SecHack365

開発環境に馴染むWebアプリ向けFuzzingツール SecHackFuzz

SH FZ

開発現場でFuzzing導入を進めてセキュアなWebアプリを当たり前に

開発駆動コース 仲山ゼミ 赤松宏紀

Fuzzingとは

Webアプリ

診断を何度も外注できない

あまりセキュリティを考えない

安全性を少ない手間で確かめたい
→ 機械的に検査できるFuzzing

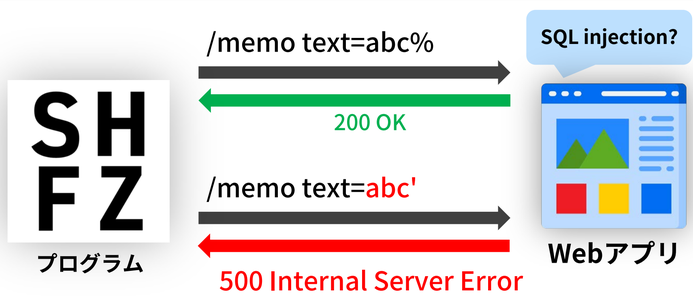
Fuzz(問題を引き起こしそうなデータ)**を大量に送り込む**

Webアプリ内の**不適切な処理に起因するエラー**を機械的に見つける検査手法

→ 既存のFuzzingツールはWeb開発者にとって使いにくい

開発現場に導入されるためには

Web開発者目線のFuzzingツールが必要

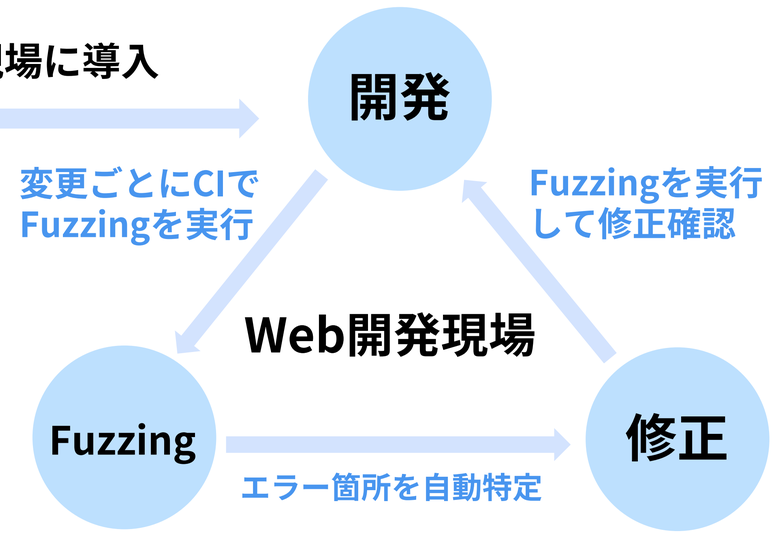


SecHackFuzzで実現する継続的なセキュアWebアプリ開発

SH FZ

SecHackFuzz

CIで開発現場に導入



SecHackFuzzはOSSとして公開中
https://github.com/shfz

SecHackFuzzをCIで開発環境に組み込む

↓

開発~Fuzzing~修正のサイクルを継続的に回す

↓

セキュアなWebアプリが当たり前に

導入から実行・修正までWeb開発者目線

① どんなWebアプリでも
② Web開発者が導入・設定
③ 少ない時間で
④ 修正までサポート

① どんなWebアプリでも検査可能

既存のFuzzingツールの限界

CSRFトークン

複数のリクエストを送る

ワンタイムパスワード

可変パラメータの生成

複数のリクエストや可変パラメータが必要なWebアプリは
既存のFuzzingツールでは検査できない

TypeScriptのシナリオを記述してCLIプログラムで実行

(またはJavaScript)

```
// 1度ページを表示し、CSRFトークンを取得する
let res = await sh.http.get("register page", "/register");
let $ = c.load(res.data);
let csrf_token = $('input[name="csrf_token"]').val()

// CSRFトークンを使ってユーザー登録APIをテストする
await sh.http.postForm("register api", "/register",
  { user, pass, csrf_token: csrf_token }
);


// シークレットからワンタイムパスワードを生成
const totp = require("totp-generator");
const otp = totp(totp_secret)

// ワンタイムパスワードを使用してログインAPIをテストする
let res = await sh.http.postForm("login api", "/login",
  { user, pass, totp: otp }
);


// ページにユーザー名が正しく表示されているか確認
let $ = c.load(res.data);
if($('p[id="user"]').text() !== user) {
  await sh.http.error("No username in response");
};
```

複数のAPIを検査可能

複数のAPI呼び出しを1つのシナリオ内で記述できるため、CSRFトークンなどの複数のリクエストが必要なWebアプリを検査可能

外部パッケージによる拡張

シナリオへnpmパッケージを導入してワンタイムパスワードを始めとするWebアプリの様々な仕様に対応可能

言語に付属する柔軟な条件設定

HTTPステータスコード以外にも、画面に表示された文字や遷移先のページなどをエラー判定の条件に

② Web開発者が導入・設定しやすく

SH FZ SecHackFuzz — GitHubから**バイナリをダウンロードするだけ**で実行
CI上で実行可能なCLIツール

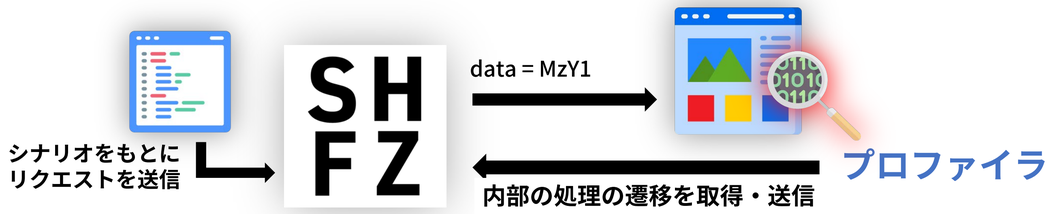
シナリオ — **Web開発者が使い慣れている言語**で記述

パッケージ管理ツールで導入&シナリオを拡張

プロファイラ — **パッケージ管理ツール**で導入

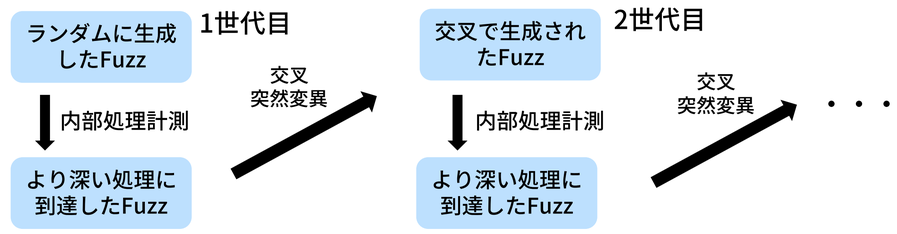
③ 遺伝的アルゴリズムによるFuzzingの効率化

Webアプリの**内部の処理の遷移を取得**するプロファイラをWebアプリに導入



より深い処理まで到達したFuzzから**遺伝的アルゴリズム**によって
深い処理に到達する可能性が高いFuzzを生成

(参考) 遺伝的アルゴリズム



世代が進むたびにより深い処理に到達するFuzzが生成される確率が高くなる

④ 修正作業をサポートするエラー箇所自動特定

修正作業でFuzzingの膨大なログからエラーを引き起こすFuzzを取り出し、
どこでエラーが発生するのか1つずつ特定する作業は大変

→ プロファイラでエラーの発生箇所を自動特定**し、**
まとめたレポートをGitHub Issueに自動投稿

→ このIssueをもとにすぐ修正作業へ

今後

Fuzz生成機構の精度向上・プロファイラの対応Webフレームワークの拡張などを行い、SecHackFuzzの長所をさらに伸ばすことで
「Webアプリ開発現場でFuzzingが当たり前に」を目指す

NICT 国立研究開発法人 情報通信研究機構

SecHackFuzz : https://github.com/shfz　画像 : Flaticon https://www.flaticon.com/