ViewLegacy.

`capability.yaml` における入力契約・実行制約・出力契約の防御効果

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09月07日

1. エグゼクティブサマリー（1/2）

AIエージェントは、誤ったパラメータを選び、敵対的な指示に従い、正当な操作を危険な規模で呼び出す可能性がある。モデルへのプロンプトは誤りを減らせるが、決定論的な強制境界にはならない。本書は、定義済みの業務ケイパビリティに設ける3つの外部統制を評価する。

- **入力契約（`params`）**は、受理する名前、型、値、範囲、長さ、パターン、形式を狭める。
- **実行制約（`policy`）**は、読取・書込、実行時間、行数、応答バイト数、公開範囲を制限する。プリペアドステートメントとDB最小権限がこの層を補強する。
- **出力契約（`result`）**は、オンプレミスエージェントから出せるフィールドをallow-list化し、必須フィールドの欠落を検出する。

評価には機密顧客列を含む決定論的なインメモリモデルを使い、同じ18シナリオを7構成で実行した。計測対象は契約の観測可能な結果であり、侵害確率ではない。

1. エグゼクティブサマリー（2/2）

3層すべてを使う構成は、**16件の不利なシナリオのうち15件**を拒否、制約、除外、または検出した。2件の正当な要求は成功した。残る1件は、許可済みの`name`フィールド内に機密テキストが入るケースであり、フィールド名のallow-listでは意味を認識できない。

入力のみは8件、ポリシーのみは独立した行数制限を含む6件、出力のみは5件を処理した。各層が異なる段階を統制するため、組み合わせが強かった。件数は本シミュレーションだけを表す。

> **主な結論。**狭いケイパビリティに3つの契約をすべて適用した構成が、シミュレーションした7構成の中で最良だった。ケイパビリティ単位の認可とDB読取専用アカウントを加えた構成が、本書で検討した最も強いアーキテクチャになる。ただし、機密性、可用性、認可、ホスト侵害、設定不備のリスクは残る。

不適切なSQL、広すぎるケイパビリティ、危険なデフォルト、機密性のある許可フィールドは、意図した境界を崩しうる。読取専用は破壊的リスクを大きく下げるが、開示や高コストな読取は防げない。

2. 背景（1/2）

AIエージェントの判断は確率的である。攻撃がなくても、誤ったステータスを選び、識別子を誤って転記し、過大な結果を要求しうる。プロンプトインジェクションや汚染されたコンテキストは、ユーザーの意図を超える操作を促す。OWASPは、プロンプトインジェクション対策の一部として、ツールアクセスの制限と最小権限を推奨している[4]。

汎用APIはJSON構文を検証しても、業務上の意味を広いまま残すことがある。数値の`limit`が100万でも構文上は正しい。整形式の文字列でも、呼出者が閲覧できない顧客を指しうる。DB権限だけでは通常、ある操作だけを5秒、500行、3フィールドに限定できない。

したがって入力検証は受理する要求空間を狭めるが、本人性、所有権、同意、目的を確立しない。認可は別の判断として残る。

2. 背景（2/2）

ケイパビリティ定義は、機械可読な実行契約かつインターフェース契約として働く。オンプレミスエージェントはレビュー済みの定義を起動時に読み込み、未定義のケイパビリティを拒否し、サービス開始前に構造を検証し、モデル生成SQLではなくプリペアドSQLを使う。変更には意図的なデプロイまたは再起動が必要である。

これにより強制処理はモデルの外へ移る。レビューとテストも、自由形式のクエリ面ではなく、名前付き業務操作を対象にできる。NISTのゼロトラスト指針は、リソースの保護と、可視性・アクセスへの最小権限適用を重視する[5]。

レガシーテーブルでは出力が特に問題になる。公開属性の隣に、運用メモ、リスクスコア、リセットトークン、後日追加された列が並びやすい。SQLが意図より多くの列を返すと、別の境界で応答を絞らない限り露出する。契約モデルは、クエリの正しさと外部可視性を分離する。

3. 3つの防御層

各境界が、受理入力・実行挙動・可視出力という別々の次元を狭める。

層は順番に適用されるが、価値は単なる加算ではない。異なる次元を制約し、独立して失敗しうる。入力上限は物理的なクエリ結果を制限しない。timeoutはどのフィールドが公開可能かを決めない。フィールドallow-listは呼出者を認可しない。

3.1 入力契約（1/2）

OWASPは、明示的な型、範囲、長さ、有限の許容値を使うサーバー側allow-list検証を推奨する[1]。JSON Schemaは、`type`、`enum`、`minimum`、`maximum`、`minLength`、`maxLength`などの構造キーワードを提供する[6]。ケイパビリティでは、DB実行前にこれらの検査を行う。

契約要素	防御効果	例
`type`、`required`、`default`	形状と欠落時の挙動を安定化	`limit`は整数、既定値50
`enum`	有限の業務状態に限定	`active`または`inactive`
`minimum`、`maximum`	無意味または過大な値を拒否	`1 <= limit <= 500`
`minLength`、`maxLength`	解析と下流入力サイズを制限	顧客コードは最大32文字
`pattern`、`format`	定義した字句形式を検査	テナントコードや日付構文
未知フィールド拒否	未宣言の挙動切替を防止	`include_internal=true`を拒否

3.1 入力契約（2/2）

入力契約は、不正な識別子・日付、過大な文字列、負数・ゼロ・過大な上限、不正な列挙値を減らす。インジェクション様の値もenumやpatternで拒否できる場合がある。この検査はプリペアドステートメントを補完する。OWASPは、検証済みデータでもSQL文字列連結に安全とは限らないと警告する[2]。

限界も明確である。

- 正しい形式のアカウントIDでも、他人のアカウントを指しうる。
- 個別に正しいフィールドの組み合わせが、業務上は不正な場合がある。
- 正規表現と`format`は意味の正しさを証明しない。
- 広すぎるenumや危険なdefaultは、形式的に有効な危険要求を作る。
- 識別子によっては、基本的なpatternでは表現できない正規化規則が必要になる。

入力検証は、認証、ケイパビリティ認可、行所有権、同意、ワークフロー承認の代替ではない。「要求が許容形状か」には答えるが、「この呼出者が対象レコードに実行できるか」には答えない。

3.2 実行制約（1/2）

実行制約は、入力受理後の挙動を制限する。`readonly: true`はケイパビリティ層で更新文を拒否する。アプリケーション側の分類に失敗しても独立して拒否できるよう、DB読取専用アカウントを併用すべきである。PostgreSQLはトランザクションの読取専用モードを説明する一方、「read only」はエンジン固有の高水準概念であると注意している[8]。

`timeout`、`max_rows`、`max_bytes`は異なる資源次元を制限する。timeoutは時間、行上限は件数、バイト上限は直列化後の量を制限する。OWASPのAPI指針は、無制限な資源消費が可用性とコストへ影響するため、入出力に関連する上限を推奨する[3]。

`expose_in_openapi`は内部ケイパビリティを生成された発見面から隠す。発見可能性は下がるが、実行時認可は依然として呼出しを拒否しなければならない。発見フィルタはアクセス制御の代替ではない。

3.2 実行制約（2/2）

プリペアドステートメントはパラメータ値とクエリ構造を分離し、SQLインジェクションの主要対策になる[2]。起動時のlint、プレースホルダ検査、`EXPLAIN`などのDB検証は、サービス開始前に不正な定義を検出できる。ただし、クエリが正しい業務規則を実装したことまでは証明しない。

実行制御は暴走クエリ、過大取得、偶発的な書込、一部のインジェクション試行を軽減する。次のリスクは残る。

- 正当だが広すぎる読取による機密性リスク
- 上限未満の処理や反復呼出しによる可用性リスク
- join、filter、テナント条件の意味的な誤り
- エンジン固有の読取専用例外とドライバのcancel挙動
- オンプレミスエージェントホストの侵害

`limit <= 500`と`max_rows <= 500`は意図的に分離する。前者は過大な要求を拒否し、後者はSQLがパラメータを無視・誤適用・削除した場合でも実際の返却行数を制限する。

3.3 出力契約（1/2）

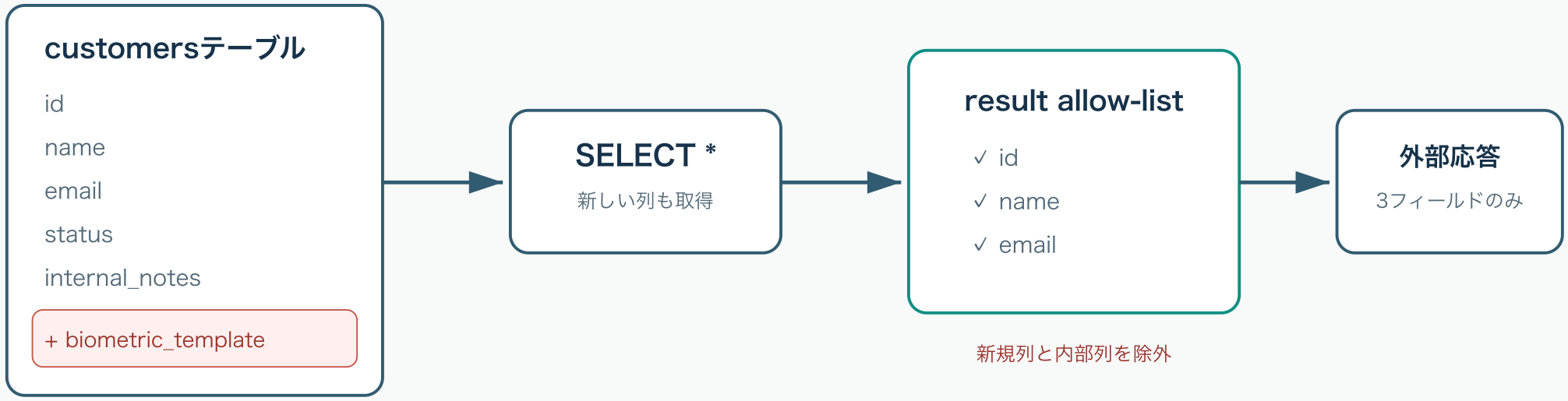
`result`は明示的なフィールドallow-listを定義する。エージェントは、SQLが返してもリストにないフィールドを削除し、必須フィールドが欠落すれば失敗し、契約がなければ行オブジェクトのフィールドを返さない。したがって、`SELECT *`だけでは新しい列を公開できない。

この層は次を提供する。

- SQL projectionから独立したフィールド最小化
- 偶発的な`SELECT *`と後日のクエリ拡張への防御
- 期待フィールドが消えた際のfail closedな不一致検出
- 内部スキーマ変更に対する公開インターフェースの安定性
- オンプレミス境界を越えられるフィールドのレビュー可能な宣言

明示的なSQL列指定は、内部で読み取り処理するデータも減らすため、引き続き望ましい。出力契約は独立した出口境界であり、広いSQLを書く許可ではない。

3.3 出力契約（2/2）



クエリやスキーマが広がっても、フィールド最小化は維持される。

出力allow-listはフィールド名を統制し、意味は解釈しない。許可された`name`フィールドにコピーされたリセットトークン、自由記述に埋め込まれた秘密、許可クエリの反復から推論できる情報を検出できない。どの行を誰が閲覧できるかも決めない。

機密性が本人性、所有権、内容に依存する場合は、行単位認可、DBのrow security、意味分類、マスキング、DLPが必要になる。出力フィルタはクエリ実行後に働くため、広いSQLによるDB処理量そのものも減らさない。

4. 統制構成の比較（1/2）

構成	入力の自由度	実行の自由度	出力の自由度	想定影響範囲	主な弱点
1. 契約なし	任意	任意のクエリ/API挙動	返却された全列	本統制では無制限	モデル・呼出者がクエリと結果を形成
2. 入力のみ	宣言値のみ	受理後は無制限	返却された全列	不正入力リスク低下	正当要求でも長時間実行・列開示が可能
3. policyのみ	広い	読取専用、時間・行・byte制限	上限内の全列	実行被害を制限	意味的な不正入力・機密列が残る
4. resultのみ	広い	無制限	allow-list列	列開示を低減	書込、高コスト読取、誤った行が残る
5. 入力 + policy	狭い	制限あり	返却された全列	要求と実行範囲を縮小	出口列がSQL依存のまま
6. 入力 + result	狭い	無制限	allow-list列	要求と列範囲を縮小	長時間・破壊的実行が残る
7. 3層すべて	狭い	制限あり	allow-list列	シミュレーション最小	認可、行、意味、ホストリスクが残る
8. 3層 + scope認可 + DB読取専用	狭く呼出者別	DB防波堤付き制限	allow-list列	検討中で最も強い	設定と実装が安全性を左右

4. 統制構成の比較（2/2）

運用上の複雑さは、単一の開放的なクエリ面から、レビュー済み定義、テスト、デプロイ統制、ケイパビリティ単位認可へ増える。再現性と限定権限がアドホックな柔軟性より重要な、決定論的な業務操作では、この負担に合理性がある。

構成8が本書の推奨本番パターンだが、ローカル実験が計測するのは構成1から7である。認可とDB権限は、実際のID、ポリシー、DBに対する統合テストが必要になる。

分析担当者が柔軟なアドホックSQLを必要とする場合、幅広い分析joinが必要な場合、クエリ構造を動的生成する必要がある場合、このモデルは適合しにくい。統制されたデータウェアハウス、セマンティックレイヤー、隔離された分析環境の方が適することがある。複数トランザクションにまたがるワークフローや書込には、3つの契約に加え、承認、冪等性、トランザクション設計、競合制御、復旧が必要である。

5. 評価方法（1/2）

再現可能な`src/evaluate-scenarios.mjs`は、架空の顧客620件を使う。基礎モデルには`internal_risk_score`、`internal_notes`、`password_reset_token`があり、スキーマ変更シナリオでは`biometric_template`を追加する。意図したケイパビリティは`search_customers(status, limit)`、公開フィールドは`id`、`name`、`email`である。

評価器は各構成で、有効な統制を本番と同じ順に適用する。

1. 宣言済みパラメータを検証し、未知フィールドを拒否する。
2. 読取専用、模擬timeout、プリペアド値処理、行・バイト制限を適用する。
3. 出力allow-listを適用し、必須フィールド欠落を検出する。

入力と結果はすべて決定論的である。スクリプトは、JSONとMarkdownの証跡を書く前に3層構成の結果をassertする。DBもネットワークも不要である。

5. 評価方法（2/2）

18シナリオは、3件の正常・残余ケースと、契約反応を期待する15件を含む。未知入力、不正enum、ゼロ・負数・過大limit、過大文字列、インジェクション様入力、timeout、過大な行数とバイト数、予期しない列、`SELECT *`、スキーマ拡張、必須出力欠落、書込試行を扱う。

S18は、許可された`name`に意図的に機密テキストを入れ、既知の意味的限界を試す。S1とS17は正当な要求で、成功すべきである。

結果ラベルは次の通り。

- **拒否:** 実行または応答を停止
- **制約:** 上限内または安全な値処理経路でのみ継続
- **除外:** 未許可フィールドを削除
- **検出:** 契約不一致により失敗
- **許可:** 有効な統制がシナリオを変更しない

これは説明用の契約テストであり、形式的セキュリティ証明、侵入テスト、性能ベンチマーク、事故確率の測定ではない。

6. 評価結果: シナリオ結果（1/3）

シナリオ	契約なし	入力のみ	policyのみ	resultのみ	入力+policy	入力+result	3層すべて
S1 正常要求	許可	許可	許可	許可	許可	許可	許可
S2 未知パラメータ	許可	拒否	許可	許可	拒否	拒否	拒否
S3 不正enum	許可	拒否	許可	許可	拒否	拒否	拒否
S4 limitが0	許可	拒否	許可	許可	拒否	拒否	拒否
S5 limitが負数	許可	拒否	許可	許可	拒否	拒否	拒否
S6 極端なlimit	許可	拒否	制約	許可	拒否	拒否	拒否

S6は独立した上限を示す。入力検証は100万行の要求を拒否する。入力検証がない場合も、`max_rows`がクエリ結果を制約する。

6. 評価結果: シナリオ結果（2/3）

シナリオ	契約なし	入力のみ	policyのみ	resultのみ	入力+policy	入力+result	3層すべて
S7 過大文字列	許可	拒否	許可	許可	拒否	拒否	拒否
S8 injection様の値	許可	拒否	制約	許可	拒否	拒否	拒否
S9 timeout超過	許可	許可	拒否	許可	拒否	許可	拒否
S10 max_rows超過	許可	拒否	制約	許可	拒否	拒否	拒否
S11 max_bytes超過	許可	許可	拒否	除外	拒否	除外	拒否
S12 予期しない機密列	許可	許可	許可	除外	許可	除外	除外

S8ではenum検証が実行前に値を拒否する。policyのみでは、プリペアド値処理によりデータとして扱う。どちらも、許可値が認可済みであることまでは証明しない。

6. 評価結果: シナリオ結果（3/3）

シナリオ	契約なし	入力のみ	policyのみ	resultのみ	入力+policy	入力+result	3層すべて
S13 SQLが `SELECT *` に変更	許可	許可	許可	除外	許可	除外	除外
S14 機密列を追加	許可	許可	許可	除外	許可	除外	除外
S15 必須列が欠落	許可	許可	許可	検出	許可	検出	検出
S16 読取専用下の書込	許可	許可	拒否	許可	拒否	許可	拒否
S17 正常な意図結果	許可	許可	許可	許可	許可	許可	許可
S18 許可列に機密文	許可	許可	許可	許可	許可	許可	許可

S13とS14は、明示的列選択と出力契約が補完関係にあることを示す。S18は3層すべての範囲外で、内容認識または業務固有の保護が必要になる。

6. 評価結果: 件数集計

構成	拒否	制約	除外	検出	軽減合計	許可
契約なし	0	0	0	0	**0**	18
入力のみ	8	0	0	0	**8**	10
policyのみ	3	3	0	0	**6**	12
resultのみ	0	0	4	1	**5**	13
入力 + policy	11	0	0	0	**11**	7
入力 + result	8	0	4	1	**13**	5
3層すべて	11	0	3	1	**15**	3

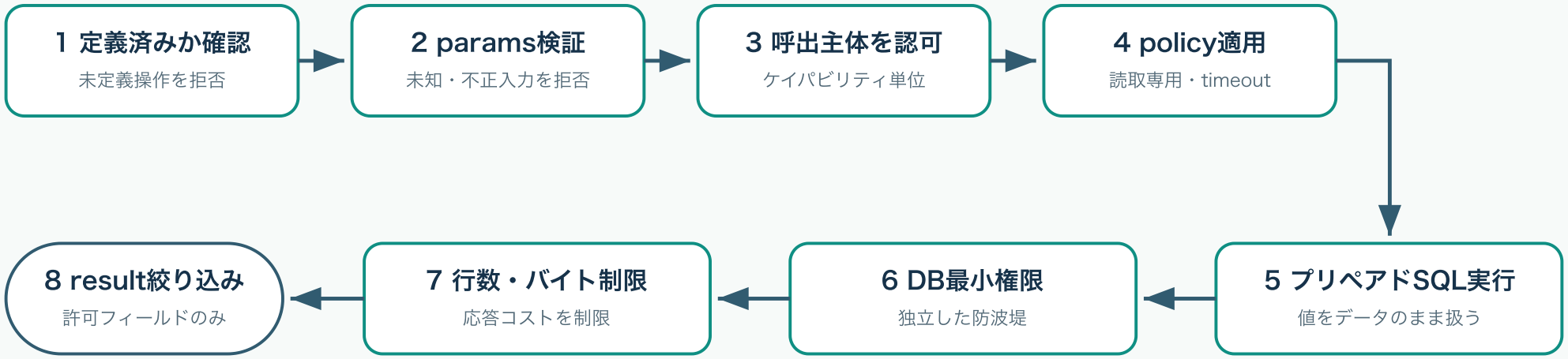
「軽減合計」は拒否、制約、除外、検出を数える。セキュリティスコアではない。3層構成はS1とS17を意図通り許可し、未解決の意味内容ケースS18も許可した。

6. 評価結果: 統制層の有効性

層	主対象シナリオ	観測した効果	主な残余リスク
入力契約	S2–S8、S10	未知、不正、無意味、過大な値を拒否	形式上正しい未認可値、項目間の業務規則
実行制約	S6、S8–S11、S16	値処理、時間、行、byte、書込を制限	広い正当読取、反復呼出し、driver/engine挙動
出力契約	S11–S15	予期しない列を除外し、必須出力欠落を検出	誤った行、許可列内の機密的意味
3層すべて	正常経路を除くS2–S16	構造、実行、フィールド形状の全不利ケースを処理	S18の意味漏えいとモデル外統制

層ごとの件数は重複する。この重複は意図的であり、同じシナリオを異なる段階で制約し、一方が失敗したときの防波堤を残す。

6. 評価結果: 多層防御



独立した検査は、他の統制が欠落・誤設定・迂回された場合の被害を抑える。

3つの契約は、ケイパビリティ存在確認、呼出者認可、プリペアドSQL、DB権限で囲むと実質的に強くなる。MCPは`inputSchema`と任意の`outputSchema`を公開できるが、クライアントから見えるツール宣言はサーバー側強制の代替にならない[10]。ツールannotationはhintであり、ポリシーとして信頼できない。

6. 評価結果: 独立した防波堤

シナリオ	主統制	副統制	追加で必要な統制
過大な要求limit	`params.maximum`	`policy.max_rows` と `max_bytes`	rate/concurrency制限、監視
injection様パラメータ	プリペアドステートメント	enum/pattern検証	動的識別子回避、DB最小権限
偶発的 `SELECT *`	明示的なSQL projection	`result` allow-list	SQL・schema差分レビュー
書込試行	`policy.readonly`	DB読取専用アカウント	ケイパビリティ認可、host保護
未定義ケイパビリティ	ケイパビリティallow-list	scope済み発見・認可	Agent側強制
必須フィールド削除	`result` 不一致失敗	起動時SQL検証	対象schemaでのdeploy test

異なる状態を観測する重複上限には意味がある。入力上限は呼出者の意図を、実行上限は実際の処理と出力を制限する。

6. 評価結果: 残余リスク

リスク	`params`	`policy`	`result`	追加統制
正当形式のレコードに権限がない	対象外	対象外	対象外	ID、capability認可、行所有権/RLS
許可列内の機密テキスト	対象外	対象外	対象外	意味分類、masking、DLP
広い読取専用SQL	対象外	一部	列のみ	SQL review、view、RLS、DB grant
上限未満の反復要求	対象外	一部	対象外	rate/concurrency quota、監視
Agent host侵害	対象外	対象外	対象外	host hardening、secret分離、IR
危険な契約変更	対象外	対象外	対象外	version control、review、lint、test、署名release
engine固有のread-only gap	対象外	一部	対象外	制限DB account、engine別test
誤ったglobal default	対象外	一部	一部	保守的default、設定test

契約が制限するのは1回のケイパビリティ呼出しである。完全なID、データガバナンス、ホストセキュリティ、可用性の仕組みではない。

7. 考察（1/2）

実験が支持するのは多層防御の結論であり、安全性の保証ではない。入力契約は受理要求空間を狭める。policyは実行時挙動を制限する。result契約はAgentから出る名前付きフィールドを制限する。各層は他の層が許可するケースを処理する。

静的定義は、再現性、起動時検証、変更レビュー、監査解釈を改善する一方、柔軟性を下げる。この設計は、注文状態や限定的な顧客検索のような決定論的な業務読取に適する。探索的分析、広いjoin、動的生成クエリには適しにくい。

プリペアドステートメントは値のインジェクションを扱うが、正当な広いクエリは危険なままである。読取専用は破壊的結果を減らすが、機密性と可用性リスクは残る。出力allow-listは偶発的なスキーマ拡張に耐えるが、行認可や自由記述の機密解釈を提供しない。

7. 考察（2/2）

主な失敗モードは、制御不能なモデル出力から設定品質へ移る。広いパラメータ、長いtimeout、非常に大きな上限、機密result列を持つ`export_all_customers`は、スキーマに適合してもセキュリティ目的に反する。過度に許容的なglobal defaultは、同じ誤りを全ケイパビリティへ広げる。

したがってケイパビリティ設定は、セキュリティ上重要なコードと同様に扱うべきである。owner、code review、version control、構造lint、query検証、scenario test、制御されたdeployが必要になる。人手レビューだけでは、default、新しい列、SQLとresultの相互作用を見逃しうる。

オンプレミス側の強制実装も信頼基盤に含まれる。バグやhost侵害は通常の検査を迂回しうる。DB権限、network制御、secret管理、監視、incident responseは引き続き必要である。

8. 実務的なアーキテクチャ指針（1/2）

公開はdeny by defaultにし、汎用テーブルアクセスではなく狭い業務操作を定義する。外部から受理する全パラメータを明示し、未知の名前を拒否し、有限のドメインではenumを優先する。意味のある数値上下限と、文字列の長さ・pattern・format制約を適用する。

ユーザー値はプリペアドステートメントのパラメータに保つ。可能なら動的SQL識別子を避ける。読取中心のケイパビリティは`readonly: true`と、必要な`SELECT`だけを持つDBアカウントで始める。明示的timeout、行数・バイト数の両上限を適用する。反復呼出しが資源を枯渇させうる場合は、ケイパビリティ外でもconcurrency・rate制御を加える。

OpenAPIまたはMCPの発見面には、公開かつ認可されたケイパビリティだけを生成する。OpenAPI Schema Objectは入力・出力型を記述できる[7]が、サーバー側強制が正本である。

8. 実務的なアーキテクチャ指針（2/2）

明示的な`result` allow-listを定義し、`SELECT *`を避ける。必須resultフィールドが一つでも欠けたらfail closedにする。新しいDB列は、レビュー済み契約変更で許可するまで非公開とする。デプロイ前にSQL projection拡張とschema変更をテストする。

定義はversion control下に置く。構造lint、未知設定field拒否、placeholder検査、可能な範囲でDB接続を使う起動時検証を行う。global defaultは1つの危険値が多くのケイパビリティへ影響するため、注意してレビューする。

ログには、capability名、caller/key識別子、status、error category、latency、必要に応じてresult countを記録する。SQL、DB credential、Authorization header、生のparam値、業務payloadを集中ログへ出さない。error classとlimit到達を監視し、不正call、policy強制、schema不一致、infrastructure failureを区別できるようにする。

8. 例A: 強く制約したケイパビリティ

```yaml
capabilities:
  search_customers:
    description: "対応するステータスで顧客を検索"
    sql: |
      SELECT id, name, email
      FROM customers
      WHERE status = :status
      ORDER BY id
      LIMIT :limit
    params:
      status:
        type: string
        required: true
        enum: ["active", "inactive"]
        maxLength: 8
      limit:
        type: integer
        default: 50
        minimum: 1
        maximum: 500
    policy:
      readonly: true
      timeout: 5s
      max_rows: 500
      max_bytes: 256KB
      expose_in_openapi: true
    result:
      id: { type: integer }
      name: { type: string }
      email: { type: string }
```

SQL projection、実行上限、出口allow-listは独立した統制である。

8. 例B: 意図的に弱いアンチパターン

```yaml
capabilities:
  query_customers:
    description: "柔軟な顧客クエリ"
    sql: |
      SELECT *
      FROM customers
      WHERE status LIKE :search
    params:
      search:
        type: string
    policy:
      readonly: false
      expose_in_openapi: true
    # timeout、max_rows、max_bytes、result allow-listがない
```

このアンチパターンは、制約のない文字列を受理し、広いSQLを使い、根拠なく書込可能なpolicyを有効化し、実行・出力上限を省略する。本番では推奨しない。

この広い操作をスキーマで囲っても、業務ケイパビリティは狭くならない。契約キーの存在より契約品質が重要である。

9. 制限事項と今後の課題（1/2）

本シミュレーションは説明用であり、セキュリティ監査、侵入テスト、形式的脅威モデル、本番性能試験の代替ではない。強制実装が正しいことを前提とする。すべてのDBエンジン、SQL方言、driver、ORM、API gateway、MCP client、agent framework、transaction semantics、cancel経路、deployment topologyを扱っていない。

静的な入力契約では全業務規則を表現できない。regexとformatは意味的正しさを確立しない。出力契約は行単位認可を提供せず、許可された自由記述内の機密的意味を自動検出できない。読取専用でも機密性・可用性リスクは残る。時間・行・バイト上限も、特に並行・反復要求による資源枯渇を完全には防げない。

DBの読取専用挙動はエンジン・設定で異なる。ワークロード固有の性能統制も必要である。

9. 制限事項と今後の課題（2/2）

本書は読取中心のケイパビリティに焦点を置く。書込には、呼出者認可、承認ポリシー、冪等性、トランザクション境界、競合制御、rollback挙動、結果確定、変更監査が必要である。複数トランザクションにまたがる複雑なワークフローには、`params`、`policy`、`result`を超えるオーケストレーションと復旧契約が必要になる。

今後は、複数の実DBエンジン・driver、parameterのproperty-based test・fuzz test、schema evolution matrix、capability差分review、SQL static analysis、policy検証、host侵害前提を評価すべきである。書込契約、row-level security、意味的な出力分類、rate・concurrency制限、cancel保証、end-to-end認可も対象になる。

ローカルモデルと生成結果を`src.zip`に残し、レビュー担当者が件数を再現して前提を明示的に変更できるようにする。

10. 参考文献（1/2）

1. OWASP Cheat Sheet Series, “Input Validation Cheat Sheet.”
   https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
2. OWASP Cheat Sheet Series, “SQL Injection Prevention Cheat Sheet.”
   https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
3. OWASP API Security Top 10, “API4:2023 Unrestricted Resource Consumption.”
   https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/
4. OWASP Cheat Sheet Series, “LLM Prompt Injection Prevention Cheat Sheet.”
   https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html
5. NIST SP 800-207, “Zero Trust Architecture,” August 2020.
   https://csrc.nist.gov/pubs/sp/800/207/final
6. JSON Schema, “Validation: A Vocabulary for Structural Validation of JSON,” Draft 2020-12.
   https://json-schema.org/draft/2020-12/json-schema-validation

10. 参考文献（2/2）

7. OpenAPI Initiative, “OpenAPI Specification v3.1.2,” September 19, 2025. https://spec.openapis.org/oas/v3.1.2.html
8. PostgreSQL Global Development Group, “SET TRANSACTION.” https://www.postgresql.org/docs/current/sql-set-transaction.html
9. PostgreSQL Global Development Group, “Client Connection Defaults,” `statement_timeout` を含む。 https://www.postgresql.org/docs/current/runtime-config-client.html
10. Model Context Protocol, “Tools,” specification version 2025-11-25. https://modelcontextprotocol.io/specification/2025-11-25/server/tools
11. Model Context Protocol, “Authorization,” specification version 2025-11-25. https://modelcontextprotocol.io/specification/2025-11-25/basic/authorization

参照先は2026年09月07日に確認した。製品固有の挙動と前提はリポジトリの `architecture.md` に従う。本シミュレーションは本書の参照モデルを試験するものであり、本番実装を認証するものではない。

最後に

入力・実行・出力の完全な契約は、シミュレーションした7構成の中で最良の結果を出した。推奨する本番構成では、ケイパビリティ単位認可と最小限の読取権限を持つDBアカウントを加える。

これはOnprestの中心的なアーキテクチャ方針、すなわちデータベースや生クエリ面ではなく、狭く決定論的なケイパビリティを公開する考え方を支持する。ただし結論は、狭いケイパビリティ設計、保守的なデフォルト、正しいAgent側強制、レビュー済み変更、ID・行・意味・可用性・ホストセキュリティの統制を条件とする。

発行: 合同会社ビューレガシー
著者: 増田朝陽
発行日: 2026年09月07日