限されない。

3. 比較対象の封じ込めアーキテクチャ（2/6）

パターン2: 無制限SQL実行ツール

エージェントが `run_sql(sql_text)` のようなツールを呼ぶ。credentialはtool host内に隠せるが、SQL言語そのものが実行可能面に残る。直接credentialとの違いは秘密の保管場所であり、必ずしも権限の広さではない。

- **権限と書き込み:** wrapperとDBアカウントが受理する全statement
- **発見と呼び出し:** 一つの小さなツールがquery/command言語全体を公開しうる
- **パラメータと出力:** parser、statement policy、result filter、資源制御がなければ極めて自由
- **レート制御と監査:** wrapperで可能だが、SQL記録は業務意図の監査粒度が弱い
- **想定被害範囲:** 任意SQLを受けるならDB直接アクセスに近い
- **モデル依存:** 非常に高い。「SELECTのみ」というpromptはenforcementではない

プリペアドステートメントは、statement shapeを固定したときにデータパラメータを保護する。任意SQL自体を安全にはしない。

3. 比較対象の封じ込めアーキテクチャ（3/6）

パターン3: 広範なCRUD APIまたは内部API

`list_table`、`update_record`、完全な内部API catalogなどの汎用endpointでSQLを構造化呼び出しに置き換える。入力syntaxと既存identity基盤は改善するが、多数のobject、method、field、workflowへ到達しうる。

- **権限と書き込み:** 公開された全endpointとHTTP method
- **発見と呼び出し:** 管理機能を含みうる広いresource inventory
- **パラメータと出力:** SQLより狭いが、filter、sort、field selector、bulk、paginationが自由な場合がある
- **レート制御と監査:** API gatewayでendpoint単位に実装しやすい
- **想定被害範囲:** 全公開APIと到達可能object
- **モデル依存:** endpointごとのcaller認可とbounded contractがなければ高い

OWASPは、すべての業務機能に対してdefault deny、明示grant、一貫したfunction authorizationを推奨している [4]。

3. 比較対象の封じ込めアーキテクチャ（4/6）

パターン4: フィルタされない大規模MCPツールカタログ

MCP serverが `tools/list` で業務、汎用、管理操作を含む多数のtoolを公開する。tool schemaは機械利用を改善する一方、混乱または侵害されたruntimeが発見・試行できる面も広げる。

- **権限と書き込み:** 呼び出せる全tool。副作用を持つものを含みうる
- **発見と呼び出し:** 全callerへname、description、input schemaを公開
- **パラメータと出力:** tool次第。raw SQLまたはexport tool一つでリスクを支配しうる
- **レート制御と監査:** 実装されていればMCP serverまたはgatewayで実施
- **想定被害範囲:** 公開された全toolの権限の和集合
- **モデル依存:** serverが文脈に適切なtoolだけをモデルが選ぶと仮定する場合は高い

発見filterは情報露出と誤選択を減らすが、認可ではない。callerは `tools/call` を直接送れる。

3. 比較対象の封じ込めアーキテクチャ（5/6）

パターン5: ロールfilterと実行時認可

serverはcallerのroleで許可されたtoolだけを返し、呼び出しごとに認可を再確認する。mappingとenforcementが正しければ、漏えいしたsales keyはtool名を推測してもadmin toolを呼べない。

- **権限と書き込み:** roleへgrantされたtoolの和集合
- **発見と呼び出し:** role scopeへ縮小。未許可の直接callを拒否
- **パラメータと出力:** 各tool次第。許可されたexportやgeneric searchは広い場合がある
- **レート制御と監査:** APIまたはMCP serverへ集中させやすい
- **想定被害範囲:** roleのtoolと各tool内部の自由度
- **モデル依存:** cross-roleでは低下するが、許可tool内部では残る

この方式は有意な改善だが、toolに行数、バイト数、field、timeout、業務状態の制約がなければ constrained executionとは同じではない。

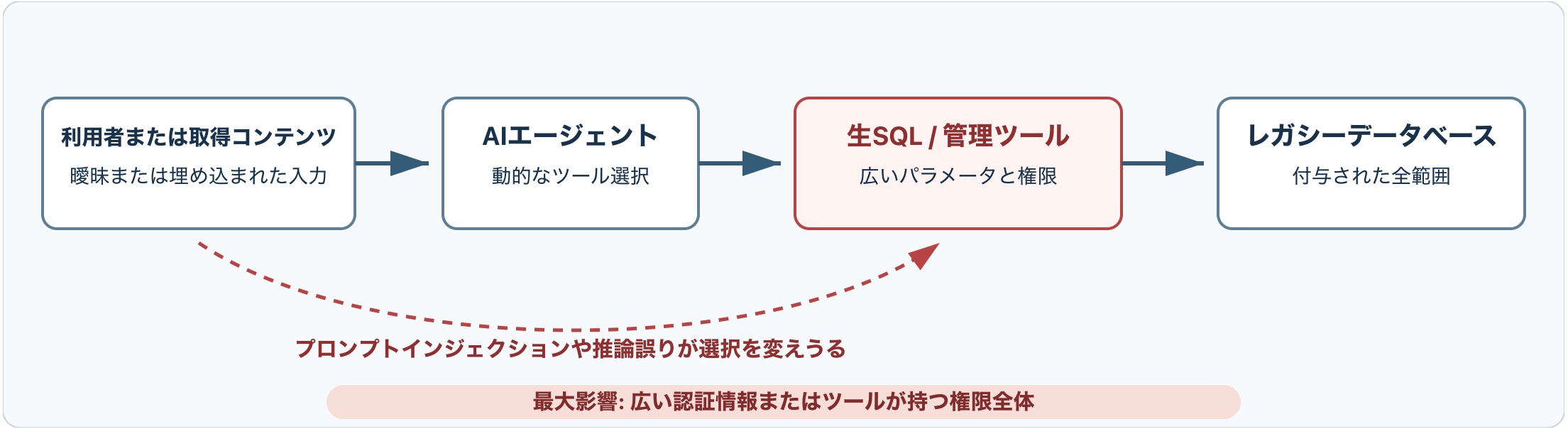
3. 比較対象の封じ込めアーキテクチャ（6/6）

パターン6: Gatewayとオンプレミス enforcementによるケイパビリティ方式

エージェントはtableやSQLではなく、`search_orders` などの業務ケイパビリティを呼ぶ。Gatewayはcaller認証、IP/rate policy、発見filter、API key scope確認、routingを行う。オンプレミスagentは未定義capabilityを独立して拒否し、入力を検証し、実行policyを適用し、定義済みprepared SQLを実行し、出力をfilterする。

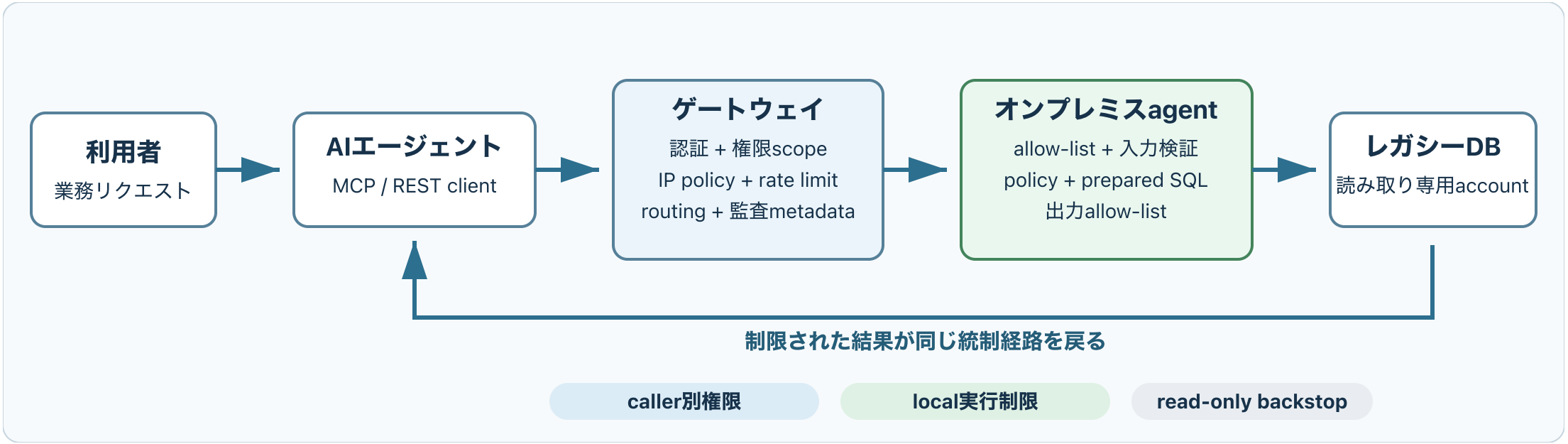
- **権限と書き込み:** 明示定義・認可されたcapability。初期scopeはread-only
- **発見と呼び出し:** caller別catalogとcall単位認可
- **パラメータと出力:** schema、enum、range、length、field、row、byte、timeで制限
- **レート制御と監査:** Gatewayとagent concurrency。payloadなしで業務操作を記録
- **想定被害範囲:** 正しく設計・実装された場合、認可capabilityと制限の内側
- **モデル依存:** 決定的境界がモデル外にあるため低い

3. 図1: 広い権限と大きな被害範囲



一度の誤選択が、複数table、tenant、業務process、read/write境界を横断しうる。taskより大きな権限grantの内部で、モデルに自己制限を求めている。

3. 図2: ケイパビリティベースの封じ込め

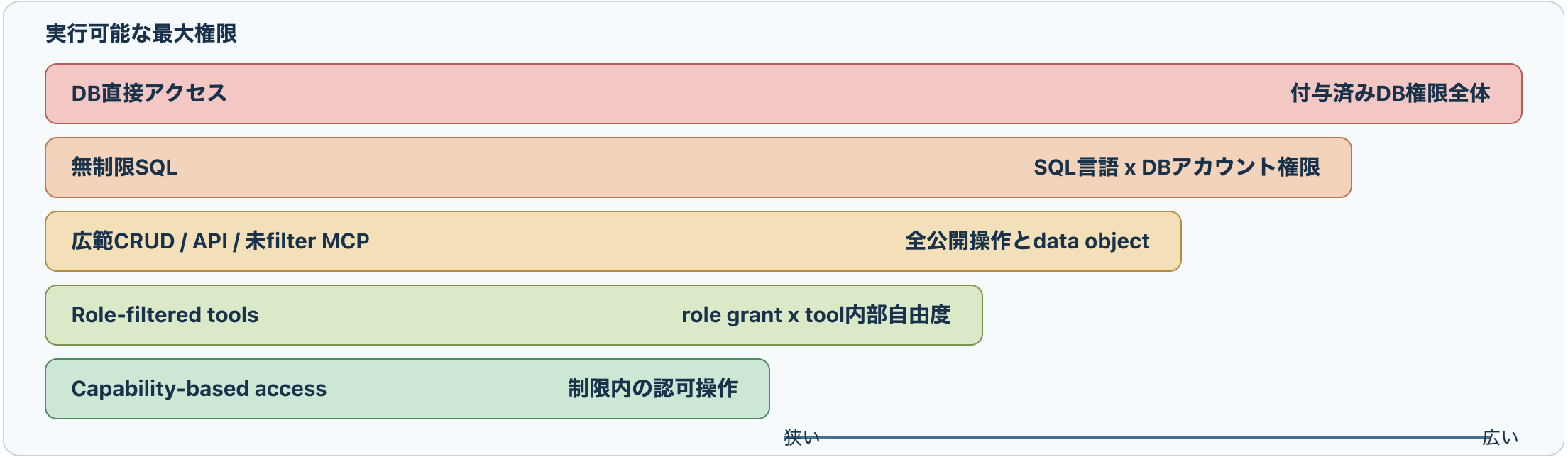


Gatewayはraw SQLやDB credentialを必要としない。オンプレミス境界はGatewayが新しい操作を発明できるとは信頼せず、未定義capabilityをlocalで拒否する。

3. 図3: 多層enforcementモデル

単一layerでは不十分である。discovery filtering、runtime認可、agent policy、DB権限は異なる役割を持つ。

3. 図4: アーキテクチャ別の最大影響



これは実行可能な最大権限の比較であり、障害確率の比較ではない。host侵害や設計不良capabilityは依然としてcritical impactを生みうる。

3. 比較: 権限と実行可能面

パターン	最大実行権限	発見可能操作	呼出可能操作	parameter自由度	output自由度	write露出	model自己制限
DB認証情報直接	DB account権限	schema、metadata、DB機能	native DB操作	極めて高い	極めて高い	account次第、しばしば広い	極めて高い
無制限SQL	受理SQL x DB権限	SQL toolと推測可能schema	任意の受理statement	極めて高い	極めて高い	parser/enforcementがなければ高い	極めて高い
広範CRUD/API	全endpoint/object	広いAPI inventory	公開method	高い	高い	create/update/deleteを含みやすい	高い
未filter MCP	全公開toolの和集合	全tools/list	caller別認可なしの全tool	tool次第	tool次第	catalog次第	高い
Role filter + runtime auth	role grant tools	role filter済み	role認可済み	tool次第	tool次第	role/tool次第	中程度
Capability方式	制限内の認可capability	caller scope	認可かつlocal定義のみ	低い/制限済み	明示field/limit	初期read-onlyではなし	低い

capabilityが汎用query言語、無制限export、広範admin wrapperになれば利点は失われる。

3. 比較: Enforcement、運用、障害範囲

パターン	rate制御場所	監査粒度	revoke	障害分離	legacy適合	運用複雑性	想定blast radius
DB認証情報直接	DB/network	共有DB identity、SQL	credential rotation	account/DB分離がなければ弱い	nativeだが侵襲的	初期低、governance高	DB権限全体
無制限SQL	tool host/DB	SQL statement	tool token + DB credential	弱い	広いDB対応	中	DB直接に近い
広範CRUD/API	API gateway	endpoint/object	token/key rotation	中	API coverage必要	中から高	全公開API
未filter MCP	MCP server/gateway	tool名	token/key rotation	tool次第	adapter必要	中	全公開tool
Role filter + runtime auth	MCP/API gateway	caller + tool	role/key更新	role間は改善	adapter必要	高	role grant tools
Capability方式	gateway + agent queue/timeout	caller + 業務capability + status	scoped key revoke/rotate	capability単位で強い、host共有	predefined SQLがlegacy DB向き	設計負荷最大	認可capability制限内

運用複雑性は現実のcostであり、policy、key scope、review、monitoring、local deploymentを継続的に正しく保つ必要がある。

4. 評価基準（1/2）: 権限とデータ移動

評価は、文書が何をすべきと書くかではなく、誤誘導または侵害されたagentが何を実行できるかを問う。

評価軸	評価で用いる問い
最大実行権限	現在のcredentialとtoolで到達できる最大action setは何か
破壊操作への露出	create、update、delete、DDL、副作用を実行できるか
data exfiltration面	どのrecord、field、tenant、join、exportを取得できるか
prompt injection blast radius	tool選択が乗っ取られた場合の最大権限は何か
parameter制約強度	type、enum、range、length、format、unknown fieldを強制するか
output最小化	業務目的に必要な返却fieldだけをallow-listするか
row/byte limit	requested valueだけでなく実際のresponseにも上限を強制するか
timeout enforcement	一要求がDB、agent、network資源を無期限に占有できるか

機密性、完全性、可用性をすべて扱う。read-onlyは強い完全性controlだが、完全なsecurity postureではない。

4. 評価基準（2/2）: Identityと運用

評価軸	評価で用いる問い
rate limit / burst control	caller/IPとdownstream capacityで反復を制限するか
capability scope認可	identityごとにdefault denyと明示grantを持つか
tool discovery filtering	tools/listがcallerの認可capabilityだけを返すか
業務操作単位の監査	payloadなしでcaller、intent、status、latencyを追跡できるか
revoke / key rotation	全callerのDB credentialを変えずに一callerを停止・縮小できるか
failure isolation	一layerの侵害や過負荷が別layerの制御を迂回するか
legacy互換性	legacy application sourceを変えずにcontrolを追加できるか
運用複雑性	policy設計、配備、test、review、incident responseの負荷は何か
AI自己制限への依存	prompt/model behaviorだけでenforceされる判断は何か

tool descriptionは誤用を抑制しうるが、server-side decisionだけがcallを拒否できる。

5. 脅威モデルと評価方法（1/5）: Actor

脅威モデルは、通常の失敗と侵害の両方を含む。

- 曖昧な依頼をする正当な利用者
- 誤ったtoolまたはparameterを選ぶAI agent
- 取得コンテンツに埋め込まれた直接・間接prompt injection
- user intentを無視する侵害済みagent runtime
- 未許可contextから使われる漏えいAPI key
- caller policyの迂回を試みる侵害済みgateway
- local secretとcodeへ到達する侵害済みon-prem agent host
- key scopeまたはcapability policyを変える悪意ある/誤ったadministrator
- 業務名が示すより広い不良capability定義

すべてのactorが悪意を持つとは仮定しない。同じ境界がretry bug、hallucinated parameter、盗難key、injected instructionを、その試行操作に応じて制限すべきである。

5. 脅威モデルと評価方法（2/5）: Trust Boundary

GatewayとOn-prem agentは絶対に信頼できるcomponentではなく、侵害されうるものとして扱う。影響は異なる。

- **Model-level guidance:** 選択とUXを導くが、決定論的policy boundaryではない
- **Gateway enforcement:** 認証、capability scope、discovery filter、IP/rate policy、routing、非payload metadata記録
- **Agent-side enforcement:** 操作語彙、input contract、raw SQL、prepared execution、limits、output contract、local detail errorを所有
- **Database permission:** 特にread-only accountによる最終least-privilege backstop
- **Operational monitoring:** payloadを集めずdenial、spike、error、latency、policy change、connection anomalyを検知

Gateway侵害でagent上の未定義操作を作れてはならない。Agent host侵害はlocal DB credentialと実行policyへ到達するため、より重大である。

5. 脅威モデルと評価方法（3/5）: 例示システム

simulationは架空のlegacy order management systemを扱う。操作は `get_customer_summary`、`search_orders`、`get_invoice_status`、`check_inventory`、`export_order_report`、`update_order_status`、`delete_order`、`run_sql`。key profileは `sales_partner`、`support_agent`、`internal_operator`、`admin` である。

6方式ごとに、filtered discovery、runtime authorization、on-prem capability allow-list、input validation、read-only、row/byte/time limit、output allow-list、rate limit、payload-free log、agent-side enforcementの有無を定義した。

local evaluatorは15 scenarioすべてに6方式分のoutcomeが一つずつあることを検証してから、scenario matrixとcontrol matrixを生成する。評価labelは **Critical**、**High risk**、**Medium risk**、**Lower risk**、**Blocked by design**、**Blocked if correctly implemented**、**Depends on implementation**、**Allowed**、**Out of scope** とした。

これは最大影響を比較する例示評価であり、形式的証明、完全なpenetration test、確率推定ではない。

5. 脅威モデルと評価方法（4/5）: Scenario S1-S8

ID	Scenario	主に検証する性質
S1	Prompt injectionが `run_sql` を要求	raw query言語を実行面から除去
S2	Prompt injectionがorder deleteを要求	read-only scopeとwrite除外
S3	許可searchを500回呼ぶ	rate、burst、concurrency、downstream load
S4	100,000行を要求	numeric validationとrow/byte ceiling
S5	taskに不要な機微fieldを要求	output allow-listとdata minimization
S6	sales partner agentが誤capabilityを選択	discovery filteringとruntime認可
S7	無効なorder status enumを渡す	決定論的schema validation
S8	漏えいsales keyでadmin capabilityをcall	capability-scoped credentialとdefault deny

S8では各patternの広範credentialを等価なaccess tokenとして解釈する。protocol syntaxではなくcredential blast radiusを比較するためである。

5. 脅威モデルと評価方法（5/5）: Scenario S9-S15

ID	Scenario	主に検証する性質
S9	侵害agent runtimeが全toolを列挙	caller-filtered `tools/list` とinventory露出
S10	侵害gatewayが未定義capabilityをcall	独立したagent-side allow-list
S11	on-prem agent host侵害	credential、SQL、runtime、endpoint trust
S12	正当callerがvalid parameterで許可read	intended functionの維持
S13	十分なreviewなしにwrite capability追加	capability設計とchange governance
S14	retry loopがavailabilityを低下	rate、timeout、queue、concurrency limit
S15	logが機微payloadを誤収集	logging boundaryとsecondary disclosure

scenario setは、推奨patternでも高い影響が残るdefeat caseを意図的に含む。信頼できる分析は、preferred architectureが失敗する場所も示す必要がある。

5. 例示ポリシー（1/2）

```yaml
capabilities:
  search_orders:
    description: "認可顧客の最近の注文を検索"
    sql: |-
      SELECT order_id, status, ordered_at
      FROM orders
      WHERE customer_id = :customer_id AND status = :status
      LIMIT :limit
    params:
      customer_id: { type: string, required: true, maxLength: 64 }
      status:
        type: string
        enum: ["open", "shipped", "cancelled"]
      limit: { type: integer, default: 50, minimum: 1, maximum: 200 }
```

操作名は業務意図を表す。固定statement shapeとprepared parameterにより、callerの権限からSQL grammarを除く。enum、length、numeric rangeが不正・過剰入力を実行前に拒否する。

5. 例示ポリシー（2/2）

```
policy:
  readonly: true
  timeout: 5s
  max_rows: 200
  max_bytes: 256KB
result:
  order_id: { type: string }
  status: { type: string }
  ordered_at: { type: string }
```

`readonly` はwrite statementを除外し、DB accountも独立してread-onlyにする。`timeout` はexecution occupancy、`max_rows` と `max_bytes` はquery estimateやcaller parameterが誤っていてもretrievalを制限する。`result` はoutput allow-listである。

これは完全なproduction configurationではない。caller-to-capability grant、IP policy、rate setting、secret storage、transport security、schema migration、monitoring、capability review workflowを省略している。

6. 評価結果: 封じ込めパターン比較

パターン	runtime制約	write露出	監査粒度	想定blast radius	主な残存risk
DB認証情報直接	別途構築しなければDB nativeのみ	account全scope	共有identity + SQL	付与DB権限全体	credential/runtime侵害がDB侵害になる
無制限SQL	wrapper次第、しばしば弱い	全受理statement	tool + SQL、業務意図弱い	DB account権限に近い	一つの言語toolが広いaccessを再現
広範CRUD/API	endpoint次第	広いcreate/update/delete	endpoint/object	全公開APIと到達data	auth drift、bulk、過剰field
未filter MCP	tool次第	公開write tool	tool名	全公開toolの和集合	無関係/admin tool発見・選択
Role filter + runtime auth	role間は強い、tool内は可変	role grant writes	caller + tool	role grant x tool内部自由度	広い認可toolとparameter/output自由度
Capability方式	capability/schema/row/byte/time/output	評価read-only scopeではなし	caller + 業務操作 + result metadata	制限内の認可capability	設計不良、host侵害、availability、leakage

「制限内」という結果は、allow-listと全enforcement layerが正しいことを前提とする。

6. Scenario結果（1/3）: 直接誤用と量

Scenario	Direct DB	Raw SQL	Broad CRUD/API	Unfiltered MCP	Role-filtered	Capability-based
S1 `run_sql` injection	Critical	Critical	Blocked by design	High risk	Blocked if correct	Blocked by design
S2 delete injection	Critical	Critical	High risk	High risk	Blocked if correct	Blocked by design
S3 500-call loop	High risk	High risk	High risk	High risk	Medium risk	Lower risk
S4 100,000 rows	Critical	Critical	High risk	High risk	High risk	Blocked if correct
S5 extra fields	Critical	Critical	High risk	High risk	Medium risk	Blocked if correct

`Blocked if correct` は保証ではなく、named deterministic controlが拒否すると期待され、testが必要という意味である。`Lower risk` もriskなしではない。許可callは資源を消費し、認可scope内の機微dataを返しうる。

6. Scenario結果（2/3）: Scopeと侵害

Scenario	Direct DB	Raw SQL	Broad CRUD/API	Unfiltered MCP	Role-filtered	Capability-based
S6 wrong capability	High risk	High risk	Medium risk	High risk	Blocked if correct	Blocked if correct
S7 invalid enum	High risk	High risk	Depends on impl.	Depends on impl.	Depends on impl.	Blocked if correct
S8 leaked sales key -> admin	Critical	Critical	High risk	High risk	Blocked if correct	Blocked if correct
S9 enumerate all tools	Critical	High risk	High risk	High risk	Lower risk	Lower risk
S10 compromised gateway -> undefined	Out of scope	Out of scope	Depends on impl.	Depends on impl.	High risk	Blocked if correct

S10はgateway認可とagent-side enforcementを区別する。Capability architectureではgatewayはSQL定義や新capability発明を行えない。local agentがloaded policyにないnameを拒否し、gateway compromiseを有効local capabilityとそのlimit内に制約する。

6. Scenario結果（3/3）: 残存・運用risk

Scenario	Direct DB	Raw SQL	Broad CRUD/API	Unfiltered MCP	Role-filtered	Capability-based
S11 on-premise host compromised	Critical	Critical	Critical	Critical	Critical	Critical
S12 valid allowed read	Allowed	Allowed	Allowed	Allowed	Allowed	Allowed
S13 unsafe write added	High risk	High risk	High risk	High risk	High risk	High risk
S14 retry-loop availability	High risk	High risk	High risk	High risk	Medium risk	Lower risk
S15 payload captured in logs	High risk	High risk	High risk	High risk	High risk	High risk

推奨architectureも全scenarioで優位ではない。Host compromiseはlocal secret boundaryを越える。Unsafe capabilityはsecurity boundary自体を変える。Payload loggingはdurableなsecondary exposureを作る。これらにはhost hardening、change control、log schema enforcement、incident responseが必要である。

6. Blast Radius比較matrix

失敗・誤用	Direct DB / raw SQL	Broad API / unfiltered MCP	Role-filtered tools	Capability-based containment
Wrong tool	query言語/広いaccountが到達可能	任意の公開endpoint/toolが動作	role grant内	scoped local capability内
Wrong parameter	任意predicate/value	endpoint schema次第	tool次第	type/enum/range/length/formatを強制
Excessive retrieval	DB-scale read可能	pagination/export次第	tool次第	row + byte + output field ceiling
Unintended write	account/toolが広くwrite可能	公開write methodへ到達	role grant writeへ到達	初期read-onlyで除外、DB account backstop
Repeated call	loadは主にDBが吸収	gatewayが助ける場合あり	rate controlがburst低減	rate + timeout + bounded queue/concurrency
Leaked credential	credential全権限	tokenの広いcatalog scope	role scope	explicit capability scope、未指定はzero
Gateway compromise	該当なし	backend authorityはgateway trust次第	広いbackend toolに届く場合	agentがundefined name拒否、local limit適用
Agent host compromise	DB/runtime侵害相当	adapter/runtime侵害相当	adapter/runtime侵害相当	critical residual: credential/policy露出

最大到達影響の比較であり、発生確率やcontrol assuranceではない。

6. Control-to-Risk Mapping（1/2）

Control	低減するrisk	解決しないもの	誤設定時のfailure
Capability allow-list	undefined/generic operation試行	broad defined capability、host compromise	allow-all/wildcardが広いaccessを再現
API-key capability scope	leaked key/cross-role authority	allowed capability内のabuse	over-permission keyがblast radius拡大
Zero permission default	accidental grant/config omission	explicit bad grant	default allowで新capabilityを自動公開
Filtered `tools/list`	inventory disclosure/wrong-tool selection	guessed direct call	unfiltered listがadmin/write tool公開
Runtime authorization	name既知でもunauthorized call	bad authorized action	missing checkでdiscovery filterが見かけだけ
Input validation	invalid type/enum/range/length/format	semanticに誤ったvalid input	coercion/unknown fieldがintent迂回
Prepared statements	data parameter経由SQL injection	unsafe fixed SQL、host compromise	string concatでinjection復活
Read-only + DB account	destructive SQL/unintended write	confidentiality、inference、availability	writable accountでapp-only claim無効

Controlは合成される。DB accountがstatement policyを、runtime authがdiscovery filterを、explicit key scopeがuser intentを補完する。

6. Control-to-Risk Mapping（2/2）

Control	低減するrisk	解決しないもの	誤設定時のfailure
Row + byte limits	bulk extraction/oversized response	repeated small read、allowed sensitive field	高すぎるlimitは名目だけ
Timeout	long query/resource occupancy	多数の短い高cost call	cancellationなしでDB work継続
Output allow-list	unnecessary column disclosure	approved fieldのsensitive value、inference	permissive result schemaがhidden column露出
Rate + burst limits	retry loop/high-frequency abuse	distributed source、slow expensive call	disabled/shared bucketでamplification
IP restrictions	unexpected originからの利用	compromised allowed host、proxy mistake	trusted-header errorでspoofing
Payload-free gateway logs	secondary cloud/log disclosure	sensitive metadata/local logs	debug payload loggingがclaim破壊
Local detailed errors	cloud leakageなしtroubleshooting	local host/log compromise	raw error upstreamでSQL/value露出
Single-agent connection	competing/duplicate connector	malicious live agent、per-call abuse	weak agent authでtakeover
Explicit capability review	unsafe scope/write introduction	unknown implementation flaw	unreviewed changeがboundary再定義

Single-agent connectionは運用分離であり認可の代替ではない。機微変更にはauthenticated rolloutとmonitoringが必要である。

6. Riskの移動: Controlは場所を変えるが現実は消えない

Risk	Broad accessでの場所	Capability方式での場所	必要な残存control
SQL/credential	agent runtimeまたはbroad tool host	on-prem agentのみ	host hardening、secret protection、read-only DB
Authorization	model choiceまたはcoarse credential	gateway key scope + per-call check	default deny、review、revocation test
Operation definition	SQL/API/tool catalog	`capability.yaml`	security-sensitive code review/lint
Data minimization	agent queryまたはbroad response	agent-side result allow-list	field classification/contract test
Availability	legacy DBがbroad workloadを吸収	gateway/agentがcontrol共有	rate benchmark、concurrency budget、monitoring
Error detail	tool/API responseとcentral log	local detailed error log	local access/rotation/retention/incident handling
Gateway compromise	backend credential/logic露出可能	SQL定義不可、valid capability callは可能	agent allow-list/limit/transport auth/anomaly response
Agent host compromise	adapterまたはDB boundary	最大impactのlocal trust zone	OS/network hardening、EDR、least privilege、recovery

Architectureはtrustをmodelbehaviorからexplicitcontrolへmoveするが、capability設計とon-prem hostへcritical trustをconcentrateする。

6. 結果の解釈と残存リスク

例示matrixでは、tool言語の除去、caller scope、input/output bounding、独立enforcementで最大の改善が見られる。Prompt injectionはtool選択へ影響し続けるが、危険操作が存在せず、許可操作が狭いcontractを持てば、最大実行影響を減らせる。

Availabilityは依然確率的である。Rate limit、timeout、bounded request queue、single active agent connectionはamplificationを減らし、failure isolationを改善する。しかし許可queryが安価とは保証せず、distributed abuseやlegitimate burst下のDB availabilityも保証しない。

Confidentialityも残る。Repeated bounded readはdataを蓄積でき、approved fieldも機微であり、timing/existence checkによるinferenceもありうる。複数capabilityの相関で一response以上を推測できる。Monitoring、caller budget、semantic authorization、定期reviewが必要である。

7. 考察（1/4）: Governanceと柔軟性

狭いtool setはreasoning、test、revoke、auditが容易だが柔軟性は低い。Capability品質が実際のboundaryを決める。`get_invoice_status(invoice_id)` はgovernableだが、`query_any_business_data(query)` は安全そうな名前でraw accessを再現しうる。

Role filteringとcapability scopingは異なる問題を解く。Role filteringはvisible/callable toolを減らす。Capability constraintはauthorized tool内部の可能行為を制限する。Agentはrole boundaryを越えずにallowed toolを誤用できるため、両方が必要である。

Output minimizationも重要である。Input validationでwell-formedなrequestでも、broad responseが個人・金融・商用fieldを余分に返しうる。Result contractはpositive allow-listとし、queryとcontractが不一致ならsafe failureにする。

7. 考察（2/4）: Read-Onlyと可用性

Read-onlyはdelete、意図しないstate transition、schema changeなど多くの破壊結果を防ぐ。一方でdata leakage、inference、高cost scan、readによるlock contention、credential theft、resource exhaustionを防がない。したがって強い一層ではあるが、全risk結論ではない。

Rate limitingはloopとrepeated-call amplificationを減らせる。OWASPはinteraction frequency、records per response、payload size、execution timeの制限を推奨する [5]。ただし厳格すぎるlimitは季節的burstやincident responseを拒否する。可能ならcaller/operation単位とし、legacy capacityに結び付け、generic defaultをコピーせずtestする。

Asynchronous workにはjob identity、quota、cancellation、idempotency、durable statusが必要になりうる。Synchronous request limitだけではlong-running exportやmulti-step workflowを安全に統制できない。

7. 考察（3/4）: 侵害境界とログ

Gateway compromiseとagent-host compromiseの結果は異なる。Gatewayはtransient trafficを観測しvalid capabilityを呼べるが、SQLやDB credentialを持つべきではなく、agentがundefined operationを拒否すべきである。On-prem host侵害ではDB credential、raw SQL、capability definition、local detailed error、live resultが露出しうる。これはcritical residual riskである。

Logは封じ込めを密かに無効化しうる。Auditabilityにはcapability名、caller identity、status、error code、timestamp、latencyが必要だが、request parameterやresponse bodyは不要である。OWASP logging guidanceはaccess token、DB connection string、secret、機微な個人・商用dataの除外を推奨する [6]。

Detailed DB errorはlocalかつaccess-controlledにし、upstreamはSQL text、credential、driver detail、input valueを含まないstable public error codeを使う。

7. 考察（4/4）: 書き込みと代替方式

Writeにはreadより強いsafeguardが必要である。Change authorization、高impact actionへのhuman approval、idempotency key、transaction boundary、concurrency control、rollback/compensation、証拠性の高いaudit trailである。Retry可能な `update_order_status` はtransitionを二重適用してはならず、deleteはreversible workflowへ置き換える場合がある。

Capability方式は、legacy systemに対して狭く反復可能な業務操作を行うoperational agentに一般に適する。次の場合は最適でないことがある。

- analystがad hoc explorationやbroad joinを必要とする。governed warehouseやBI semantic layerが適する
- batch analyticsやMLが大規模historical datasetを必要とする
- human administratorがhardened admin consoleで広いinteractive controlを必要とする
- capabilityが広がりraw system accessを実質的に再現する
- low-latency local automationが厳格統制環境内のdirect executionを必要とする

8. 実践的アーキテクチャ指針（1/3）: 最初に権限を定義する

Agentはいずれ誤った、または敵対的に影響された判断をすると仮定する。Tool公開前に、confidentiality、integrity、availability、cost、運用scopeについて最大許容impactを定義する。

Read-only capabilityから始め、tableではなくbusiness intentを中心に定義する。`select_invoice_rows` ではなく `get_invoice_status`、generic inventory CRUDではなく `check_inventory` とする。Raw SQL、raw schema access、DB credential、不要なbulk admin tool、write functionを除外する。

Default denyを用いる。Explicit capability listのないAPI keyはzero permissionとする。全keyをnamed capabilityへscopeし、OpenAPIと `tools/list` をfilterし、呼び出しごとに認可を繰り返す。全callerの共有DB credentialを変更せず、一callerをrevoke/rotateできる手順をtestする。

8. 実践的アーキテクチャ指針（2/3）: 全契約をEnforceする

外部実行境界で全inputを決定論的に検証する。Unknown fieldを拒否し、type、enum、numeric range、string length、identifier、date、formatを制約する。Predefined SQLとprepared parameterを使い、目的のread-only scopeに合うDB accountで実行する。

Timeout、max rows、max bytes、bounded concurrency、bounded queueを強制する。Caller提供の `limit` だけに依存しない。Explicit output allow-listでresponseをfilterし、missing required columnとunexpected extra columnの両方をtestする。

Gatewayでcaller/IP単位のrate/burst controlを適用し、spoofed forwarded headerを受け入れないtrusted proxy処理を行う。Deployment modelが必要とする場合はauthenticated single active agent connectionを使うが、connection exclusivityをrequest認可やhost integrityと混同しない。

8. 実践的アーキテクチャ指針（3/3）: 境界を運用する

Request ID、timestamp、protocol、capability名、caller/API-key identity、status、public error code、latencyをlogする。Business parameter、result payload、authorization header、SQL、DB credential、secret、raw DB errorはlogしない。Operational metadataをbusiness dataから分け、log accessとretentionを保護する。

全capability変更をsecurity-sensitiveとして扱う。Business purpose、caller group、statement shape、write effect、input constraint、result field、worst-case rows/bytes/time、query plan、concurrency impact、rollbackをreviewする。Writeにはapproval、idempotency、state-transition rule、transaction design、より強いauditを加える。

Production前にprompt injection、wrong-tool、invalid value、boundary maximum、repeated call、retry、rate limit、revocation、gateway compromise assumption、agent disconnect、response/logのsensitive-data absenceをtestする。Denial spike、rate-limit event、unusual capability use、latency shift、policy changeをalertする。

9. 制限事項と今後の課題（1/2）

本評価は例示であり、security audit、formal threat model、penetration test、production capacity benchmarkの代替ではない。Local simulationは記載したarchitecture propertyを比較するが、すべてのMCP client、agent framework、DB engine、API gateway、transport、legacy systemを実装していない。

Prompt injectionを完全に防げるとは証明しない。Deterministic controlが正しく実装され、capability definitionが正しくscope/reviewされ、identity mappingが健全で、別経路からpolicyを迂回できないと仮定する。主としてread-oriented operationを扱う。

Availability attackとhigh-volume abuseはbenchmarkしていない。Human operation、privileged administrator、supply chain compromise、side channel、cross-capability inference、長期間のdata aggregationも十分には評価していない。On-prem OS、network、endpoint protection、DB、backup securityは依然criticalである。

9. 制限事項と今後の課題（2/2）

Write operationにはapproval、idempotency、transaction design、rollback/compensation、concurrency control、change authorization、business-state invariant、より強いaudit trailの追加評価が必要である。Read-orientedな結果をdestructiveまたはfinancial workflowへ一般化すべきではない。

今後の課題は次を含む。

- real MCP clientと代表的agent framework
- real prompt-injection datasetとmulti-step tool-chain evaluation
- red-team exercise、formal threat modeling、capability-review automation
- chaos test、disconnect recovery、retry-loop test、rate-limit benchmark
- repeated bounded callによるaccumulated-data/inference test
- write-operation safety、approval、idempotency、concurrency、rollback
- policy parsing、input schema、result contract、error handlingのfuzzing
- allowed/adversarial request mixでのlegacy DB load測定

これらの証拠でcapability limitと運用runbookを更新すべきである。

10. 参考文献（1/2）

1. OWASP GenAI Security Project. **LLM01:2025 Prompt Injection.** 直接・間接prompt injectionとprompt-only mitigationの限界を説明する。
   https://genai.owasp.org/llmrisk/llm01-prompt-injection/

2. OWASP GenAI Security Project. **LLM06:2025 Excessive Agency.** 過剰な機能、権限、自律性によるconfidentiality、integrity、availability影響を扱う。
   https://genai.owasp.org/llmrisk/llm062025-excessive-agency/

3. Model Context Protocol. **Tools - Specification 2025-11-25.** `tools/list`、`tools/call`、tool schema、input validation、access control、rate limit、output sanitization、timeout、confirmation、audit loggingを定義する。
   https://modelcontextprotocol.io/specification/2025-11-25/server/tools

4. OWASP API Security Project. **API5:2023 Broken Function Level Authorization.** Default deny、明示grant、一貫したfunction-level authorizationを推奨する。
   https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization/

10. 参考文献（2/2）

5. OWASP API Security Project. **API4:2023 Unrestricted Resource Consumption.** Execution time、payload size、records per response、interaction frequency、server-side parameter validationの制限を扱う。
   https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/

6. OWASP Cheat Sheet Series. **Logging Cheat Sheet.** Security event loggingとtoken、credential、connection string、personal/commercial dataの除外・保護を説明する。
   https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

7. NIST. **Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1.** AI riskを設計、配備、利用、評価で管理するvoluntary framework。
   https://doi.org/10.6028/NIST.AI.100-1

8. NIST. **Zero Trust Architecture, SP 800-207.** Network locationの暗黙trustではなく、user、asset、resourceを中心にaccessを扱う。
   https://doi.org/10.6028/NIST.SP.800-207

9. MITRE. **ATLAS Matrix for AI Systems.** Prompt injectionやagent tool invocationを通じたexfiltrationを含むAI system向けadversary tactics/techniquesのknowledge base。
   https://atlas.mitre.org/

最後に

実務上の目的は、AIエージェントを誤り不能にすることではない。誤り、埋め込まれた命令、漏えいcredential、retry loopが広いsystem authorityになる前に、決定論的な境界へ到達するよう周囲のsystemを設計することである。

ケイパビリティベースアクセスは、discoverable、callable、parameterizable、returnable、repeatableな範囲を狭めることでblast radiusを減らしうる。Security valueは、explicit grant、runtime check、local policy、read-only DB permission、bounded execution、output minimization、auditable metadataの組合せから生じる。

境界はcapability、configuration、implementation、host、運用規律と同じ強さしか持たない。「データベースではなく、ケイパビリティを公開する」は保証ではなく、architecture上の制約である。

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年8月3日