БУДУЩЕЕ
В НАШИХ
РУКАХ

Фаззинг-тестирование Go: как собрать свой вариант велосипеда и успешно на нем поехать

Иван Золотников

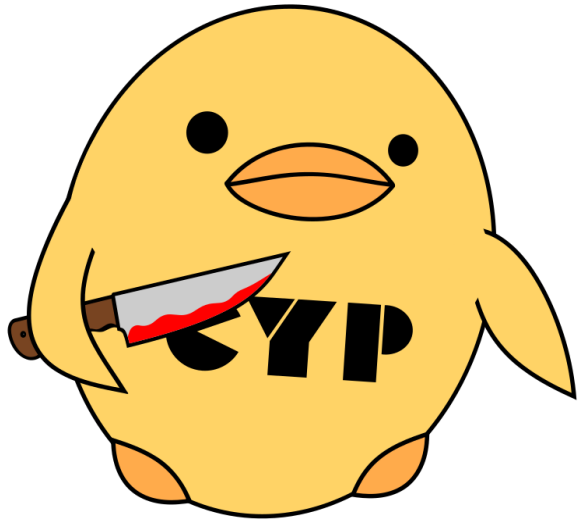
Младший инженер,
Департамент разработки платформы,
YADRO

Кто такой CYP?

CYP (Common Yadro Platform) — это инженерная платформа для построения продуктов YADRO

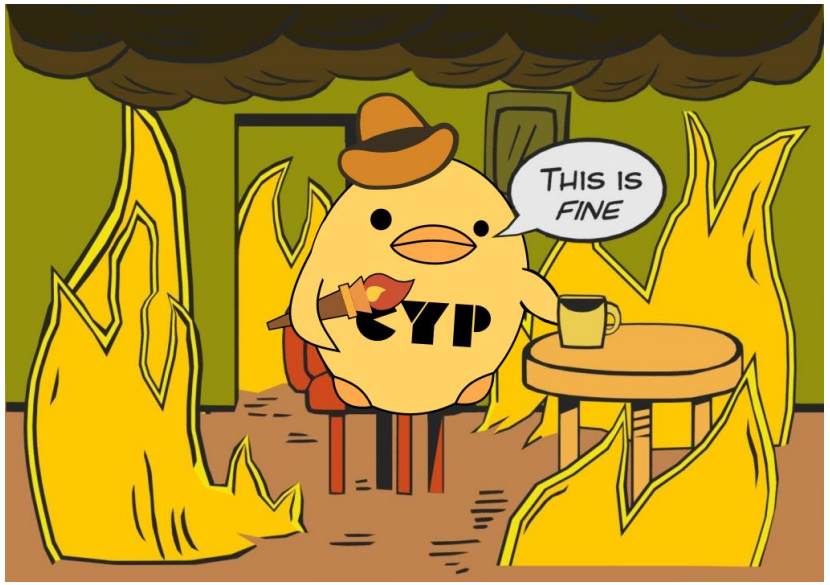
CYP обеспечивает:

- контроль и воспроизводимость сборок
- минимизацию усилий проектных команд
- общее управление технологиями в проектах
- общие сервисы



Постановка задачи

«Пройти процедуру сертификации написанного нами кода на Golang, который используется другими продуктами. Организовать процесс фаззинг-тестирования нашего Golang-кода и автоматизировать его. Выполнить требования к процессу фаззинг-тестирования, описанные в ГОСТе по динамическому анализу»



Что же такое фаззинг?

Фаззинг – это техника автоматического тестирования, при которой на вход исследуемого кода подаются заведомо неожиданные или случайные данные

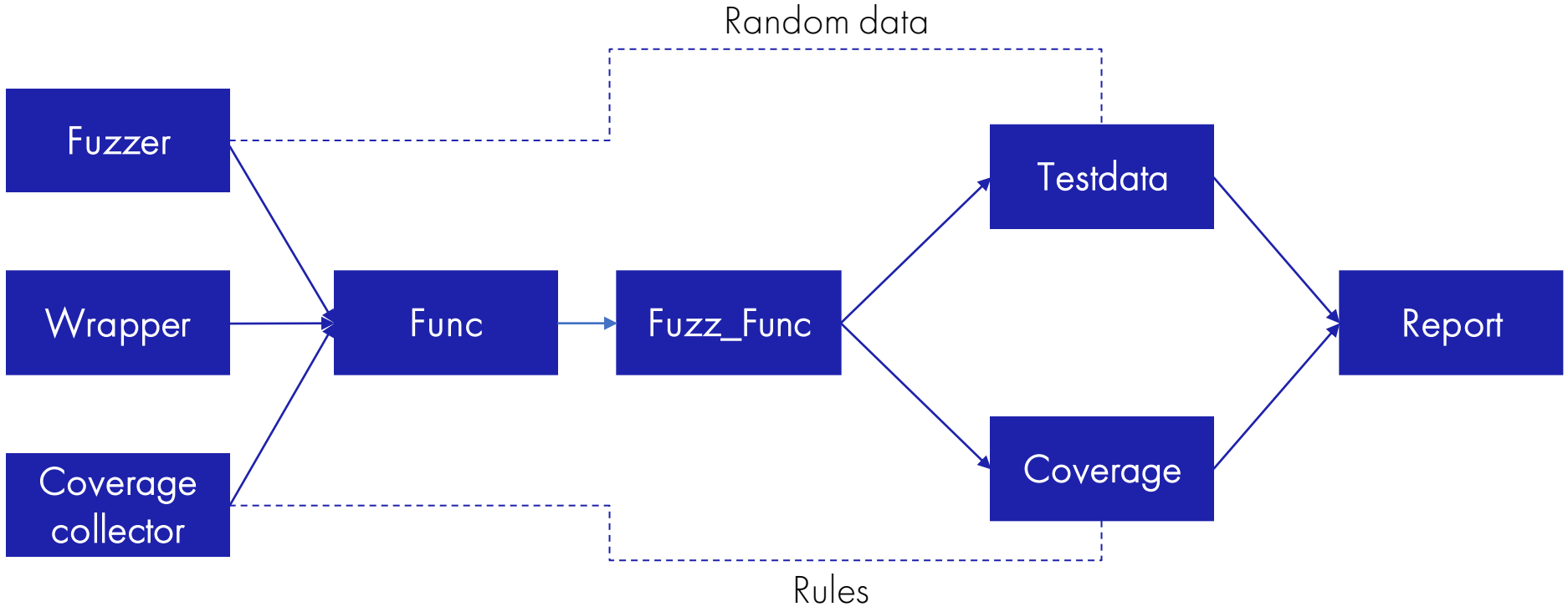
Входные данные формируют **корпус** – набор сгенерированных фаззером тестовых вводов

Цель для фаззинга – исследуемый код, приложение или сервис

Фаззинг-тест – обертка, позволяющая передавать входные данные на вход цели и запускать процесс тестирования

Покрытие – набор строк исходного кода, вызванных в результате использования корпуса

Как фаззинг работает?



Какие алгоритмы фаззинга бывают?

Данные

- White-box
- Grey-box
- Black-box

Операции

- Генерационный
- Мутационный
- Комбинированный

Связь

- Feedback-driven
- Not feedback-driven

Цели фаззинг-тестирования

Сертификация продукта

- 80% кода
- 1.5 млн итераций
- Регулярное фаззинг-тестирование

Дополнительное тестирование

Автоматизация регулярного тестирования

Фаззинг и Go

Go-fuzz от D. Vyukov

Go test –fuzz

- Ngolo
- Fzgen

Libfuzzer (CGO)

Go-fuzz

Плюсы

- Генетический алгоритм с обратной связью
- Не требует перезапуска
- Высокое покрытие

Минусы

- Нет стандартизированного формата обертки
- Требователен к настройке
- Существенно устарел

Go-fuzz

Является попыткой привнести возможности Libfuzzer

Предлагает раннюю возможность интегрировать фаззинг в Go

Появился примерно к версии Go 1.14

Хорошая отправная точка для исследования фаззинга

Go Fuzz

Плюсы

- Содержится в базовом пакете
- Легко запускается
- Поддерживает генераторы

Минусы

- Требует перезапуска
- Требует большего времени на тестирование

Go Fuzz

Официальная интеграция фаззинга в Golang

Использует возможности пакета testing

Появился к версии Go 1.18

Аналог обычных тестов

Fzgen

Инструмент для генерации фаззинг-тестов для Golang

Имеет очень ограниченные возможности

Использует AST

Неудобен в использовании

Возникшие проблемы с Fzgen

На начальном этапе самым доступным решением выглядела борьба с недостатками Fzgen'а:

- Устаревшие пакеты
- Нет возможности генерировать тесты на многие функции
- Нелогичное внутреннее устройство инструмента
- Требовательность к настройке окружения

SPOILER: этого было недостаточно

Механизация скриптами

```
grep --recursive --include='**_test.go' --files-with-matches 'func Fuzz' . |
            while read file
            do
                grep -oP '(?<=func )Fuzz\w+' "$file" |
                while read func
                do
                    echo "Fuzzing $func in $file" >> "$path_to_logfile"
                    parentDir=$(dirname $file)
                    go test "$parentDir" -run='^'$func'$' -fuzz='^'$func'$' -
fuzztime=${OPTARG}s >> "$path_to_logfile"
                    rc=$?
                    echo $rc >> "$path_to_RCfile"
                    echo "...finished with rc=$rc" >> "$path_to_logfile"
                done
            done
```

Libfuzzer

Плюсы

- Стандартизация
- Генетический алгоритм с обратной связью
- Удобство работы
- Высокое покрытие

Минусы

- Неочевидная интеграция
- Проблемы со сбором покрытия

Возможность интеграции libfuzzer

Libfuzzer является общепринятым стандартом фаззинга

Эффективно используется практически в любом языке

Сложность интеграции зависит от совмещения C и Golang

Сравнение путей

	Go-fuzz	Go fuzz	Libfuzzer
Качество подхода и покрытия	Высокое	Среднее	Высокое
Непрерывность процесса	Не поддерживает	Не поддерживает	Поддерживает
Возможность кастомизации	Каждый тест пишется вручную	Генераторы	Вплоть до Structure-aware
Автоматизация и удобство	Требует сложной настройки	Bash Magic	Простой запуск бинарника

Мы можем сделать лучше!

Необходимо реализовать инструмент, который будет:

- Удобен для использования
- Использовать продвинутый алгоритм и генерировать качественный корпус
- Самостоятельно создавать все необходимые промежуточные сущности
- Работать без участия инженера
- Собирать удобные и понятные отчеты

SPOILER: это все может Libfuzzer!

Go-build & Go-118-fuzz-build

Использует CGO для интеграции Libfuzzer

Позволяет через go build и флаги собрать статическую библиотеку на C

Фаззинг запускается из бинарного файла под Libfuzzer

По полученному корпусу можно собрать покрытие

Пара слов о генераторах

Можно написать любой кастомный

Генератор организован с помощью шаблонов

Генератор тестов на функции GRPC API из Proto

Генератор моков для фаззинга сервисов

Как быть с покрытием?

Golang не поддерживает метки покрытия от Libfuzzer

Собирается с помощью Go tool cover

Использует сгенерированные тесты

Использует полученный корпус

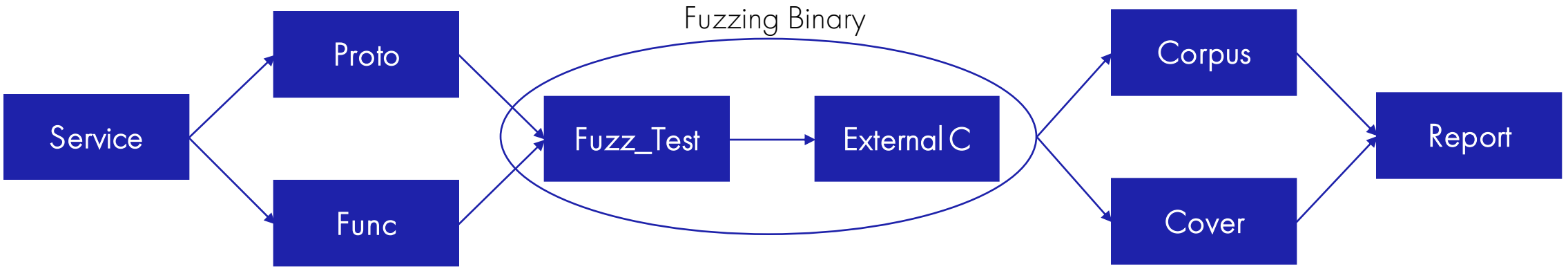
Про артефакты

Сертифицируется код, а не созданные в процессе сборки артефакты

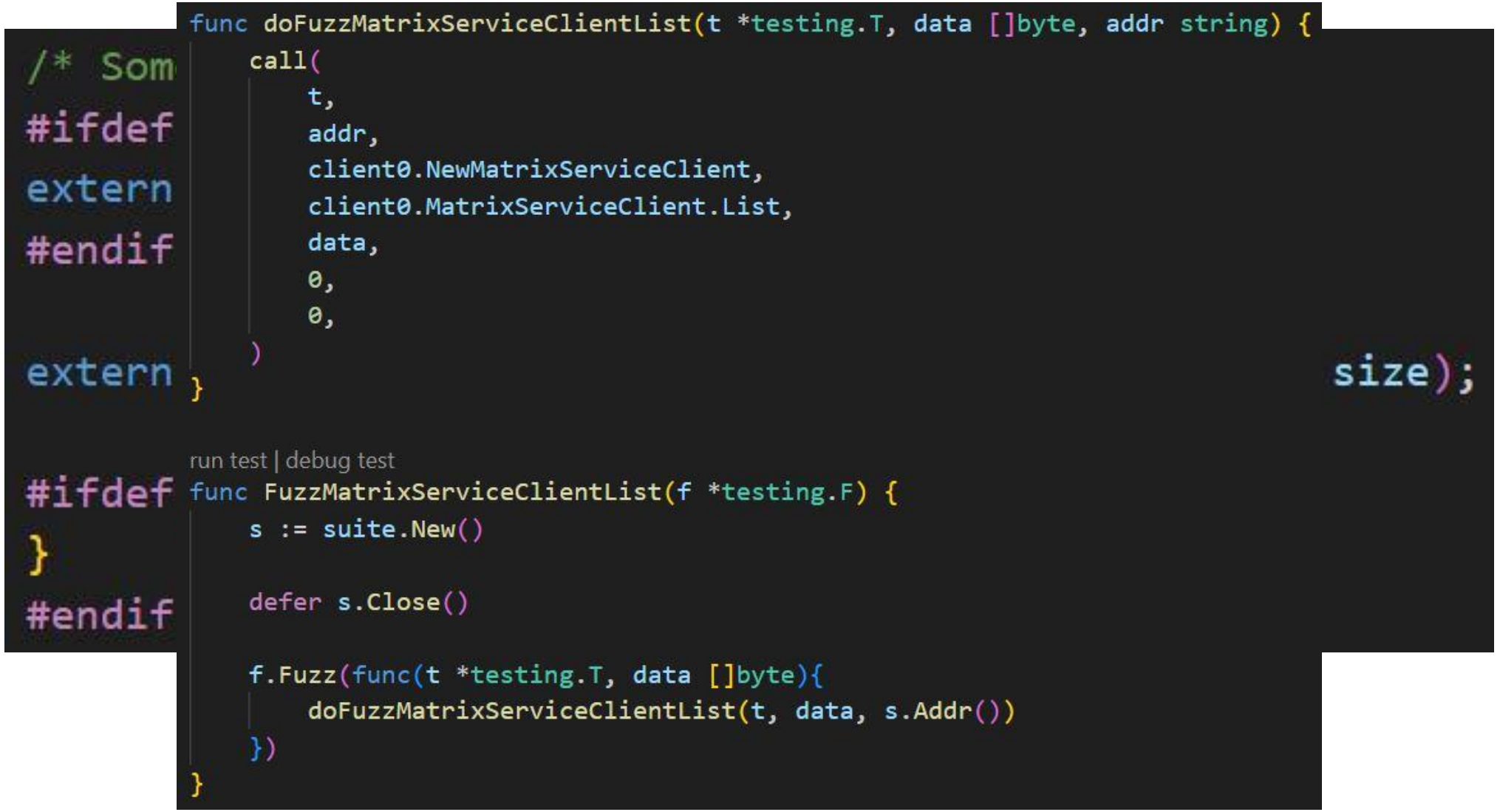
Мы можем воспроизводить зависания/падения

Покрытие фаззинг-тестами ≠ покрытие обычными тестами

Схема инструмента



Про точки входа



```
/* Som
#ifdef
extern
#endif


extern


#ifdef
}
#endif
```

```
func doFuzzMatrixServiceClientList(t *testing.T, data []byte, addr string) {
    call(
        t,
        addr,
        client0.NewMatrixServiceClient,
        client0.MatrixServiceClient.List,
        data,
        0,
        0,
    )
}
```

```
size);
```

```
run test | debug test
func FuzzMatrixServiceClientList(f *testing.F) {
    s := suite.New()

    defer s.Close()

    f.Fuzz(func(t *testing.T, data []byte){
        doFuzzMatrixServiceClientList(t, data, s.Addr())
    })
}
```

Демонстрируем работу

```go
func ReadMatrixFromFile(fsys afero.Fs, filePath string) ([]Record, error) {
	file, err := fsys.Open(filePath)
	if err != nil {
		return nil, err
	}

	scanner := bufio.NewScanner(file)
	scanner.Split(bufio.ScanLines)
	var matrixSlice []Record

	for scanner.Scan() {
		matrix, err := ParseMatrix(scanner.Text())
		if err != nil {
			file.Close()
			return nil, err
		}
		matrixSlice = append(matrixSlice, *matrix)
	}

	err = file.Close()
	if err != nil {
		return nil, err
	}

	return matrixSlice, nil
}
```

```
yadro.dev/cyp/cyp-rbac/v2/matrix/matrix.go (77.1%)
yadro.dev/cyp/cyp-rbac/v2/metrics/registry.go (33.3%)
yadro.dev/cyp/cyp-rbac/v2/objects/object.go (51.2%)
yadro.dev/cyp/cyp-rbac/v2/operations/operation.go (49.0%)
yadro.dev/cyp/cyp-rbac/v2/rbac/contract/contract.go (0.0%)
yadro.dev/cyp/cyp-rbac/v2/rbac/errors.go (0.0%)
yadro.dev/cyp/cyp-rbac/v2/rbac/file.go (84.1%)
yadro.dev/cyp/cyp-rbac/v2/rbac/rbac.go (70.2%)
yadro.dev/cyp/cyp-rbac/v2/roles/role.go (89.7%)
yadro.dev/cyp/cyp-rbac/v2/utils/resolver.go (0.0%)
yadro.dev/cyp/cyp-rbac/v2/utils/utils.go (61.5%)
Average cover: 51.759459 percents
```

К чему мы пришли?

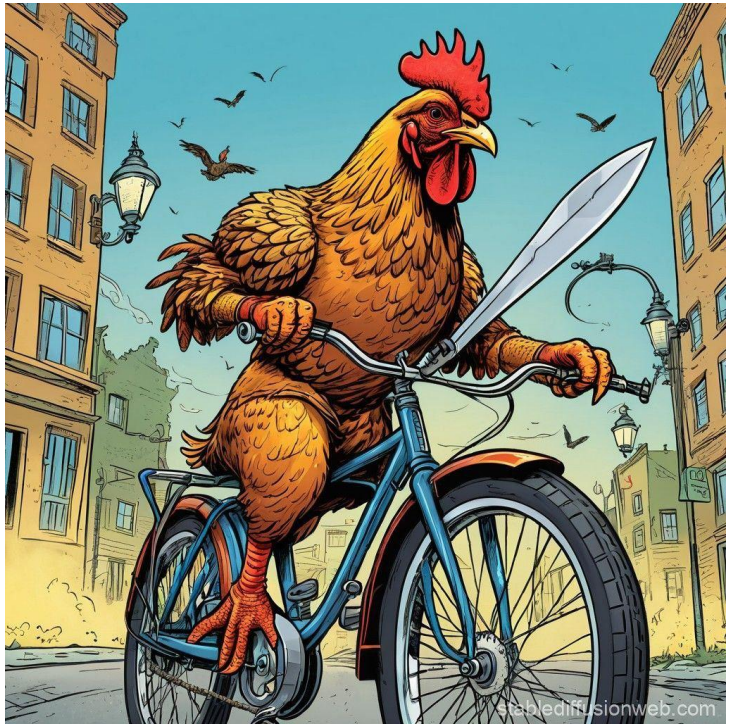
Можем генерировать все необходимые сущности для фаззинга

Можем манипулровать корпусом для повышения покрытия

Инструмент функционирует самостоятельно и требует минимальной настройки

Можем получить подробный и понятный отчет

Фаззинг-тесты можно использовать как unit-тесты




YADRO