

LiveBreach

全エンジニアを「守れる」開発者へ。開発プロセス融合型・自動攻撃シミュレーター

1. プロダクトコンセプト

ユーザーが書いたコードをリアルタイムで「ハッキング」し、自分の書いたコードが原因で情報が漏れる瞬間を眼の前で体験できる、衝撃型のセキュリティ学習ツール。

2. 背景と課題

社会的背景

サイバー攻撃被害が頻発している中、DevSecOps や Shift Left の潮流により、一般のアプリケーションエンジニアにも高いセキュリティ意識と実装能力が求められている。### 既存課題静的解析ツール (SAST) の警告は、実害がイメージしにくいいため無視されがちである。また、フレームワークの機能だけでは防げない「ロジック (仕様) の脆弱性」は、従来のツールでは見落とされやすい。

3. 解決策

「検知」ではなく「実証」。無機質な警告ではなく、開発中のローカル環境で実際に攻撃を成功させ、データが漏れる様子を可視化することで、開発者に強烈な原体験を提供する。

技術スタック

- **Frontend:** Next.js + Electron (UI 表示・デスクトップアプリ化)
- **Analysis Engine:** ts-morph (TypeScript コンパイラ API を用いた構造解析)
- **Attack Engine:** Puppeteer (Headless Browser による DOM 操作・攻撃実行)
- **AI Engine:** Ollama (ローカル LLM 実行基盤) ※セキュリティ特化モデル WhiteRabbitNeo 等を採用

ターゲットシステム

Next.js (App Router) + Prisma ※現代の Web 開発で主流の構成であり、かつ Server Actions 等にロジック脆弱性が入り込みやすい点に着目。

コア技術: Context-Aware DOM Injection

Next.js Server Actions の通信 (Flight Protocol) は解析が困難であるため、通信経路上での改ざん (MITM) は行わない。代わりに、**ブラウザ上の DOM 層で攻撃コードを注入**することで、フレームワーク内部の複雑なシリアライズ仕様に依存せず、Mass Assignment 等の攻撃を成立させる独自手法を採用する。

動作の流れ

1. **Launch:** 開発中にデスクトップアプリとして起動
2. **Scan:** プロジェクト内の `schema.prisma` や `actions.ts` を解析し、攻撃対象候補を特定 (AST 解析)
3. **Approve:** ユーザーに攻撃実行の許可を求める (意図しない破壊の防止)
4. **Exploit:** Puppeteer がバックグラウンドでブラウザを操作・改ざんし、攻撃を成功させる
5. **Educate:** 攻撃成功後、AI が「なぜ漏れたのか」の解説と修正コードを提示する

処理パイプライン

フェーズ	技術スタック	詳細	速度
1. 候補抽出	AST (ts-morph)	プロジェクト全体から「DB 操作を行っている Server Action」等を瞬時に特定し、コード片を切り出す。	爆速
2. 精密解析	AI (Ollama)	切り出されたコード片のみを LLM に入力し、「バリエーション欠如」や「権限チェック不足」を判定させる。	中速
3. 攻撃生成	AI + Rule	脆弱性と判定された場合、具体的な攻撃ペイロード (JSON) を生成し、Puppeteer 操作スクリプトに変換する。	爆速
4. 攻撃実行	Puppeteer	アプリを自動操作し、DOM 改ざんや URL 操作を行ってリクエストを送信。	高速
5. 結果判定	Logic Check	ステータスコードだけでなく、レスポンスの DOM 構造を比較し「データ漏洩」を自動判定。	高速
6. 解説生成	AI (Ollama)	脆弱性の根本原因と、Zod 等を用いたセキュアな実装例を生成・表示。	低速

4. 狙う脆弱性と攻撃ロジック

- **Mass Assignment (DOM Injection)**
 - `schema.prisma` から `isAdmin`, `role` などの機密フィールドを抽出。
 - フォーム送信直前に Puppeteer が `<input type="hidden" name="isAdmin" value="true">` を DOM に注入し、権限昇格を実証。
- **IDOR (Insecure Direct Object References)**
 - URL パラメータが `Int` 型 ID の場合、ID ± 1 へのアクセスを自動試行。
 - 本来閲覧できない他人のデータが表示された場合を検知。
- **Broken Access Control (Middleware Bypass)**
 - `middleware.ts` の設定漏れを突き、未認証状態で保護ページ (`/dashboard`) への直接アクセスを試みる。

5. 安全性・倫理設計

- **Localhost Binding:** Puppeteer の接続先をローカル環境 (`127.0.0.1`) にハードコードで制限し、外部サイトへの攻撃を防止。
- **Non-Destructive:** データの削除・破壊 (`DELETE`, `DROP`) は行わず、情報の閲覧・権限変更のみを実証範囲とする。
- **Seed Injection:** 開発初期の「データが空で攻撃できない」問題を解決するため、起動時にテスト用ダミーデータ (Seed) を一時的に注入する機能を提供。

6. 技術的な挑戦と解決策

- **課題:** Next.js Server Actions の通信 (Flight Protocol) はバイナリ解析が極めて困難。
- **解決:** 通信レイヤーではなく、**DOM レイヤーでの改ざん**にシフトすることで、フレームワークのバージョンアップや内部仕様変更強い堅牢な攻撃エンジンを実現。
- **課題:** ローカル LLM の推論待ち時間が UX を損なう。
- **解決:** 攻撃実行と判定はルールベースと軽量な解析で行い「即時フィードバック」を実現。重い AI 処理は事後の「解説生成」のみに分離する非同期アーキテクチャを採用。

7. まとめ

- **新規性:** 「守り方」ではなく「攻められ方」を体験させることで、能動的な学習を促す逆転の発想。
- **技術点:** 高速な AST 解析と柔軟な AI 推論を組み合わせたハイブリッド構成、および DOM Injection によるモダンフレームワーク攻略。
- **目標:** セキュリティを「インフラエンジニアの仕事」から「全開発者の自分事」へと変革する。