

Динамический анализ. Пара слов про фаззинг.

Дужак Евгений

Руководитель группы разработки СЗИ



«Вид динамического анализа, в котором на вход объекту исследования подаётся некорректный ввод в надежде получить корректную реакцию объекта исследования.»

Преимущество: могут работать почти без участия человека (автоматически/автоматизировано).

Фаззинг. Цели. Теория.

Цели:

- 1) упростить работу отдела ИБ/верификации/тестирования, высвободив ресурсы на более интеллектуальные задачи.**
- 2) Увеличить кодовое покрытие**

Задача: перебрать наибольшее количество граничных, случайных, семантически правильных значений.



Фаззинг. Цели. Практика.

Практика:

- 1) нужны машинные ресурсы (обычно очень весомые)
- 2) Нужны входные данные
- 3) Нужна подготовка объектов для исследования (в том числе модификация кода)
- 4) Не дружит с классическим тестированием
- 5) Высокие временные затраты (чем сложнее ОИ – тем дольше фаззинг)

Ожидания

Реальность



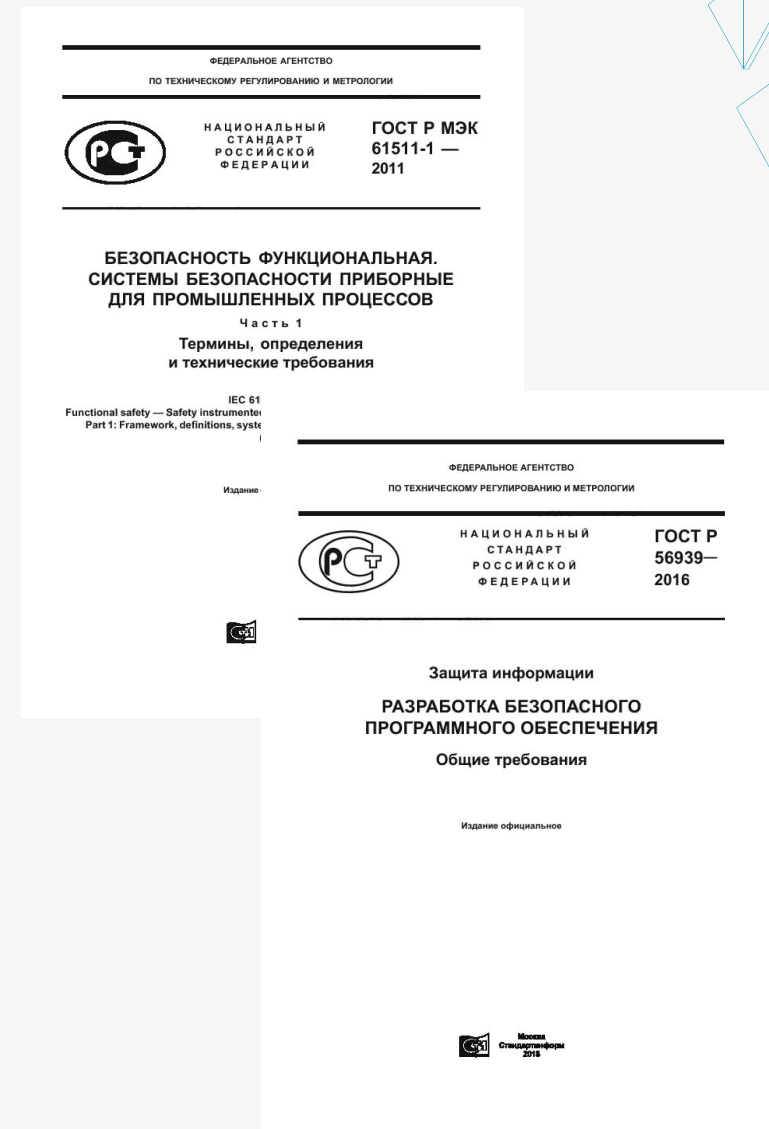
Фаззинг. ИБ vs ФБ

ИБ: успешное прохождение фаззинг-тестирования = доверительная стойкость перед потенциальным злоумышленником.

ФБ: успешное фаззинг-тестирование = 100%

Почему так много?

ФБ не любит функции с высокой цикломатической сложностью.

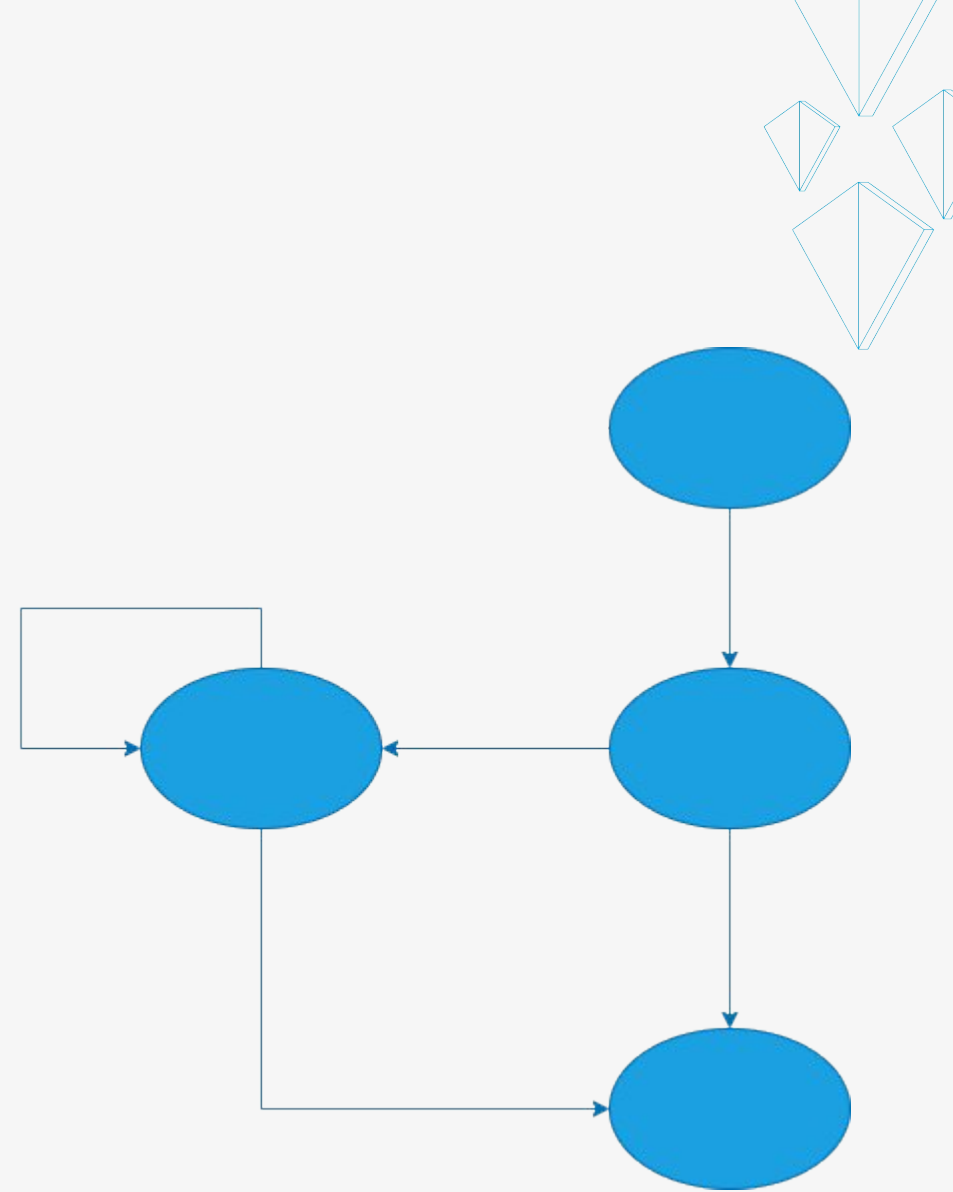


Цикломатическая сложность

Простым языком: количество возможных вариантов выполнения кода (функция без ветвлений имеет $ЦС = 1$, один `if` = $ЦС + 1$ и тд)

Для ФБ нежелательно использовать функции с $ЦС$ выше 10.

Для ИБ нет таких ограничений, но высокая $ЦС$ – повод включить компонент в поверхность атаки.



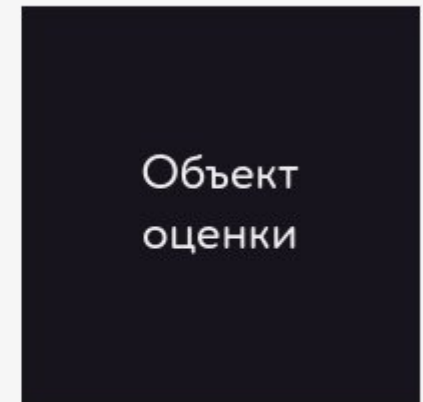
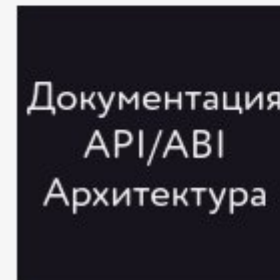
Поверхность атаки.

Совокупность компонентов ОИ со следующими признаками:

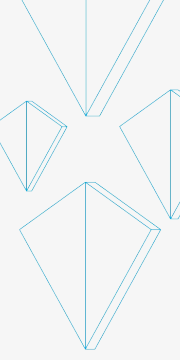
- 1) Важность с точки зрения их функционала (например, функции, реализующие СЗИ)
- 2) Цикломатическая сложность
- 3) Потенциал уязвимости
- 4) Входы и выходы ОИ (а также дополнительные функции «транспорта» прямые или косвенные)



Аналитик



Классификация фаззинг-тестирования



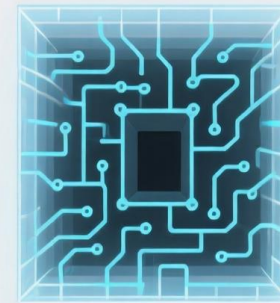
По виду обратной связи:

- 1) Рандомный – нет никакой обратной связи между ОИ и генератором входных данных (пример radamsa)
- 2) Мутационный – используются генетические алгоритмы и обратная связь по покрытию (пример afl++ и его bitmap)
- 3) Гибридный – мутационный+, иногда называется генерационным, в виду явной обратной связи с помощью генерации дополнительных входных данных (символьное, конколическое и конкретное исполнение)

Классификация фаззинг-тестирования

По виду доступа к исходным кодам ОИ

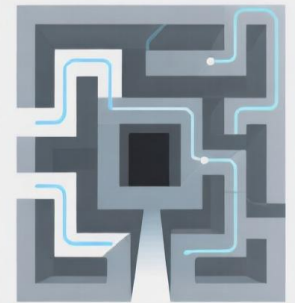
- 1) Черный ящик – нет доступа ко внутреннему устройству ОИ
- 2) Белый ящик – полный доступ ко внутреннему устройству, обычно сопряжен с пересборкой ОИ и его статической инструментацией.
- 3) Серый ящик – гибрид белого и черного (не всегда требует полного знания об ОИ и его пересборки, можно частично знать структуры данных и семантику)



White-box



Black-box



Grey-box

Классификация фаззинг-тестирования

По виду инструментации

- 1) Статическая (компиляторная) – внедрение неких инструкций в ОИ на этапе компиляции

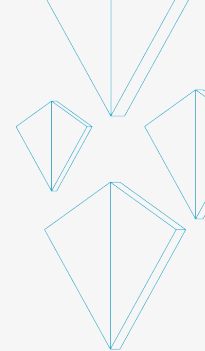
Примеры: afl-clang-fast/afl-gcc-fast/libfuzzer

- 2) Динамическая (эмуляторная) – внедрение инструкций на этапе выполнения ОИ

Примеры: intel PT, DynamoRIO, QEMU-TCG, valgrind, QEMU-usermode



Фаззинг-тестирование и инструментация



Зачем нужна инструментация?

Для увеличения эффективности фаззинг-тестирования – по скорости выполнения (статическая) и качеству (статическая/динамическая).

Можно использовать и без, но тогда у фаззинга пропадет обратная связь.

Лучшее сочетание: мутационный фаззинг со статической инструментацией + источник дополнительных данных в виде символьного/конколического/конкретного движка, эмулирующего и обсчитывающего наилучшие входные данные для глубокого внедрения в ОИ

Что может быть целью фаззинг-тестирования?

Кратко:

любая программа компилируемая или интерпретируемая

Подробно:

Функция (точнее ее модульные тесты)

Модификация модульных тестов (afl persistent mode)

Прикладная программа

Любой интерфейсы (сокеты, файл, порты)

Операционная система (syzkaller, kAFL)

Прошивки (afl++ unicorn mode)

Инструменты?

Opensource:

afl (нынче afl++)

syzkaller

libfuzzer (часть LLVM)

QSym

Angr

Driller

Отечественные:

Киннор Динамика (НИЦ ФБ)

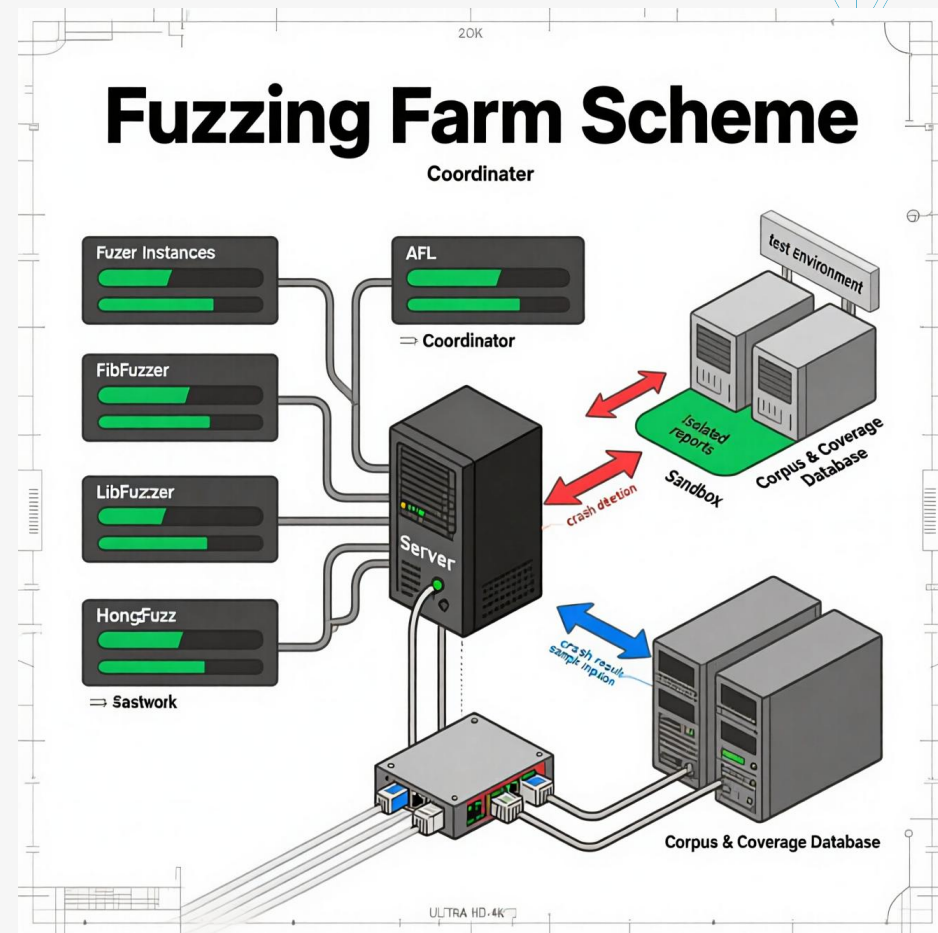
Crusher/Sydr (ИСП РАН)

Как внедрять?

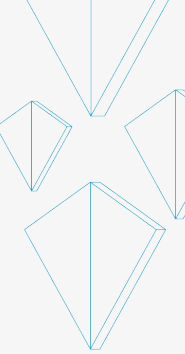
Совет дня: организуйте фаззинг-ферму

Два пути:

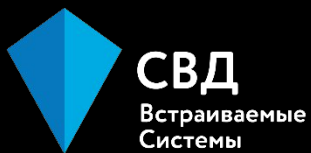
- 1) Хороший(-ие) инженер(-ы) (свои или найти) + много ресурсов (временных и машинных) + R&D на базе opensource
- 2) Найти продукт, который уже прошел этот наукоёмкий путь



Типовой сценарий фаззинг-тестирования



- 1) Есть продукт. Есть архитектура.
- 2) Делай модель угроз
- 3) Выделяй ПА продукта
- 4) Выделяй из ПА компонент
- 5) Выделяй ПА компонента
- 6) Выбирай метод фаззинга (классический, persistent mode, qemu mode)
- 7) Подготовь входные данные (можно с допобработкой, можно без)
- 8) Запускай фаззинг
- 9) Обработывай результаты и заводи баги, если для этого есть повод.
- 10) Возвращайся к пункту 4 и так пока не закончатся компоненты



Спасибо за внимание!

Дужак Евгений

Руководитель группы разработки СЗИ

+7 (812) 317-55-56

www.kpda.ru