

AIに気持ちよく働いていただく技術

「お願い」で終わらせない、ハーネスエンジニアリングの実践

Takuya Suemura @ Ubie株式会社

自己紹介



- 末村 拓也
- Ubie株式会社 Software Engineer in Test
- 自動テストに詳しいエンジニア
 - 前職は自動テストSaaS

JaSST Online



TOKYO TEST FEST

Practical guide to test automation

テスト自動化 実践ガイド

継続的にWebアプリケーションを改善するための知識と技法

末村拓也 著
Takuya Suemura

E2Eテスト自動化の

検討 導入 改善をサポート!

テスト担当者・開発者が知っておくべき、自動テストの効果を高める「勘所」
フレームワークに依存しない普遍的なテクニックが身につく



SE
SHOESHA

FULL STACK TESTING

✦ Gayathri Mohan ✦ 末村拓也 堀明子 松浦隼人

フルスタック テストイング

全方位型ガイドで
品質を底上げせよ。

- 10 モバイルテスト
- 09 機能横断要件テスト
- 08 アクセシビリティテスト
- 07 パフォーマンステスト
- 06 セキュリティテスト

10の
テスト手法で
実践する
高品質
ソフトウェア開発

- 05 ビジュアルテスト
- 04 データテスト
- 03 継続的テスト
- 02 自動機能テスト
- 01 手動探索的テスト

SE
SHOESHA

原理 戦略 実践手法 を網羅する一冊

前置き

AIに仕事させるのはもはや普通のこと

少し前

- ChatGPT などに **質問・依頼**
- 文面の下書き、コード補助
- 結果は **人間がコピペして使う**

いま

- **AIエージェント** が前提に
- ファイル編集・外部ツール・権限つき
- **自律的に** 仕事をやり切る

AIエージェントが扱えるタスクは コーディングだけではない

UbieにはこんなAIエージェントがいます

- セキュリティアラートを出したり調べたりしてくれるAI
- インフラ周りを何でも調べてくれてバグも直してくれるAI
- n8nのワークフローを作ってくれるAI

AIエージェント自体も プロダクトになる

例) LayerX『バクラクヘルプデスク』



経費申請の締切はいつですか？



経費申請の提出期限は以下の通りです。
支払発生日の翌月第3営業日までに申請を行ってください。
期限内に申請されたものについては、翌月の給与支給日にお支払いされます。

経費申請の提出期限について回答しました。疑問は解決しましたか？

ヘルプデスクエージェント

社内問合せ対応を 自動化する

質問やマニュアル更新も
Slack で完結

＼5分でわかる資料をプレゼント／
メールで資料を受け取る

しかし、AIはやらかす

























- `--dangerously-skip-permissions` のまま `rm -rf ./`
- 「リポジトリを整理して」が `.git` ディレクトリごと削除 に
- 本番 DB に `DROP TABLE` を打ち込まれて全消失
- **API キー** を含んだコミット

「壊さないで」「顧客情報漏らさないで」
ってプロンプトで **お願い** すれば十分？

今日話すこと: AIエージェントの品質保証

AIはやらかすかもしれませんが……

人間様はもちろん間違えませんか？

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
	 @lif_yumumu <u>客先のDBを間違えて削除した話</u>	 @ryo_s0127 <u>スパゲティプログラムに3行追加したら決済ができなくなった話</u>	 @waricoma <u>コンセント配線で9万円を無駄にしかけた話</u>	 @namamurasub <u>メモリをモリモリにした話</u>	 @iikaiki <u>客先本番環境のNASデータをすべてrmした事件</u>	 @katohiro_fi <u>ADサーバーが消えた日</u>
7	8	9	10	11	12	13
 @katokonigiry <u>完了したと思っていたリプレイスに、ひそんでいた見落とし</u>	 @infinity0206 <u>メールサーバーを爆撃して部門長宛のメールを消失させた話</u>	 @tampopo256 <u>コードミスで自社メールサーバーを受信不能にした話</u>	 @piepie1231 何か書きます <div style="border: 1px solid black; padding: 5px; display: inline-block;">Overwrite</div>	 @okakajp_25 <u>本番と開発を間違えた話</u>	 @nekihcom <u>思い込みが招いたサービス稼働停止</u>	 @kaizen_nagoya <u>A story about messing up in production environment.</u>
14	15	16	17	18	19	20
 @nfujita55a <u>プログラムに1文字加えて本番環境のバッチ処理を4倍遅くした</u>	+	 @kako-jun <u>誤ったバンプコーディングでOSSが3ヶ月停滞したので、リポートで復活させた全記録</u>	 @MaseBB <u>間違って電源ぶち抜いてしまい、学園祭の全配信システムを止めた話</u>	 @harumaki2000 <u>操作ミスでめちゃくちゃになり、最終的にサ終した話</u>	 @Ayahira_0319 <u>自分で作ったSNSを自分で壊した話</u>	 @papillon <u>おきのどくですがファイルシステムはきえてしまいました</u>
21	22	23	24	25	26	27
 @Tatsu1567 思い出の詰まった大学サークルの記録DBをバックアップ取らずに全消した話 <div style="border: 1px solid black; padding: 5px; display: inline-block;">Overwrite</div>	 @ocnito 他社の本番環境で署名検証がすり抜けクビになった話 <div style="border: 1px solid black; padding: 5px; display: inline-block;">Overwrite</div>	 @kizuki1749 <u>「実家Wi-Fi、毎日落ちる。」——実家ネットワークを“やらかし本番環境”化した中学生時代の話</u>	 @mpcode_dev <u>他人が使うソフトを人生で初めて構築してとんでもないトラブルを2つも起こしてしまった話</u>	 @sevenc-nanashi <u>うっかりミスでアクティビュユーザー1万人超えのサービスをサ終させた話</u>		

再発防止策あるある

- ちゃんとやります層
 - ちゃんとやります!
 - 気をつけます!
- チェックリスト層
 - チェックします!
 - ダブルチェックします!

AIエージェントで置き換えると

- ちゃんとやります
 - プロンプトの工夫
- チェックリスト
 - AGENT.md

AIにも人間にも仕組みが必要

人間向け: フールプルーフ、ポカヨケ

AI向け: **ハーネス**

AIエージェントの例

Ubicで動いているAIエージェントたち（一例）

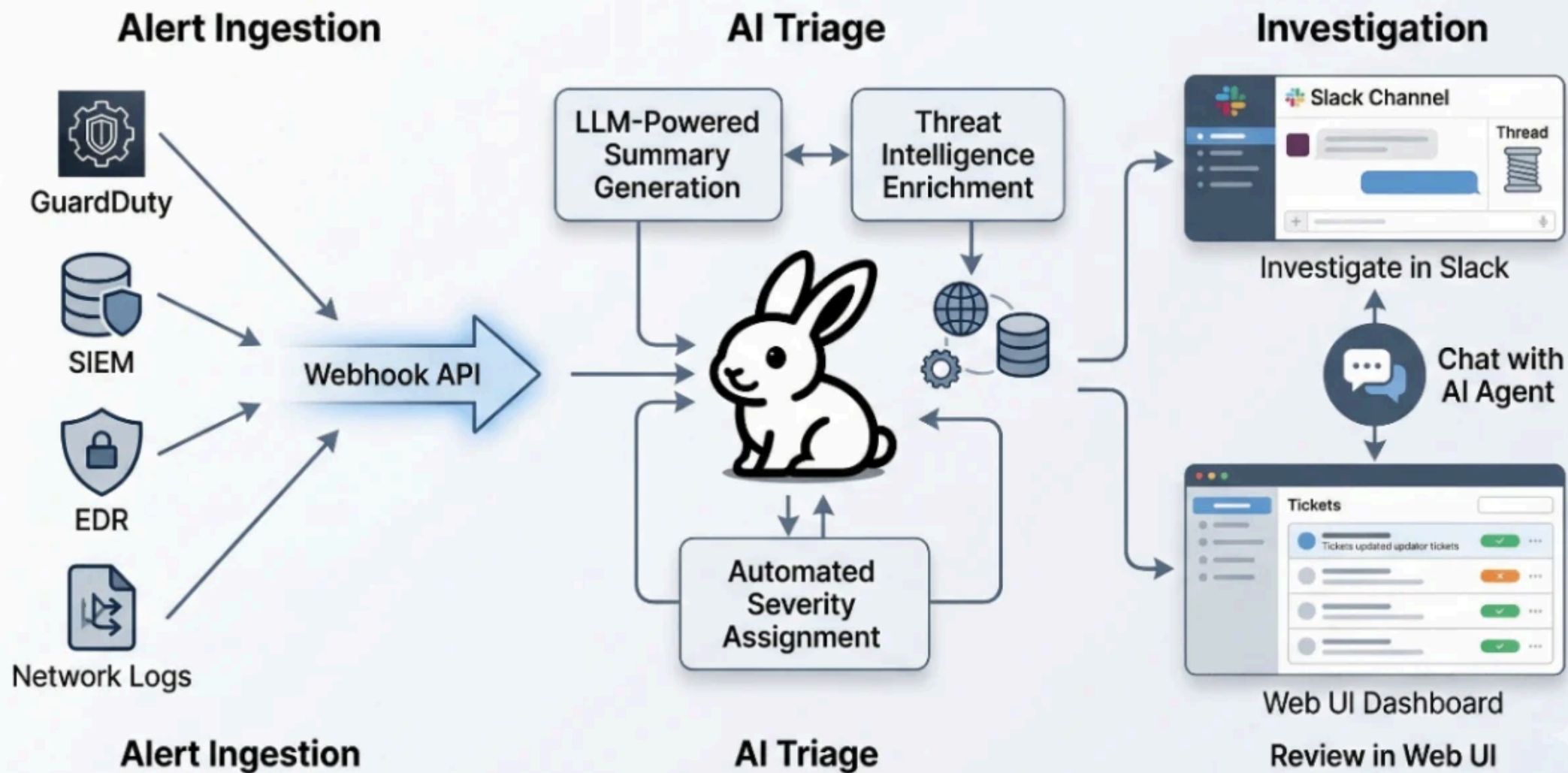
🐰 Warren

セキュリティアラートを出したり調べたりしてくれるAIエージェント

🐢 infra-agent

インフラ周りを何でも調べてくれてバグも直してくれるAIエージェント

Warren: AI Security Alert Triage





mizu-san 1 minute ago

@warren 直近24時間ほどで不審な挙動がなかったか調査して



warren APP 1 minute ago

Interpreting ... (ID: 019d861c-0e99-750e-8eb3-8cbe5f2f91f5)

Prompt: (default)

Found 8 knowledge entries

@mizu-san 了解しました。プロジェクト [redacted] およびクラスター [redacted]-gke-admin に対する直近24時間の不審な挙動を調査します。GCP監査ログ（管理操作・データアクセス）を並行して確認します。

[直近24時間の管理操作ログ確認]

Executing query:

```
SELECT
  protopayload_auditlog.authenticationInfo.principalEmail,
  protopayload_auditlog.methodName,
  protopayload_auditlog.serviceName,
  COUNT(*) as cnt
FROM [redacted].activity`
WHERE timestamp BETWEEN TIMESTAMP('2026-04-12T00:00:00Z') AND TIMESTAMP('2026-04-13T10:00:00Z')
  AND resource.labels.project_id = [redacted]
GROUP BY 1, 2, 3
ORDER BY cnt DESC
LIMIT 100
```

[Compute Engineデフォルトサービスアカウントのアクティビティ確認]

infra-agent

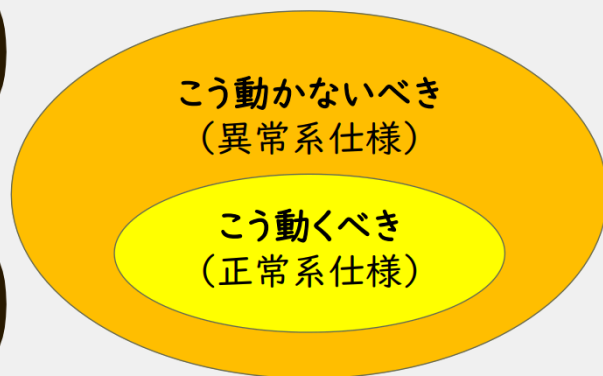
- Slackで @infra-agent で呼べる
- 「デプロイ失敗したんだけどどうして?」「権限追加してほしい」などのリクエストに対応
- インフラ関連のログ、エラー、GitHubなどにアクセスできる
- 賢すぎてバグすら直せてしまうので最近はとりあえず何でも infra-agent をお願いすることが多い

「勝手に動くもの」をどうやってテストすればいい？

- ルールは決まっているが、破るかもしれない
- 予想外の更に予想外をやるかもしれない

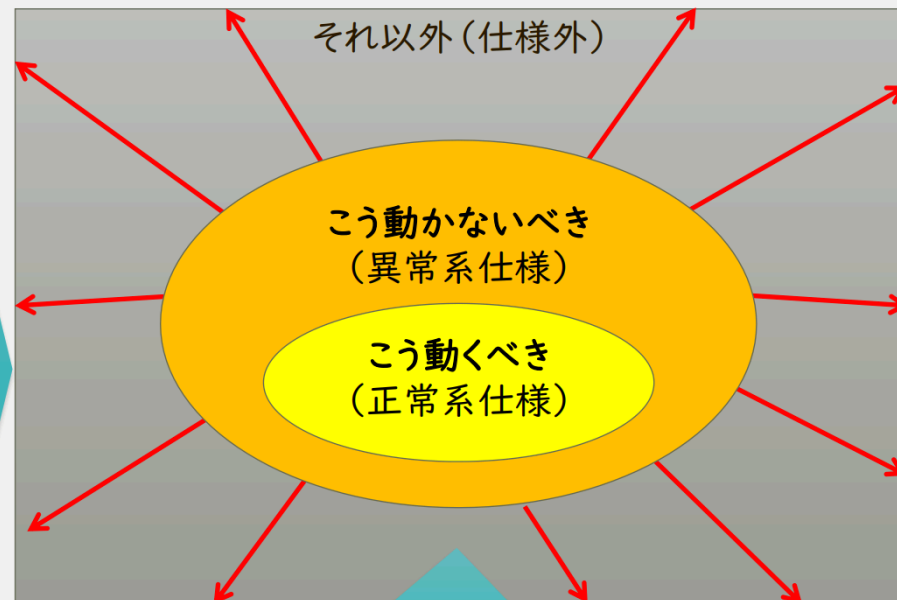
テスト時の思考

テストは仕様化された事柄について「その通りに動くか」を確認するだけでは不十分で、「それ以外で何も起きないのか」も確認せねばならない。



仕様を確認するだけでは
単なる**動作チェック**

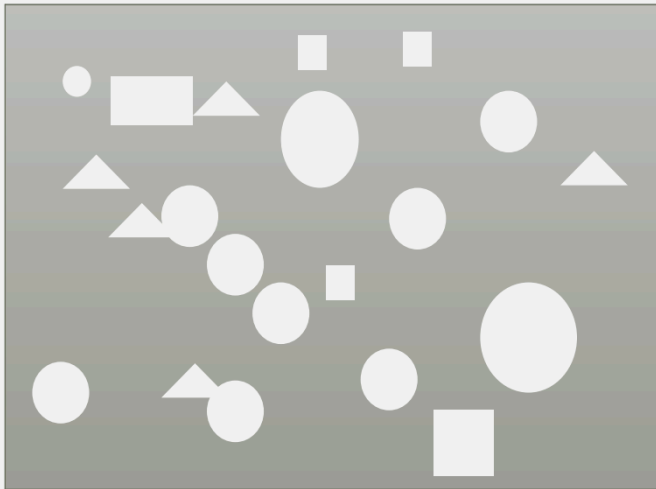
「仕様化された事柄について「その通りに動くか」を確認するだけでは不十分で、「それ以外で何も起きないのか」も確認せねばならない。」



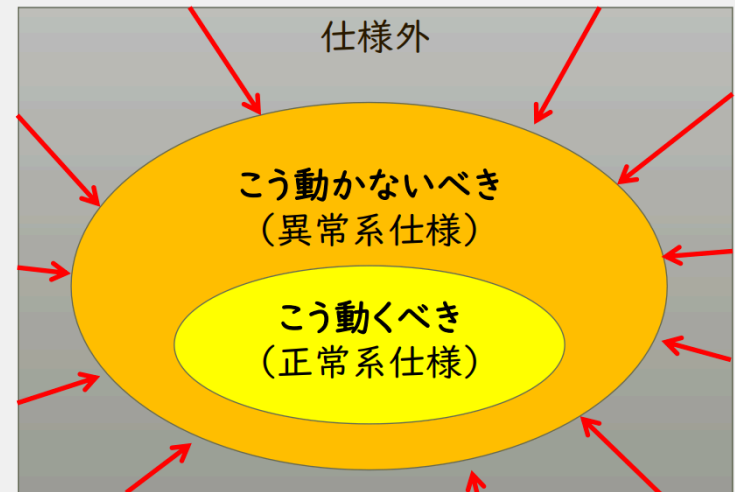
テストではむしろ
「仕様外」を扱うことが重要

設計時の思考

設計では要件を、計算機世界上で「こう動くべき」「こう動かないべき」に分類・具体化・定義することで問題領域を狭めていく（計算機の振る舞いを定義する→仕様化）



「こう動くべき」
「こう動かないべき」
に定義していくことで
計算機として扱う問題領域
を狭めていく



注意: わかりやすさを優先した図であって、ベン図ではありません

■ テストと構造的保証

テスト (Checking + Exploring)

動かして確認する 領域

担保手段: 自動テスト + 探索的テスト

例: API契約、入出力、回帰、セキュリティ、UX、
未知の組合せ

構造的保証 (設計・ハーネス)

「このように動作すべき(しないべき)を強制する」領域

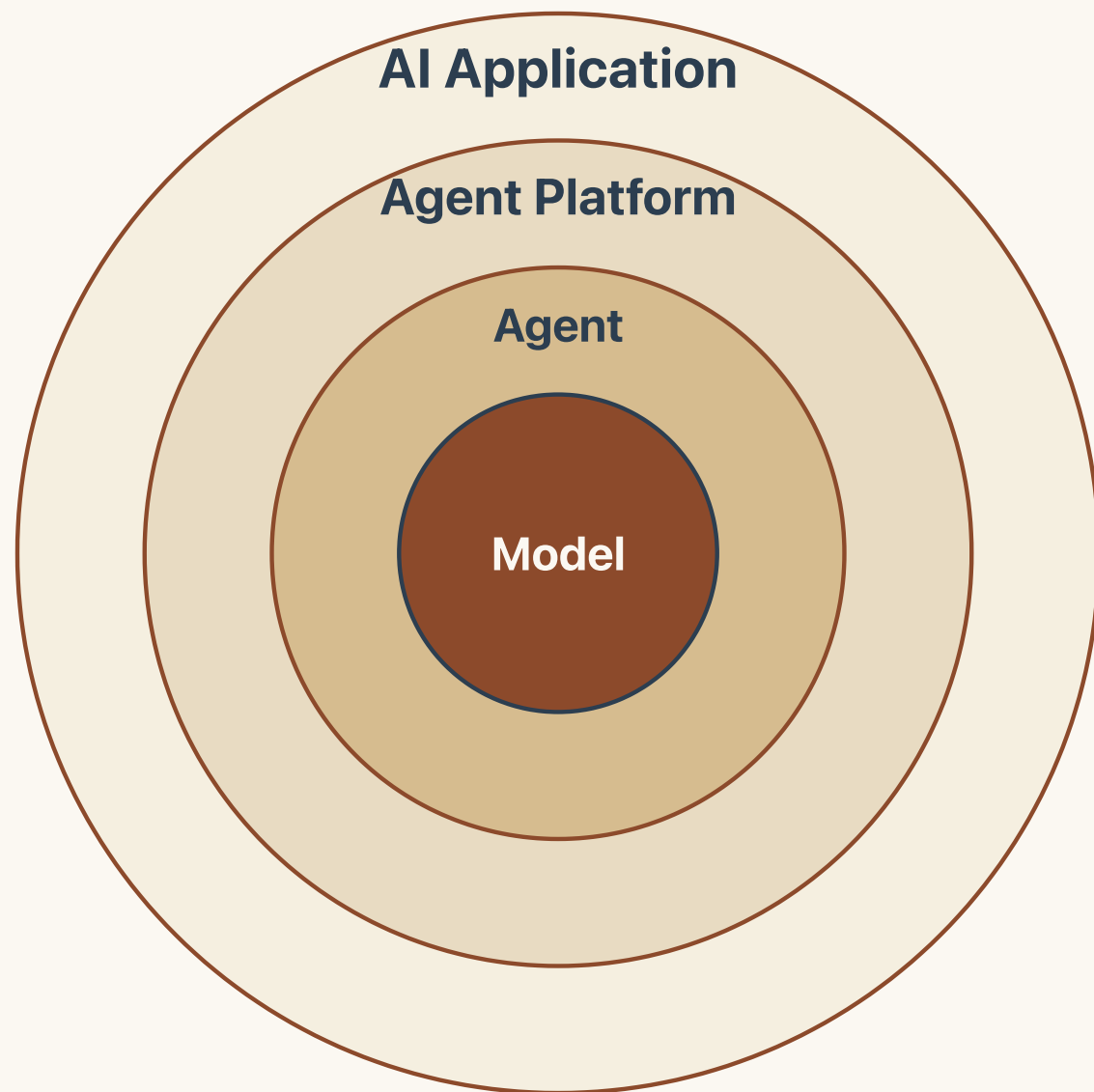
担保手段: 構造的制約

例: 権限境界、通信先制約、認証情報の隔離

- 問題領域が極端に広い、自律的に動作するものに対してのテストには限界がある
- 問題領域を狭めるために **アプリケーション設計** を用いる
 - これをAIエージェント開発の文脈では **ハーネスエンジニアリング** と呼ぶことが多い

ハーネスの実例

多層防御の考え方をを用いる



層	役割	ハーンズ例
Model	LLMそのもの	モデル選択 サンプリングパラメータ safety filter
Agent	推論・記憶の管理・ツールオーケストレーション	
Agent Platform	実行環境	コンテナ 透過プロキシ Hook
AI Application	業務ルール	HITL プロンプト構造化 決定的アプローチ リントア e2eテスト SKILL.md

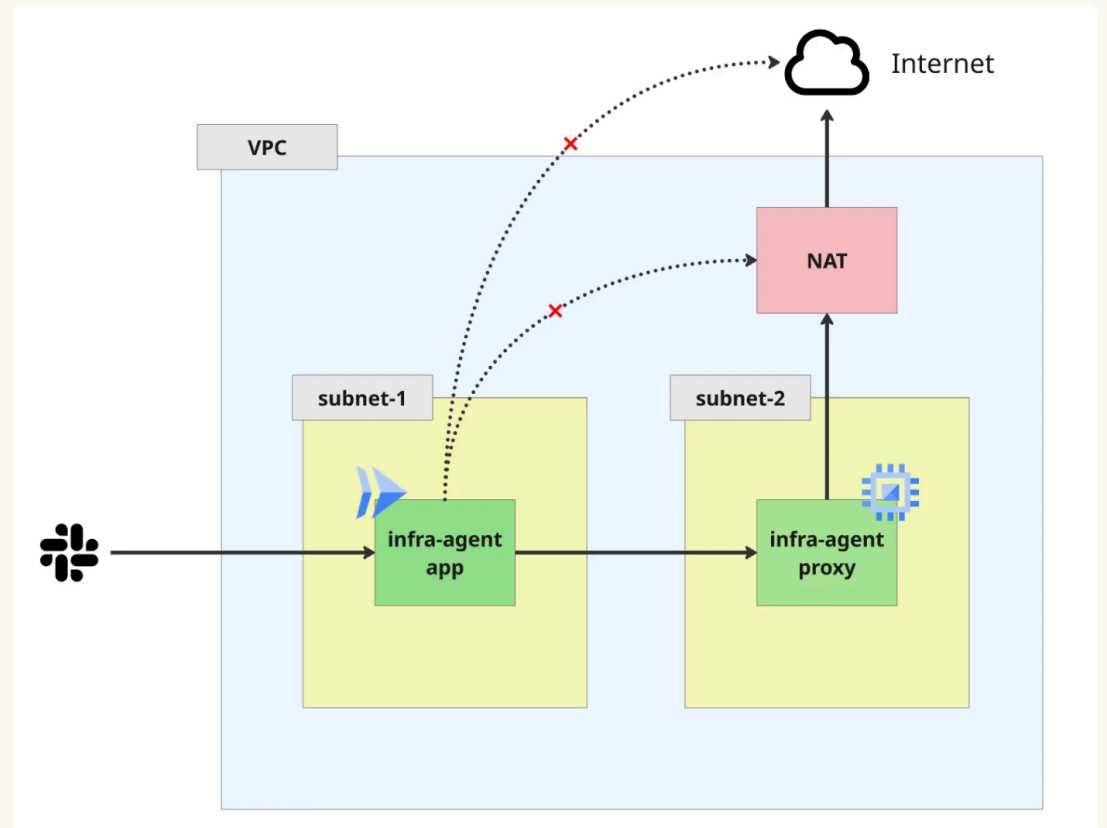
安全で柔軟な制御を実現するAgent Platform

ネットワーク制御

- infra-agentの通信は全て `mitmproxy` を経由する
- 単純にホストレベルで制御するのではなく、
 - たとえば、`github.com` を一律アクセス不可とするのではなく、自社のリポジトリだけにアクセスできるようにする

権限管理

- infra-agent はGoogle CloudのCloud Run上で動作し、最小限の権限だけを付与している



決定的アプローチ

```
package ingest.scc

alerts contains {} if {
  not ignore
}

# Log4j の脆弱性是对処済みなので問題なし
ignore if {
  input.finding.category == "Initial Access: Log4j Compromise Attempt"
}
```

Warrenではアラートの取り込みルールは **LLM に判断させず、ポリシーで決定的(Deterministic)に処理**

→ 決定的な判定が必要な部分は、そもそも LLM に触らせない

HITL (Human-in-the-loop)

- 外部にデータを送信する可能性のあるツールは **人間の承認が必須** になっている
 - Slack の対話的ボタンで人間に承認を求める仕組み
- ツール呼び出しにHITLを強制させるようにしつつ、

補足: 判断はLLMだが実行は(人間が実装した)ツール

ツール——AIエージェントの「手足」

ここで基本原則があります。「判断するのはLLM、実行するのは人間が実装したツール」です。LLMは自然言語を理解し、どのツールを使うべきかを判断しますが、実際にファイルを読んだり、コマンドを実行したりするのは、人間が実装したTypeScript関数です。この分離により、安全性と制御性を確保できます。

laiso. 作って学ぶAIエージェント——TypeScriptとLLMで切り拓くAI時代のエンジニアリング エンジン選書 (pp. 55-56). Kindle Edition.



設計で防げないものはテストする

- モデル変更による出力変化と、それに伴うデグレード
- 出力の妥当性、ハルシネーションなどの検知
- AIエージェントのワークフローをテストابلにする
 - ≡ 設計段階でテストビリティを考えないといけない
 - LangGraphなど

LLM一般の話も多いので、松木先生の本を買おう！

まとめ

- 設計段階で問題領域を絞るとテスト範囲を最小限にできる
- 複数の層で保証する(多層防御)が重要
 - テストも設計もあくまで層の一つ、とも言える
- 何をどの層で守るかを考えて最適な手段を取ろう

人もAIも環境があっからこそ気持ちよく働ける

- 裁量があるだけではただの暴走超特急
- 「ちゃんとしろ!」で改善するならプロセスはいらない
- 気持ちよく爆速で働くためには、人もAIも **環境** が重要になる
- AIにとっての環境はすなわちソフトウェアの設計
 - 人間も雑な設計の上で仕事するのやだもんね

Enjoy Testing...?

Enjoy Prompting...?

Enjoy Engineering!
