Увеличение кодового покрытия ПО систем ПАЗ с помощью «Киннор Динамика»

Дужак Евгений

Руководитель группы разработки СЗИ

О чем эта презентация?

1. Проблемы и ограничения традиционного тестирования
2. Что такое фаззинг?
3. Виды фаззинг-тестирования
4. Пять причин внедрить фаззинг-тестирование
5. Как мы внедряли фаззинг?
6. Будущее Киннор Динамика.

Анекдот

Как-то раз решили провести эксперимент по выживанию и закрыли в пустых комнатах Инженера, Физика и Математика. Дали им по банке тушенки, а открывашки не дали.
Через некоторое время открыли и видят, что:
1. Инженер сидит довольный перед пустой, но сильно раскуроченной банкой. Его спрашивают - «Что ты сделал?». Он отвечает: «Доработал банку до точки отказа»
2. Физик сидит тоже довольный перед аккуратно вскрытой банкой. Его ответ на аналогичный вопрос был таким: «Рассчитал вектор сил и, надавив на банку в определенном месте, вскрыл ее»
3. Математик же сидел перед банкой и бормотал: "Предположим, что банка открыта, предположим, что банка открыта..."

Проблемы и ограничения традиционного тестирования

1. Концентрация на «функциональной» корректности – тесты пишутся либо очевидно правильно, либо очевидно неправильно
2. «Заблуждение разработчика» – иногда (часто) тесты пишут сами разработчики.
3. Нет исследований «крайних» или необычных ситуаций
4. Лабораторные условия – изоляция от реальности
5. Чем сложнее объект исследования, тем сложнее учесть все возможные варианты использования (см. цикломатическая сложность)

Что может помочь преодолеть эти проблемы и ограничения?

Ответ: фаззинг-тестирование

Ремарка №1. Цикломатическая сложность

Цикломатическая сложность - количество линейно независимых путей через программный код. Абсолютно линейный код (например, типичный HelloWorld) - имеет сложность равную единицы, поскольку путь один. Условие = ветвление, а это +1, а то и +2 к сложности кода.

Высокая цикломатическая сложность - прямой путь к добавлению в поверхность атаки, это то что касается ИБ, а если говорить про функциональную безопасность, то большинство сертификационных органов считает, что чем выше цикломатическая сложность, тем сложнее покрыть код модульными тестами. Рекомендации, основанные на стандартах кодирования и ГОСТ, советуют ограничиваться **значением 10** (MISRA/ГОСТ Р МЭК 61508)

Что такое фаззинг?

Фаззинг-тестирование – метод автоматического/автоматизированного динамического анализа ПО, при котором объект исследования подвергается воздействию большого количества случайных, искаженных и нестандартных входных данных с целью доказательства стойкости объекта исследования.

Ограничения «классического» фаззинг-тестирования

1. Не проверяет логическую корректность
2. Не чувствителен к семантике
3. Проблема "застревания"

Плохо работает с: криптографическими проверками, хеш-суммами, магическими числами

4. Требует подготовки и постобработки:

Классификация падения (предотвращения дублирования багов)
Минимизация входов (например, afl-tmin)
Анализ стека вызовов

Фаззинг не является "панацеей"

Виды фаззинг-тестирования

1. Случайный фаззинг
   Яркие представители: radamsa, zzuf
2. Мутационный фаззинг
   Яркие представители:
   Киннор Динамика
   afl/afl++/libfuzzer
3. Символьное исполнение
   Яркие представители: klee/angr
4. Конколическое исполнение
   Яркие представители: driller/Qsym
5. Направленный фаззинг
   Яркие представители: aflgo

KINNOR DYNAMICS

KLEE

Виды фаззинг-тестирования

Эвристические	Аналитические
Случайный	Символьное исполнение
Мутационный	Конколическое исполнение
	Направленный фаззинг

5 причин внедрить фаззинг-тестирование
Причина №1

Автоматическое обнаружение уязвимостей

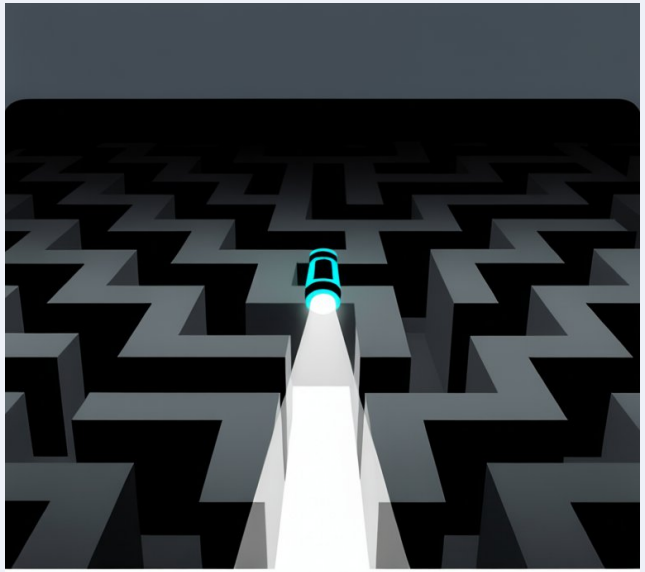
Фаззинг-тестирование эффективно выявляет ошибки, которые можно эксплуатировать:

- **Переполнение буферов**
- **Использование памяти после освобождения**
- **Утечки памяти**
- **Неопределенное поведение**
- **Зацикливаний и зависаний**

5 причин внедрить фаззинг-тестирование
Причина №2

Увеличение кодового покрытия без ручного написания тестов

Если используется coverage-guided фаззинг, то фаззер успешно увеличивает покрытие путем мутаций, завязанных на покрытие. А если использовать гибридный подход, то высока вероятность приблизиться к 100% покрытия. Особенно если использовать деление объекта исследования на модули и их покрытие фаззинг-тестами.



5 причин внедрить фаззинг-тестирование
Причина №3

Повышает безопасность из-за явного shift-left

Фаззинг является частью верификационных испытаний, указанных в большинстве современных отраслевых стандартов:

1. ГОСТ Р 56939-2024
2. ГОСТ Р МЭК 61508
3. DO-178C
4. IEC 62304

Всё это актуально для встраиваемых и safety-critical систем

ГОСТ Р

SIL
IEC 61508

IEC
IEC 62304
Medical device software

DO-178
Qualification

5 причин внедрить фаззинг-тестирование Причина №4

Фаззинг автоматизирован и масштабируем

1. **Требует наладки, дальше работает сам**
2. **Генерирует входные данные без участия человека**
3. **Crash-тесты сохраняются автоматически**
4. **Легко интегрируется в CI/CD**
5. **Масштабируется в зависимости от доступных ресурсов**
6. **Непрерывное тестирование**

5 причин внедрить фаззинг-тестирование
Причина №5

Фаззинг экономит время и деньги

1. Чем раньше обнаружен баг, тем дешевле его устранить
2. Настройка относительно быстрая
3. Снижение нагрузки на тестировщиков и ИБ-специалистов
4. Формирует культуру качества

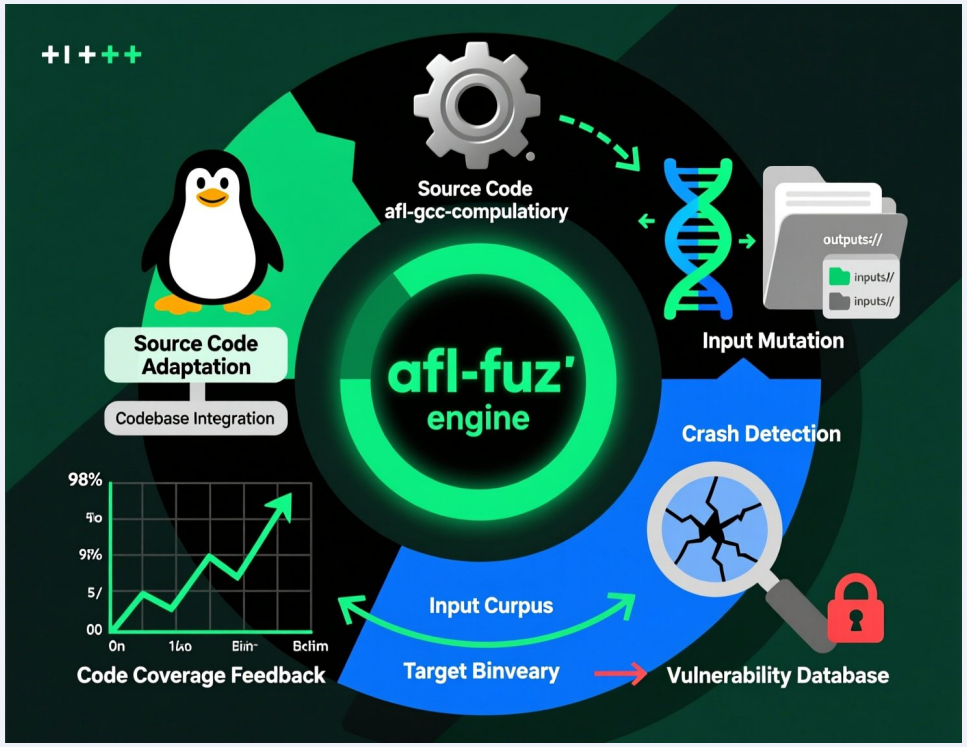
Как мы внедряли фаззинг? Кросс-методика

Методика

1. Адаптируем исходники ОИ из ПА для сборки в linux
2. Собираем с помощью компилятора afl-gcc для linux
3. Подготавливаем входные данные
4. Проводим фаззинг-тестирование.

Проблемы:

1. Наведенные и пропущенные (разница сред)
2. Трудозатратно (1 пункт)



Как мы внедряли фаззинг? Нативная методика

1. Берем оригинальные исходники ОИ из ПА
2. Собираем с помощью компилятора afl-gcc-fast для Нейтрино (совместим с КР)
3. Подготавливаем входные данные
4. Проводим фаззинг-тестирование.

Избавились от среды и трудозатратности

А можно лучше?

Как мы внедряли фаззинг? Киннор Динамика

Можно! Внедряем в CI/CD

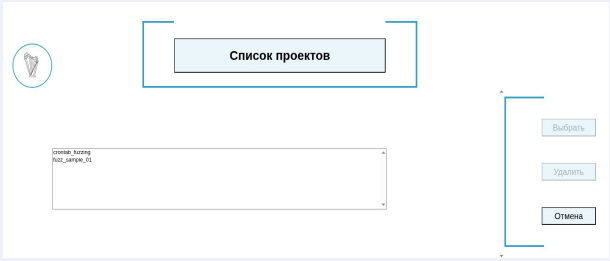
Важный момент!

Фаззинг не встроить в традиционный пайплайн – процесс потенциально бесконечный.

Решение: параллельный фаззинг-конвейер с Киннор Динамика

1. **Не просто фаззинг, полноценный фаззинг-сервис**
2. **Дружит с CI/CD (разделен на веб-интерфейс и бэкэнд)**

KINNOR DYNAMICS



Будущее. Компиляторы

1) Интеграция с clang
   - реализация на базе clang – afl-clang-fast (альтернатива afl-gcc-fast)
2) Формирование «плотной» интеграции с КР для ЗОСРВ «Нейтрино»
3) Выстраивание «альтернативного» цикла разработки с упором на аналитические инструменты.

Будущее. Гибридный фаззинг

Для чего внедрять в Киннор Динамику гибридный фаззинг?

1) **Это красиво**
   - на самом деле, если уровень доверия (по ФСТЭК) выше 4 УД, то он вам необходим
2) **Это повышает качество вашего кода**
   - фаззинг-тестирование становится более «направленным»

Будущее. Гибридный фаззинг

Варианты opensource-движков символьного исполнения для интеграции с Киннор Динамика:
- klee (требуется байткод clang)
- angr (требуется написание собственного модуля для python)

KINNOR DYNAMICS

+

или

СВД
Встраиваемые
Системы

Спасибо за внимание!

Дужак Евгений Дмитриевич

Руководитель группы разработки СЗИ

ул. Кузнецовская, д. 19, г. Санкт-Петербург,
8-812-317-55-56
www.kpda.ru
e.duzhak@kpda.ru