

この資料は、 3 つの読み方ができます

読み方	最初に読むページ	得られるもの
背景から読む	発表原版 01 ～ 10 + 追補	「なぜ ARC を作ったか」を追体験
手元で試す	原版 11 ～ 19 + Quick Start 追補	検査と許可外の拒否を再現
導入を検討する	G1 ～ G3 ・ 比較 ・ 制約 ・ 評価	既存の統制へ組み込むか判断

20 分 LT / 実体験から OSS へ

AI チームは動いている。 しかし僕は現在地を見失った。

AI エージェントに、どうやって役割を守らせる？

OSS 「Agent Role Contracts」を作った話

Netsujo

実際のログを基にしたものではなく、経験を伝えるための再構成

Agent A 「実装しました」

Agent B 「テスト成功です」

Agent C 「レビュー完了です」

Agent D 「公開の準備ができました」

報告は増えた。でも、何を根拠に判断する？

「完了しました」の意味は、ひとつじゃなかった

01 コードが変わった

実装

02 テストが通った

検証

03 別の担当が確かめた

レビュー

04 利用者が使えた

実利用

それぞれ、確認すべき証拠が違う



誰が書いた？

誰がレビューした？

何を承認した？

その承認は今の変更にも有効？

作業は進むのに、僕の頭の中の地図だけが追いつかない。

会場のみなさんにも、聞いてみたい

心当たりがある方、少しでも手を挙げてください

AI に「完了しました」と言われて、
「結局、何が終わったんだっけ？」
と思ったことはありませんか？

※この挙手は、統計を取るためのものではありません

「完了」と言う前に、証拠の種類を分ける

段階	言えること	まだ言えないこと
実装	指定のファイルを変更した	テスト済み・受入済み
テスト	定義した試験が成功した	レビューの独立性
レビュー	指定対象のレビュー結果がある	本人の認証・実行許可
公開・利用	特定環境での到達・実利用を確認	別環境での成功

何を記録すれば、現在地を取り戻せるか

POINT 01

誰が・何を

role / task / action の宣言
対象 revision ・候補の識別子
書いてよいファイル範囲

POINT 02

誰が確かめたか

reviewer と approver
何の候補に対する判断か
有効な時点・実行側の証拠

同じ課題が、世界でも議論されていた

NIST / 米国国立標準技術研究所

AI Agent Standards Initiative

2026 年 2 月: 相互運用性・安全性
9 月: ソフトウェア開発での認証・認可

OWASP / Agentic Security

Tool misuse Identity / privilege Inter-agent trust

2026 年: 自律エージェント特有のリスクを整理

共通する問いは「AI の能力を、誰の権限で、どう使うか」

NIST ・ OWASP との関係を正確に捉える

POINT 01

世界で議論される課題

NIST（2026）：エージェントの相互運用、識別、認可の課題

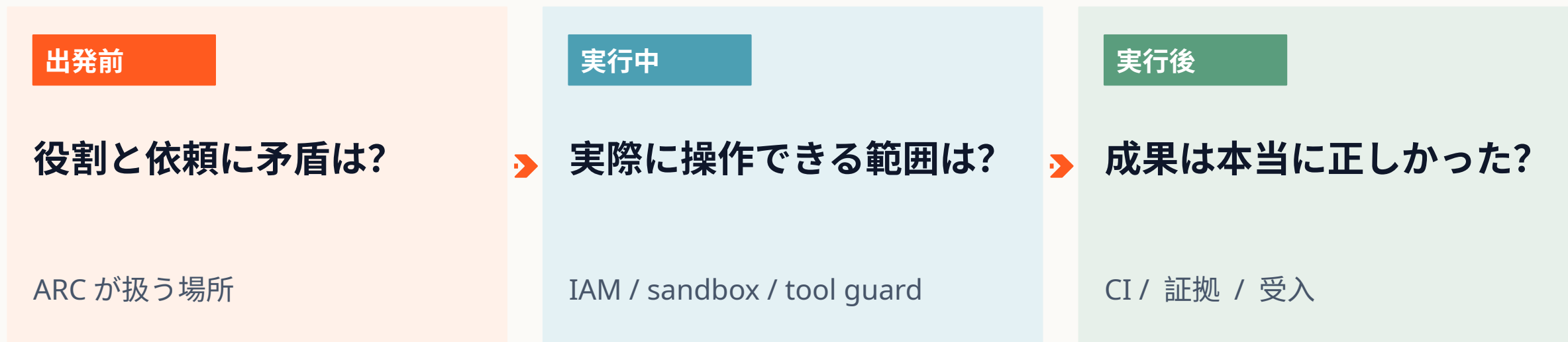
OWASP：ツール誤用、権限濫用、エージェント間の不正な連携

POINT 02

ARC が扱う狭い領域

仕事を渡す前の宣言矛盾を検査。
NIST の規格そのものではなく、OWASP リスク全体を防ぐ製品でもない。

AI エージェントの信頼を、3つの段階で考える



ARC が検査するのは、実行前の「宣言の矛盾」

信頼は「渡す前 → 動かす時 → 動かした後」で分担する

01

宣言の整合

ARC：役割、作業範囲、
レビュー対象を事前照合

02

本人と権限

実行環境：ID・承認・
許可された操作を確認

03

実際の操作

tool gateway / OS / API
が権限を強制

04

証拠と受入

CI・実物の bytes・独立
QC・利用結果を確認

社内で繰り返していた、4つの確認

ROLE

誰が担当？

SCOPE

どこまで変更できる？

REVIEW

誰が確認？

HANDOFF

次は誰へ？

Agent OS / Controller の運用では、毎回この確認が必要だった

「今回は誰がレビューするんだっけ」

「このエージェント、そこを触っていい？」

この確認を、機械で検査できないだろうか。

なぜ「宣言検査」を OSS に切り出したのか

社内固有の権限基盤を持ち込まない

エージェントの定義、ホスト状態、認証情報、実行権限は公開パッケージに含めない。

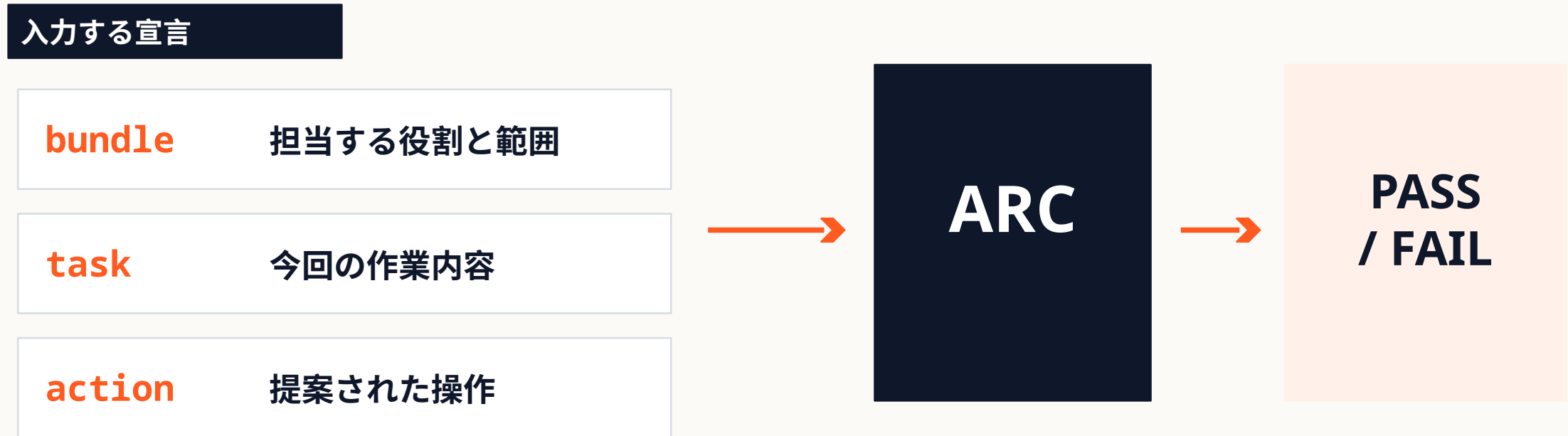
狭い仕事を高い再現性で扱う

JSON を入力し、同じ宣言に対して安定した診断結果を返す。AI プロバイダーへアクセスしない。

外部からの反証を受け付ける

既存の IAM ・ CI ・ sandbox が十分だった場合も、次の改善判断に使う。

ARC は、仕事を任せる前に宣言を確かめる



AI を起動せず、渡された宣言に矛盾がないか調べる

5 分で分かる ARC の用語

用語	何を表すか	身近な例
bundle	権限・役割・ route のまとめ	チーム全体のルール表
role	誰が何を担当するか	implementer / reviewer
task	今回の依頼と入力	src/example.mjs を変更
action	提案される操作の宣言	ファイルへの書き込み
binding	レビューと現在の対象の結び付き	変更前の承認を流用しない
lifecycle	進行・成果の宣言上の整合	status や artifact の識別

検査結果の後、実行フローはどう進む？

01**正規ルールを読む**

作業依頼の本文から権限を自動生成しない

02**ARC で宣言を照合**

検査操作に対応するAPI/CLI を選ぶ

03**FAIL なら修正**

依頼の誤りや承認の不足を確認

04**PASS 後の権限確認**

認証・実行権限・最新性を別に検査

実装 AI ・ レビュー AI ・ 人間の承認者

01

実装担当の AI

implementer

02

レビュー担当の AI

reviewer

03

人間の承認者

human

実装とレビューは別の役割。実際の人間承認は外部で確認する。

スターターの役割と route の関係

POINT 01

implementer

write_scoped
allowed_write_scopes: ["src/**"]
レビュー権限は持たない。

POINT 02

reviewer + route

reviewer は read_only。
route は executor=implementer、
reviewer=reviewer、human approval 必須と宣言。

実装役のルールは `src/\*\*`

```
"id": "implementer",  
"authority_mode": "write_scoped",  
"allowed_write_scopes": ["src/**"],  
"prohibited_capabilities": [  
  "review.perform"  
]
```

src/ の変更を
許可する宣言**

**自分の実装は
レビューしない**

実際のJSON から、重要な項目だけを抜粋

レビュー役には、書き込みを許可しない

```
"id": "reviewer",  
"authority_mode": "read_only",  
"capabilities": [  
  "filesystem.read", "review.perform"  
],  
"allowed_write_scopes": []
```

読む・レビューする

書き込みは権限外

実装とレビューに、同じ役割を指定すると...

SELF_REVIEW_DECLARED

今回の依頼は、1 ファイルの変更

```
{  
  "schema_version": "0.1",  
  "type": "code-change",  
  "inputs": {  
    "scope": "src/example.mjs"  
  }  
}
```

今回の
変更対象

src/** の内側

公式の starter-task.json から主要な項目を抜粋

権限宣言は入れ子の JSON にある

starter-bundle.json の抜粋

```
roles[].contract.authority
  authority_mode: "write_scoped"
  allowed_write_scopes: ["src/**"]
  prohibited_capabilities: ["review.perform"]
routes[].task_type: "code-change"
```

「許可」とは、ARC へ渡す宣言上のルール。OS の実権限を書き換える命令ではありません。

実際のタスクは、 **scope** で変更対象を指定する

starter-task.json (必要な項目の抜粋)

```
{
  "schema_version": "0.1",
  "type": "code-change",
  "inputs": {"scope": "src/example.mjs"}
}
```

schema とタスク種別が route に対応し、scope が implementer の許可範囲に収まるかを照合します。

【実演】最初は PASS

実装役の許可

`src/**`



今回の依頼

`src/example.mjs`

PASS

宣言は整合している。ただし、実行の許可は別に必要。

依頼するファイル名だけ、変えてみる

さっき

```
"scope": "src/example.mjs"
```



今度

```
"scope": "secrets/production.txt"
```

実装役に宣言した許可範囲は、変えていない

ARC は「どこが矛盾したか」を返す

TASK_WRITE_SCOPE_OUTSIDE_AUTHORITY

何が問題? 許可範囲外への書き込み要求

どの項目? `task.inputs.scope`

次は? 依頼を直すか、正規の権限確認に戻る

元に戻すと PASS。ファイルには何も書き込まない

STEP 01

`src/example.mjs`

PASS

STEP 02

`secrets/production.txt`

FAIL

STEP 03

`src/example.mjs`

PASS

公式デモの流れ: PASS → FAIL → PASS

同じ宣言で、 path だけを変えた結果

task.inputs.scope	宣言 `src/**` 内か	ARC の期待判定
src/example.mjs	はい	PASS : 宣言の整合
README.md	いいえ	FAIL : 許可範囲外
secrets/production.txt	いいえ	FAIL : 許可範囲外
src/another.mjs	はい	PASS : 宣言の整合

FAIL 時の診断コード: TASK_WRITE_SCOPE_OUTSIDE_AUTHORITY

自分の task.json を編集して CLI で検査する

リポジトリ直下で実行する例

```
cp examples/starter-task.json task.demo.json
node bin/agent-role-contracts.mjs explain \
  --bundle examples/starter-bundle.json \
  --task task.demo.json --format text
```

cp は macOS/Linux 向け。Windows ではファイルを手動で複製し、inputs.scope を変更して再検査。

自分の PC でも、3 分で試せる

必要なのは、Git と Node.js 22.5 以上

```
git clone https://github.com/suirindo/agent-role-contracts.git  
  
npm --prefix agent-role-contracts run demo --silent
```

Demo complete: PASS -> FAIL (expected) -> PASS.

npm install も、API キーも、エージェント起動も不要

終了コード：自動デモと CLI を混同しない

実行	exit code	意味
CLI の正常な整合検査	0	検査した宣言が整合
CLI の不整合	1	宣言上の矛盾を検出
CLI の引数・読込エラー	2	入力や実行の問題
公式の自動デモ	0	想定した FAIL も正しく検知した

「3分のデモ」の次に、何を試す？

1. 自分のワークフローをひとつ選ぶ

例：AI が src/ の実装を担当し、別ロールが read-only でレビューする開発タスク。

2. わざと矛盾を作る

許可外パス・自己レビュー・レビュー後の action 変更から 1 つ選び、診断コードを確かめる。

3. 既存の制御と比較する

同じ問題を CI ・ IAM ・ sandbox ・ 既存のガードがどう扱うかを記録する。

説明用の架空例

レビュー対象だった action

src/example.mjs

レビュー後、action の path が変更された

src/other.mjs

どちらも src/** の内側。範囲の検査結果は変わらない。

なぜ path が範囲内でもレビューが古くなる？

POINT 01

レビューした対象 A

action: src/example.mjs

review binding: digest(A)

担当がこの内容にレビューした、と宣言。

POINT 02

変更後の対象 B

action: src/other.mjs

現在の subject: digest(B)

範囲内でも、レビュー対象が変わった。

G1 は「レビュー対象の指紋」をつくる



レビューと承認の宣言を、この指紋に結び付ける。

※ 指紋は署名でも、本人確認でも、実行許可でもありません

レビュー時と現在で、対象の中身が違う

レビュー時

digest (A)

レビュー宣言の対象

実行直前

digest (B)

変更後の対象

G1_SUBJECT_MISMATCH

古いレビュー宣言を、変更後の操作に流用しない

G1: 宣言の「指紋」を作り、古い結び付きを検出

01**対象を組み立てる**

bundle 全体 + task 全体
+ action 全体

02**SHA-256 で識別**

定義済みの正規化方法で
digest を生成

03**対象に結び付ける**

同じ digest と role/route
の関係を照合

04**変更後は不一致**

旧 binding は
G1_SUBJECT_MISMATCH

G1 を自分で確かめるコマンド

Node.js 22.5+ / リポジトリのルートで実行

```
npm run demo:binding
```

```
# 想定： 元の binding は PASS  
# task/action を変えて元の binding を使うと  
# G1_SUBJECT_MISMATCH を検出
```

digest は宣言の完全性識別子。署名・本人認証・本物の承認を立証するものではありません。

「一致している」と「本物である」は違う

検査対象	ARC で分かること	ARC では分からないこと
subject digest	宣言の内容が一致	署名者の身元・失効
reviewer role	別の役割に割り当て	実際に別人・別プロセスか
approval 宣言	現在の subject を指す	権限者の認証と承認有効性
artifact SHA-256	文字列の宣言整合	実際の成果物 bytes との一致

ARC v1.0.0 で、どこまで検査できる？

G0

役割 / scope / handoff

役割や範囲の矛盾

G1

subject digest

変更前のレビュー宣言

G2

filesystem-write

書き込み操作の宣言

G3

lifecycle

進行状況の宣言

実際のアクセス権・操作・証拠の真正性は、別の仕組みで確かめる

G0 ～ G3 の対応範囲

区分	何を照合するか	典型的な不一致
G0	role / route / task / handoff	自己レビュー・許可外 scope
G1	bundle + task + action の digest	古いレビュー対象
G2	限定された write_file mapping	path ・ content digest の不一致
G3	lifecycle / artifact の宣言	古い subject ・ digest 衝突

G2： ファイル書き込みの宣言をマッピング

G2 で扱う最小の意味づけ（説明用の抜粋）

```
action.kind = "filesystem-write"  
action.parameters = {path, content_sha256}  
mapping.operation = "write_file"  
mapping.path = action.parameters.path  
mapping.content_sha256 = action.parameters.content_sha256
```

対応する操作は `write_file` の 1 種類。実ファイルの作成や `bytes` の検証は実行しません。

G3：状態と成果物の「宣言」を整合させる

G3 の概念（実行ログの真正性は外側で検証）

<code>run_id</code>	外部から与えられる相関 ID
<code>review / approval</code>	宣言された判断
<code>execution</code>	宣言された実行結果
<code>artifact.id</code>	成果物の識別子
<code>artifact.sha256</code>	宣言されたハッシュ

lifecycle の PASS は本物のイベント・成果物・時刻・リプレイ防止を証明しません。

CrewAI 連携は「設定の差」を読む実験

POINT 01

実験の内容

CrewAI 設定にある `output_file` を
`docs/example.md` →
`secrets/production.txt`
に変え、ARC のルールとのズレを検出。

POINT 02

公開 API との区別

実験コードは v1.0.0 に同梱。
正式な CrewAI API/CLI は未提供。
CrewAI の実行やツール呼び出しもしない。

既存ツールと比べて、ARC の価値はどこにある？

問いは、「作れるか」から「追加の価値があるか」へ。

海外からの質問は、設計を見直すための材料になった。

MartinLoop の作者から、核心を突く問いが来た

Keesan12 / MartinLoop 開発者

「既存の検証器が見逃すものを、ARC は見つけられる？」

MartinLoop

実行時の範囲制御・独立検証・実行記録

ARC

実行前の宣言を照合

比較すること

重複・追加価値・設定時間・別タスクでの再利用

開発者本人による比較結果は、まだ公開されていない

HumanMirror からは、別の厳しい問いが届いた

VibeMirror-coder / HumanMirror

レビュー専用の AI が、

`update_environment_variable`

を実行しようとしたら、誰が止める？

ARC v1.0.0 は、任意のツール操作を直接検査・阻止する機能を持たない

最初の外部試用レポートが届いた

Burs-IA WORKNET / issue #45 / ARC 0.5.0

許可内

ops/.../worknet-outcome-control.mjs

PASS

範囲外

runtime/secrets.txt

FAIL

元に戻す

許可内の path

PASS

外部から報告あり／宣言検査は再現済み／元の CI と別タスクでの再利用は未確認

海外からの声：何が検証され、何が未検証か

相手・出典	実際に確認できたこと	未達・注意点
MartinLoop #37	開発者との評価方針の議論	本人の実行結果は未受領
HumanMirror #37	review-only + mutating tool の論点	比較テストは NOT RUN
Burs-IA #45	意図した scope 違反を検出と報告	元の外部 CI と再利用は未確認
保守者比較 #47	範囲判定 2 件で重複	ARC の独自優位は未検証

2026-10-22 / 保守者が実施した比較

単純な範囲外の要求は、
既存のガードでも見つかっていた。

2 件： 同じ判定

1 件： ARC の書式で表せないパス

課題が重要でも、ARC 独自の価値は実証が必要だ。

既存のガードと比較して見えてきたこと

POINT 01

重なっていた部分

単純なファイル scope 違反は、比較対象の MartinLoop ガードも検出。
レビュー後の内容変更も、既存の artifact verifier が別方式で扱える。

POINT 02

残る検証仮説

role/route/review/approval を実行前の宣言としてまとめられる利点。
設定時間・誤拒否・2 件目の再利用が見合うか測定する。

2026 年 10 月 23 日。ARC の現在地は、ここ。

できていること

v1.0.0 公開
3 分の検査デモ・G0 ～ G3

これから確かめること

追加価値・設定負担
外部での再利用・誤検知

ARC が担わないこと

本人認証・実行時制御
本番権限・証拠の真正性

できることと、まだわからないことが見えてきた。

2026 年 10 月 23 日時点：安定版と実験を区別

対象	確認済みの位置づけ	強制できる権限か
npm 1.0.0	公開済み安定版／宣言検査	いいえ
CrewAI spike	同梱実験・未 export	いいえ
runtime-write pilot #67	Draft PR ・ 隔離 Linux 実験	限定したテスト条件のみ
顧客本番導入	再利用と価値は検証中	受入実績を主張しない

隔離 runtime pilot で新たに検証していること

POINT 01

実験の対象

構造化された `write_file` 要求を、実行直前の検査に接続。
許可された新規作成と拒否時の無変更を、隔離環境でテスト。

POINT 02

公開上の注意

PR #67 は Draft で未マージ。
Linux / Node 24 / overlayfs 等に限定。
正式 QC ・本番有効化 ・一般的な tool 強制の証明は未完了。

価値は、5つの実測結果で判断する

評価軸	知りたいこと	2026-10-23 時点
矛盾の発見	既存手段では見逃したか	実タスクで未証明
人間の再確認時間	何分変わるか	未計測
設定・保守時間	導入に何分必要か	未計測
正当な作業の誤拒否	業務に支障が出ないか	実利用データ不足
2件目の再利用	また使いたいのか	外部で未証明

僕が探していたのは、万能な地図ではなかった

仕事を任せる前後の確認

役割・タスクの整合

ARC

実際の権限制御

IAM ・ sandbox

成果が正しいか

CI ・ 人間

僕はまず、仕事を渡す前に「矛盾を見つける線」を引いた。

この問題、あなたの AI チームでも起きませんか？

今日、持ち帰って試してほしいこと

01

公式デモを動かす

PASS → FAIL → PASS を確認

02

自分の AI の役割と変更範囲を宣言する

implementer と reviewer を分ける

03

許可範囲外のパスを、わざと指定する

手元の CI やガードの結果とも比べる



github.com/suirindo/agent-role-contracts

今日の結論

僕だけが迷子だった。わけじゃない。

世界も同じ課題を持っていた

手元で評価するための 10 分プラン

01

公式デモ

clone して実行。想定した FAIL を確認

02

自分の小さなタスク

scope ・ executor ・ reviewer を宣言

03

わざとズラす

範囲外 scope / 自己レビュー等を試す

04

同じ条件で比較

既存ガードと比べ、時間を測る

向いているケース／今は慎重なケース

POINT 01

試す価値が高い

複数のエージェントへ役割を割り当てる。
独立レビューや人間承認がある。
既存のタスク宣言が散在している。

POINT 02

適用前に確認

実行時の OS/IAM 制御を求めている。
DB ・ SaaS API への操作が必要。
既存ガードが同じ矛盾を検出済み。

よくある質問と短い回答

質問	答え	次に確認するもの
PASS なら実行してよい?	いいえ、宣言が整合しただけ	IAM ・ 承認 ・ runtime guard
レビューは本当に独立?	ロールの分離を宣言上で検査	人と実環境の本人確認
action が変わったら?	旧 binding と digest が不一致	G1 の再検査と再承認
既存の CI で十分なら?	追加価値はない可能性もある	実タスク比較の記録

公開用の一次資料と続きの導線

コードと3分デモ

<https://github.com/suirindo/agent-role-contracts>

使い方と責任境界

[README.ja.md](#) / [docs/QUICKSTART.md](#) / [docs/INTEGRATION.md](#) / [docs/COMPATIBILITY.md](#)

評価と対話

GitHub issues #37 / #45 / #47、Draft 実験 #67、workflow-feedback フォーム

A1 / そのまま試せる Quick Start

```
git clone https://github.com/suirindo/agent-role-contracts.git  
npm --prefix agent-role-contracts run demo --silent
```

前提: Git / Node.js ≥ 22.5 、npm install ・ API キー不要

公式デモは、想定どおりの FAIL も含めて終了コード 0

A2 / 公開されている検査 API (抜粋)

```
import { explainTask, validateBundle }  
  from '@netsujo/agent-role-contracts/core';  
  
const report = explainTask(bundleJson, taskJson);  
console.log(report.valid, report.errors);
```

G1 : describeTaskAction / validateTaskActionBinding

G2 : /adapters/filesystem-write、 G3 : validateTaskLifecycle

A3 / ARC が保証しないこと

```
identity_verified = false
```

```
runtime_enforcement = false
```

```
execution_authorized = false
```

```
evidence_verified = false
```

```
source_files_checked = false
```

A4 / CrewAI 連携の実験（正式な公開 API ではない）

実験用ソース / 未公開 API

```
"output_file": "docs/example.md"  
      ↓  
"output_file": "secrets/production.txt"
```

ARC 側のルールは変えず、出力先だけ変えると FAIL になる

A5 / 根拠と未検証事項

NIST	AI Agent Standards / identity & authorization	2026 年 2 月・ 9 月
OWASP	Agentic Security Initiative	2026 年のリスク整理
ARC #37	MartinLoop / HumanMirror との公開対話	外部による比較実験は未報告
ARC #45	Burs-IA WORKNET 外部報告	別タスクでの再利用は未確認
ARC #47	保守者による比較と価値検証	既存ツールへの追加価値は未検証

一次情報を読む：仕様・手順・デモ

[公開リポジトリ / 3分デモ](#)

README.ja.md と同梱デモの入口

[Quick Start — 自分のscopeを変える](#)

宣言を編集し、CLI で PASS/FAIL を確かめる

[Integration — 信頼・認証・実行の境界](#)

実際に組み込むときの責任分担

[Compatibility — v1.0.0の公開契約](#)

公開 API ・ schema ・ digest の整合性

[External Evaluation Kit](#)

同じ条件で検証するための実験キット

一次情報を読む：世界の議論・外部評価

[NIST — AI Agent Standards Initiative](#)

2026-02-17 / 米国 NIST の研究・標準化の取り組み

[NIST — Agent Identity / Authorization](#)

2026-09-29 / コメント総括と DevSecOps ユースケース

[OWASP — Agentic Security crosswalk](#)

2026-05-25 / リスク分類への対応表

[ARC #47 — 実利用評価](#)

2026-10-23 / 比較・人間の時間計測・再利用判断

[ARC PR #67 — Runtime pilot \[Draft\]](#)

2026-10-23 / 隔離実験・安定版の機能とは区別

一次情報を読む： GitHub での公開対話

[Issue #37 — MartinLoop / HumanMirror](#)

海外エンジニアとの議論。比較の計画と未実行の境界

[Issue #45 — Burs-IA WORKNET](#)

外部試用レポート。独立再検証と再利用は別

[Issue #47 — 10ワークフロー価値検証](#)

保守者比較、実測評価、残る 5 つの判断材料

[Issue #52 — v1.0.0 公開受入](#)

2026-10-16 / npm stable 公開の正本

[PR #67 — 隔離runtime pilot \(Draft\)](#)

2026-10-23 / 安定版への統合・本番適用は未完了

公開版の読み終わりに

次に確かめるのは、
あなたの現場での追加価値。

まず 3 分の宣言検査を体験し、実際のタスク・既存の CI/IAM との重複・作業時間・2 件目の再利用を記録してください。報告が「既存の仕組みで十分」でも、次の判断に役立ちます。