Fuzzing: the state of the art

Степанов Д.С.

Анализ программ

- Статический анализ
  - Model checking
  - Data flow analysis
  - Code inspections
  - ~~Fuzzing~~
- Динамический анализ
  - Debugging
  - Functional testing
  - Fuzzing
- Гибридные методы
  - Fuzzing
  - ...

Что такое fuzzing?

“Program that generates a stream of random characters to be consumed by a target program”, Barton Miller et al., 1988

Преимущества и недостатки

Плюсы:

- Легко развернуть
- Работает 24/7 (человек 5/8)
- Генерирует редкие кейсы
- Высокая точность
- Не нужны знания о целевой программе

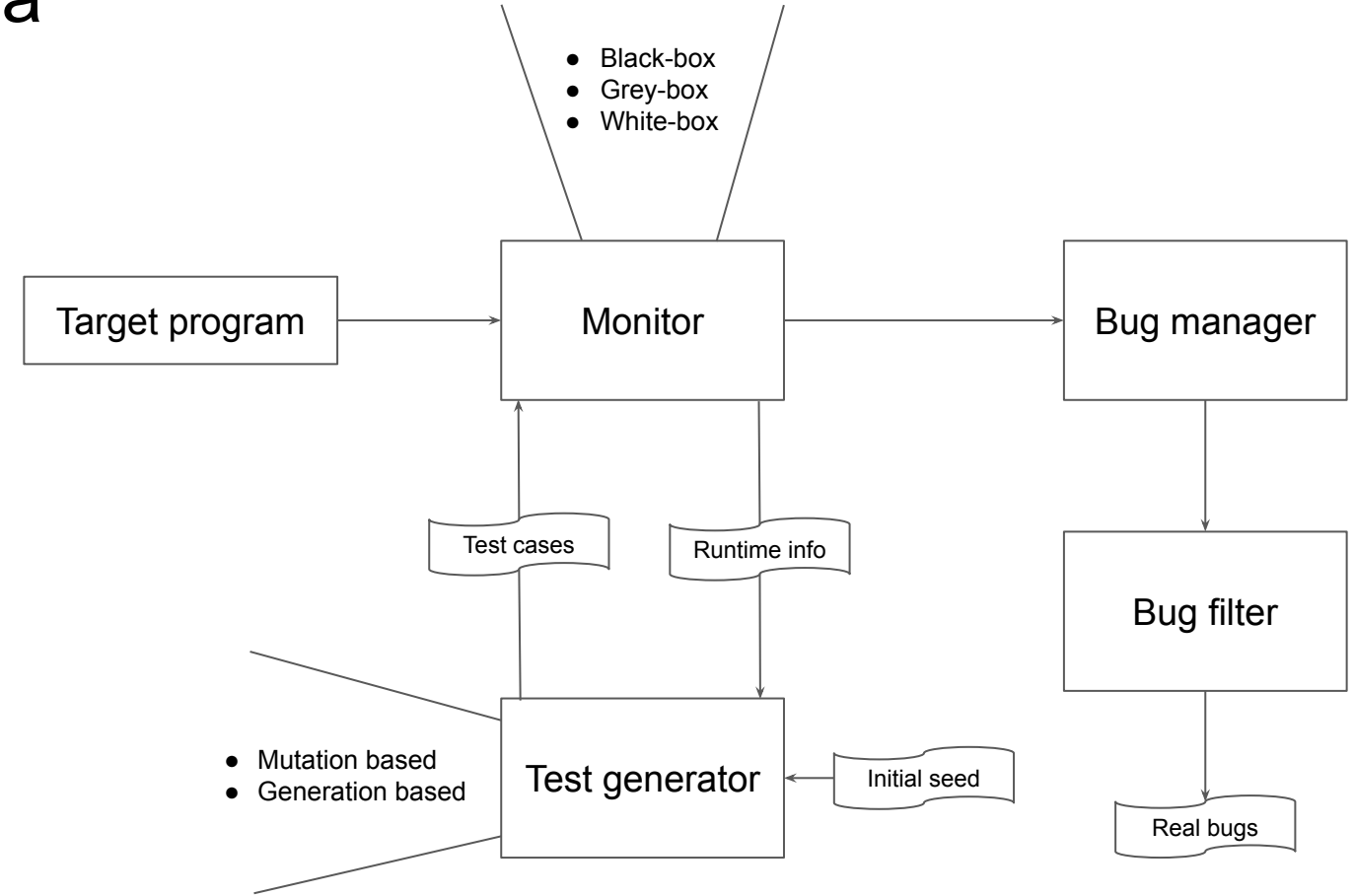
Минусы:

- Низкая эффективность
- Низкая полнота (для некоторых видов фаззеров)

А где же он применяется?

Краткий ответ: везде

Схема


Black-box
Grey-box
White-box
Target program
Monitor
Bug manager
Test cases
Runtime info
Bug filter
Mutation based
Generation based
Test generator
Initial seed
Real bugs

Taxonomy

Black-box:

- Работает только с I/O программы

White-box:

- Предложен Годфруа в 2007
- Dynamic symbolic execution + coverage-maximizing heuristic

Grey-box:

- Основан на использовании некоторых фич из white-box фаззинга

Минусы:

- Полнота

Минусы:

- Сложность реализации
- Плохо применим для больших проектов

Минусы:

- Попытка найти баланс между black-box и white-box методами

Пример black-box фаззинга

```
fun getIthEl(a: Int, i: Int): Int {
  val list = listOf(1, 2, 3)
  return if (a == 1) {
    if (i > 0) {
      list[i]
    } else -1
  } else -1
}
```



```
getIthEl(-1, 0)
getIthEl(1, 2)
getIthEl(-100, 21545135)
getIthEl(125123590, 5123553)
//… several years later
getIthEl(1, 123)
```

Пример grey-box фаззинга

```
fun getIthEl(a: Int, i: Int): Int {
  val list = listOf(1, 2, 3)
  return if (a == 1) {
    if (i > 0) {
      list[i]
    } else -1
  } else -1
}
```



```
getIthEl(-1, 0)
getIthEl(-1, -2142135)
getIthEl(-125, 352)
getIthEl(1, 0)
getIthEl(1, 1)
getIthEl(1, 123)
```

Пример white-box фаззинга

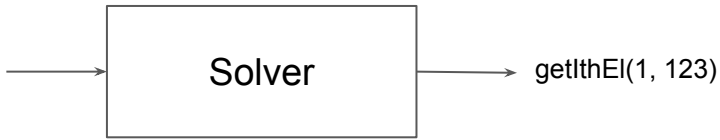
```
fun getIthEl(a: Int, i: Int): Int {
    val list = listOf(1, 2, 3)
    return if (a == 1) {
        if (i > 0) {
            list[i]
        } else -1
    } else -1
}
```

```
  @S %0 = new java/lang/Integer[3]
  @S %1 = java/lang/Integer.class.valueOf(1)
  @S *(%0[0]) = %1
  @S %2 = java/lang/Integer.class.valueOf(2)
  @S *(%0[1]) = %2
  @S %3 = java/lang/Integer.class.valueOf(3)
  @S *(%0[2]) = %3
  @S %4 = kotlin/collections/CollectionsKt.class.listOf(%0)
  @S %5 = arg$0 != 1
) -> (BEGIN
 <OR> (
  @P %5 = false
  @S %6 = arg$1 <= 0
) -> (
  @P %6 = false
  @S %7 = %4.get(arg$1)
  @S %8 = (%7 as java/lang/Number)
  @S %9 = %8.intValue()
  @S %10 = %9
),  <OR> (
  @P %5 = false
  @S %6 = arg$1 <= 0
) -> (
  @P %6 = true
  @S %10 = -1
),  <OR> (
  @P %5 = true
  @S %10 = -1
) END) -> (
  @S <retval> = %10
```

```
[DEBUG] - Args: [2, 3]
[DEBUG] - Collected trace: (
  @S %0.1 = new java/lang/Integer[3]
  @S %1.2 = java/lang/Integer.class.valueOf(1)
  @S *(%0.1[0]) = %1.2
  @S %2.3 = java/lang/Integer.class.valueOf(2)
  @S *(%0.1[1]) = %2.3
  @S %3.4 = java/lang/Integer.class.valueOf(3)
  @S *(%0.1[2]) = %3.4
  @S %4.5 = kotlin/collections/CollectionsKt.class.listOf(%0.1)
  @S %5.6 = arg$0 != 1
  @P %5.6 = true
  @S %10.7 = -1
  @S true = true
)
```

Пример white-box фаззинга

```
[DEBUG] - Args: [2, 3]
[DEBUG] - Collected trace: (
  @S %0.1 = new java/lang/Integer[3]
  @S %1.2 = java/lang/Integer.class.valueOf(1)
  @S *(%0.1[0]) = %1.2
  @S %2.3 = java/lang/Integer.class.valueOf(2)
  @S *(%0.1[1]) = %2.3
  @S %3.4 = java/lang/Integer.class.valueOf(3)
  @S *(%0.1[2]) = %3.4
  @S %4.5 = kotlin/collections/CollectionsKt.class.listOf(%0.1)
  @S %5.6 = arg$0 != 1
  @P %5.6 = true
  @S %10.7 = -1
  @S true = true
)
```



Важные вопросы

- Начальная выборка
- Генерация тестов
- Обработка runtime информации
- Постпроцессинг результатов
- Масштабируемость

Начальная выборка

- Случайное множество
- Hotset алгоритм
- ...
- Минимальное множество с наибольшим покрытием (AFL)

Dumb vs Smart fuzzers

- Мутационный подход
- Нет/почти нет информации о формате ввода
- Мутируются имеющиеся тестовые данные
- Мутации могут быть рандомными или использовать некоторые эвристики

- Генеративный подход
- Тестовые данные генерируются из описания формата (документация, грамматика и т. д.)

Dumb fuzzers mutations

- Bit-flipping
- Arithmetic mutations
- Block-based mutations

AFL

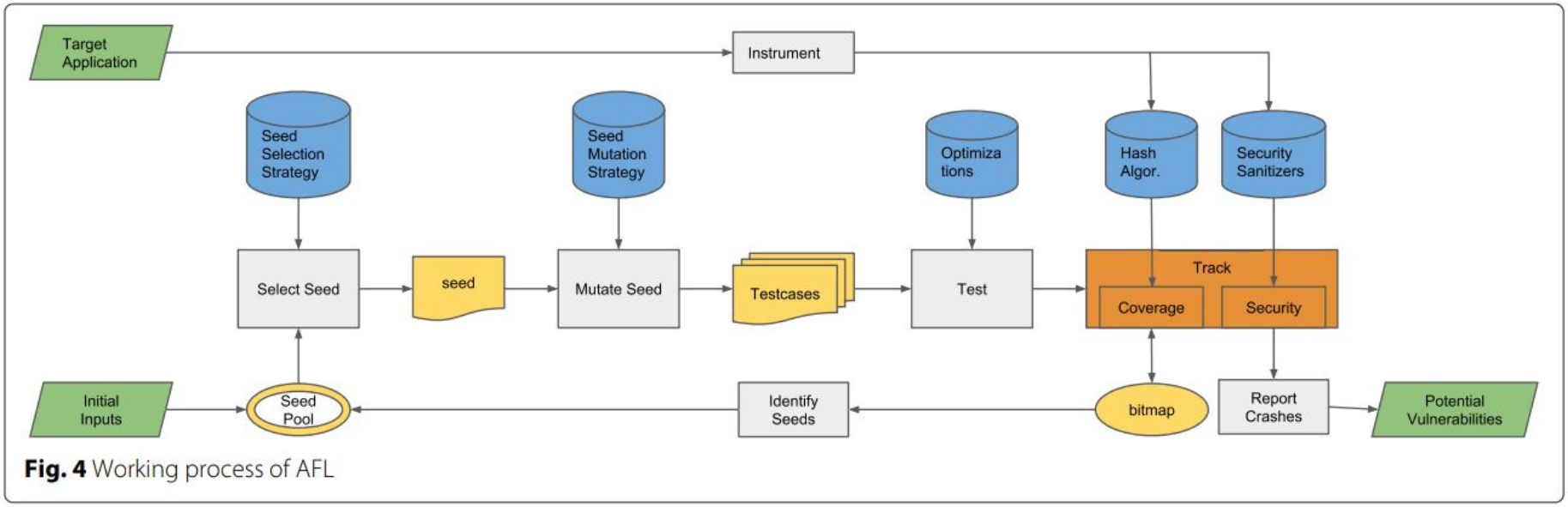


Fig. 4 Working process of AFL

AFL mutations

- Bit-flipping

```
fun Char.flip(): Char {
  val i = Random.nextInt(1, 6)
  val resInBytes = this.toByte() xor i.toByte()
  return resInBytes.toChar()
}

fun main() {
  val a = "KSPT"
  val randomInd = Random.nextInt(0, a.length)
  val randomCh = a[randomInd]
  val newA = a.replace(randomCh, randomCh.flip())
  println(newA)
}
```

Результат:

KRPT
KPPT
KSPU
KSPP
KSPP
KSPV
JSPT
JSPT
KSTT
KPPT

AFL mutations

- Arithmetic mutations

```
fun main() {
  val a = "KSPT"
  val buffer = ByteBuffer.wrap(a.byteInputStream().readAllBytes())
  val smallInteger = Random.nextInt(0, 36)
  val newInt = if (Random.nextBoolean()) buffer.int + smallInteger else buffer.int - smallInteger
  val newBuffer = ByteBuffer.allocate(4)
  newBuffer.putInt(newInt)
  println(newBuffer.array().joinToString("") { "${it.toChar()}" })
}
```

Результат:

KSPo
KSPL
KSPh
KSPT
KSPY
KSPV
KSPP
KSPE
KSP\
KSPT

AFL mutations

- Block-based mutations (вставка/удаление/замена)

```
fun main() {
   val a = "KSPT"
   val buffer = ByteBuffer.wrap(a.byteInputStream().readAllBytes())
   val randomByteSeq = generateRandomByteSeq()
   val randomIndex = Random.nextInt(buffer.limit())
   val newBuffer = ByteBuffer.wrap(buffer.getSubBuf(0..randomIndex)
         + randomByteSeq
         + buffer.getSubBuf(randomIndex..buffer.limit()))

    println(newBuffer.array().toList().joinToString("") { "${it.toChar()}" })
}
```

Результат:

jKSPT
K│□aSPT
KSPV← ¦ T
SP
KS
ㅌ┤ㅋ
KS{PT
K
Kㅅ㎭SPT
ㅜKSPT
u□jKSPT
KSP+T

Coverage-based dumb fuzzing

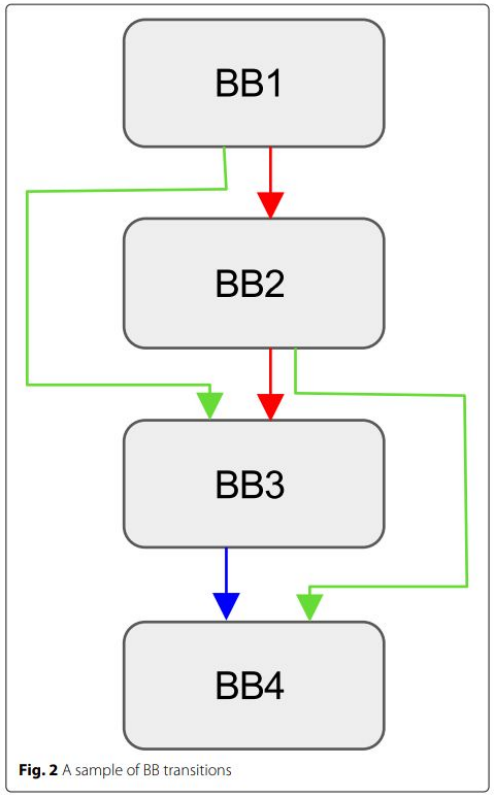


Fig. 2 A sample of BB transitions

Testcase selection:

- AFLFast
- Vuzzer
- AFLGo

AFL

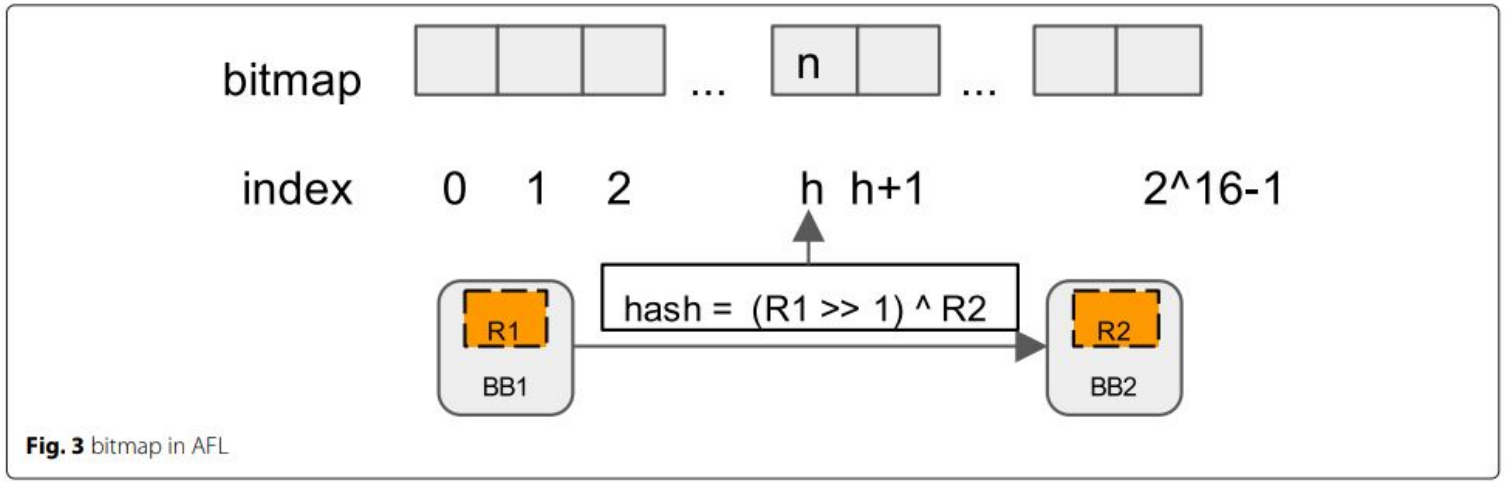
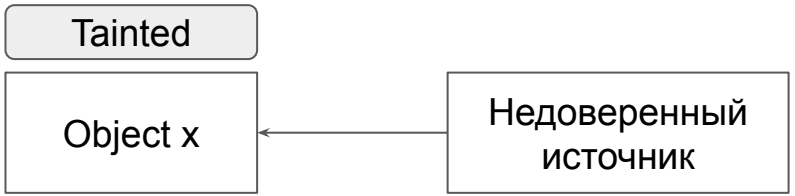


Fig. 3 bitmap in AFL

Taint analysis-based dumb fuzzing



Примеры недоверенных источников данных:

- Файлы (mp3, pdf, …)
- Протоколы (UDP, HTTP)
- USB
- Веб-камеры
- ...

Taint analysis-based dumb fuzzing

- Анализ помеченных данных используется для отслеживания влияния байтов на поток управления, чтобы выявить “ключевые”.
- Бывает статическим и динамическим

Генеративные фаззеры

- Предопределенный формат тестовых данных
  - В Peach, PROTOS, Dharma спецификация задается пользователем
  - Tavor работает со БНФ спецификацией
  - Nautilus - грамматика
  - Заточенные под какую-то задачу (jsfunfuzz, TLS-Attacker)
- Выведенный формат тестовых данных
  - Skyfire выводит вероятностную контекстно-чувствительную грамматику по набору исходных тестов
  - IMF (фаззер API ядра) выводит модель по логам
  - Learn&Fuzz использует машинное обучение

Что насчёт white-box фаззеров?

- Также нужна “правильная” начальная тестовая выборка
- Имеет ряд проблем:
  - Path explosion
  - Imprecise symbolic execution
  - Восстановление сложных данных из результатов SMT-решателя
  - Memory modelling
  - …

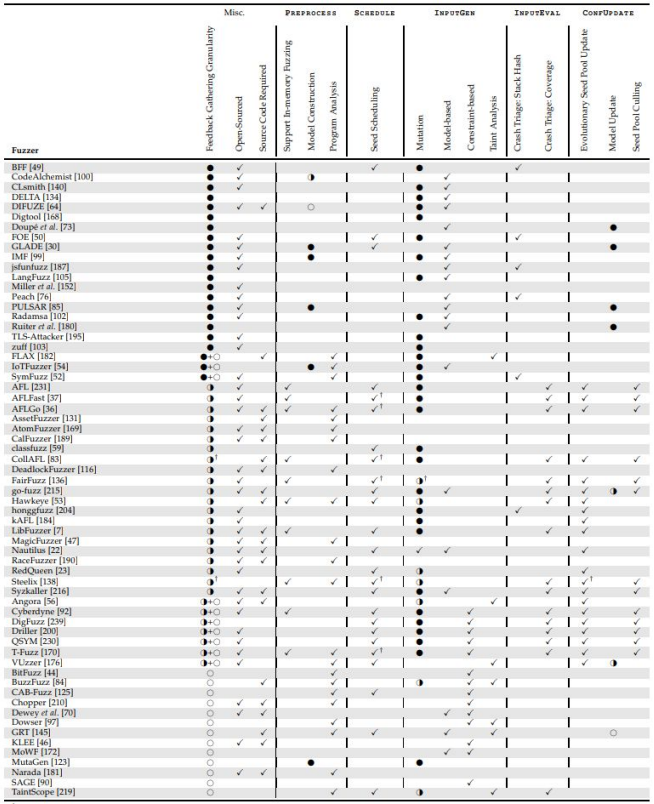
Что лучше всего на практике?

- Для разных задач и возможностей - разные подходы
- Иногда достаточно и black-box фаззера
- Наилучший показатель время/качество/возможность анализа показывает гибрид между grey- и white-box подходами (применяем white-box, чтобы направить grey-box туда, куда он изначально не мог добраться)

Обработка результатов

- Deduplication
  - Stack Backtrace Hashing
  - Coverage-based Deduplication (AFL)
  - Semantics-aware Deduplication
- Prioritization
  - Fuzzer taming problem
  - Taint analysis
- Test case minimization
  - AFL пытается удалить байты или превратить их в 0
  - Delta-debugging
  - CReduce
- Bug isolation
  - Slicing
  - Spectrum-based
  - Dynamic methods

Разнообразие фаззеров



Fuzzer	Misc.			Preprocess		Schedule	InputGen				InputEval		ConfUpdate		
	Feedback Gathering Granularity	Open-Sourced	Source Code Required	Support In-memory Fuzzing	Model Construction	Program Analysis	Seed Scheduling	Mutation	Model-based	Constraint-based	Taint Analysis	Crash Triage: Stack Hash	Crash Triage: Coverage	Evolutionary Seed Pool Update	Model Update	Seed Pool Culling
BFF [49]	●	✓					✓	●				✓				
CodeAlchemist [100]	●	✓			◗				✓							
CLsmith [140]	●	✓						●	✓							
DELTA [134]	●							●	✓							
DIFUZE [64]	●	✓	✓		○			●	✓							
Digtool [168]	●							●								
Doupé *et al.* [73]	●								✓						●	
FOE [50]	●	✓					✓	●				✓				
GLADE [30]	●	✓			●		✓		✓						●	
IMF [99]	●	✓			●			●	✓							
jsfunfuzz [187]	●	✓							✓			✓				
LangFuzz [105]	●							●	✓							
Miller *et al.* [152]	●	✓														
Peach [76]	●	✓							✓			✓				
PULSAR [85]	●	✓			●				✓						●	
Radamsa [102]	●	✓						●	✓							
Ruiter *et al.* [180]	●								✓						●	
TLS-Attacker [195]	●	✓						●								
zuff [103]	●	✓						●								
FLAX [182]	●+○		✓			✓		●			✓					
IoTFuzzer [54]	●+○				●	✓		●	✓							
SymFuzz [52]	●+○	✓				✓		●				✓				
AFL [231]	◗	✓		✓			✓	●					✓	✓		✓
AFLFast [37]	◗	✓		✓			✓!	●					✓	✓		✓
AFLGo [36]	◗	✓	✓	✓		✓	✓!	●					✓	✓		✓
AssetFuzzer [131]	◗		✓			✓										
AtomFuzzer [169]	◗	✓	✓			✓										
CalFuzzer [189]	◗	✓	✓			✓										
classfuzz [59]	◗						✓	●								
CollAFL [83]	◗!		✓	✓			✓!	●					✓	✓		✓
DeadlockFuzzer [116]	◗	✓	✓			✓										
FairFuzz [136]	◗	✓		✓			✓!	◗!					✓	✓		✓
go-fuzz [215]	◗	✓	✓				✓	●	✓				✓	✓	◗	✓
Hawkeye [53]	◗		✓	✓		✓	✓	◗					✓	✓		
honggfuzz [204]	◗	✓						●				✓		✓		
kAFL [184]	◗	✓						●						✓		
LibFuzzer [7]	◗	✓	✓	✓			✓	●					✓	✓		
MagicFuzzer [47]	◗	✓	✓			✓										
Nautilus [22]	◗	✓	✓				✓		✓	✓				✓		
RaceFuzzer [190]	◗	✓	✓			✓										
RedQueen [23]	◗	✓					✓	◗						✓		
Steelix [138]	◗!			✓		✓	✓!	◗					✓	✓!		✓
Syzkaller [216]	◗	✓	✓				✓	●	✓				✓	✓		✓
Angora [56]	◗+○	✓	✓								✓			✓		
Cyberdyne [92]	◗+○	✓		✓			✓	●		✓			✓	✓		✓
DigFuzz [239]	◗+○						✓	●		✓			✓	✓		✓
Driller [200]	◗+○	✓					✓	●		✓			✓	✓		✓
QSYM [230]	◗+○	✓					✓	●		✓			✓	✓		✓
T-Fuzz [170]	◗+○	✓		✓		✓	✓!	●		✓			✓	✓		✓
VUzzer [176]	◗+○	✓				✓	✓				✓			✓	◗	
BitFuzz [44]	○					✓				✓						
BuzzFuzz [84]	○		✓			✓		◗		✓	✓					
CAB-Fuzz [125]	○					✓	✓			✓						
Chopper [210]	○	✓	✓			✓				✓						
Dewey *et al.* [70]	○	✓	✓						✓	✓						
Dowser [97]	○					✓				✓	✓					
GRT [145]	○		✓			✓	✓		✓		✓				○	
KLEE [46]	○	✓	✓						✓	✓						
MoWF [172]	○								✓	✓						
MutaGen [123]	○				●			●								
Narada [181]	○	✓	✓			✓										
SAGE [90]	○									✓						
TaintScope [219]	○					✓	✓	◗			✓		✓			

Заключение

- Фаззинг - один из самых популярных методов автоматического тестирования программ
- По хорошему, фаззить необходимо все программы
- Самыми популярными фаззерами на данный момент являются
  - AFL - grey-box dumb fuzzing
  - Sage - white-box fuzzing
  - Microsoft Security Risk Detection - porfolio
- Перспективные направления исследований:
  - Symbolic execution
  - Input grammars generation
  - Distributed applications