Program Analysis
Fuzzing

Yue Duan

Outline

- Fuzzing recap
- Research paper:
  - Coverage-based Greybox Fuzzing as Markov Chain
  - Directed Greybox Fuzzing

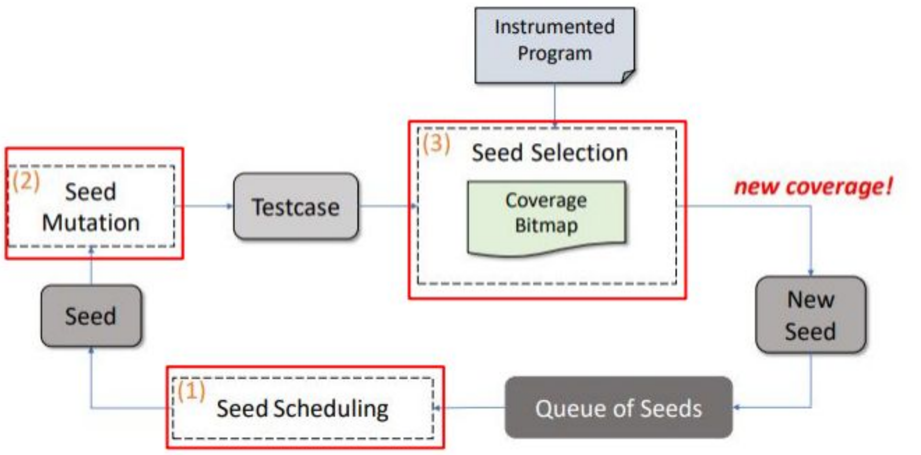
Fuzzing Recap

- Fuzz testing
  - an automated testing technique that uncovers software errors by executing the target program with large number of randomly generated test inputs
  - Three main approaches
    - black-box fuzzing
      - pure random fuzzing
    - grey-box fuzzing
      - leverage some knowledge during testing for guidance
    - white-box fuzzing
      - full knowledge about the target program is needed

Fuzzing Recap

- American Fuzzy Lop (AFL) https://github.com/google/AFL
  - coverage-based greybox fuzzer
  - code instrumentation:
    - collect runtime code coverage info
    - can be done at source code and binary level
    - source code: compile with instrumented code
    - binary: QEMU user mode for instrumentation

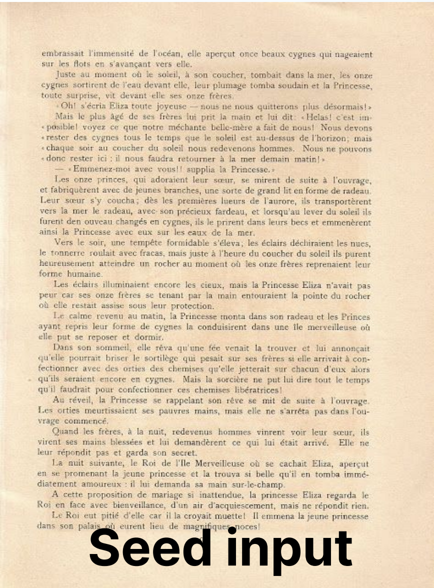
Fuzzing Recap



- Seed scheduling
  - pick the next seed for testing from a set of seed inputs
  - essentially random

Fuzzing Recap

- Seed Mutation
  - bitflips
  - boundary values (0, -1, 1, INT_MAX, etc)
  - simple arithmetics (add/subtract 1)
  - block deletion
  - block insertion



embrassait l'immensité de l'océan, elle aperçut once beaux cygnes qui nageaient sur les flots en s'avançant vers elle.

Juste au moment où le soleil, à son coucher, tombait dans la mer, les onze cygnes sortirent de l'eau devant elle, leur plumage tomba soudain et la Princesse, toute surprise, vit devant elle ses onze frères.

«Oh! s'écria Eliza toute joyeuse — nous ne nous quitterons plus désormais!»

Mais le plus âgé de ses frères lui prit la main et lui dit: «Helas! c'est im-«posible! voyez ce que notre méchante belle-mère a fait de nous! Nous devons «rester des cygnes tous le temps que le soleil est au-dessus de l'horizon; mais «chaque soir au coucher du soleil nous redevenons hommes. Nous ne pouvons «donc rester ici : il nous faudra retourner à la mer demain matin!»

— «Emmenez-moi avec vous!! supplia la Princesse.»

Les onze princes, qui adoraient leur sœur, se mirent de suite à l'ouvrage, et fabriquèrent avec de jeunes branches, une sorte de grand lit en forme de radeau. Leur sœur s'y coucha; dès les premières lueurs de l'aurore, ils transportèrent vers la mer le radeau, avec son précieux fardeau, et lorsqu'au lever du soleil ils furent den ouveau changés en cygnes, ils le prirent dans leurs becs et emmenèrent ainsi la Princesse avec eux sur les eaux de la mer.

Vers le soir, une tempête formidable s'éleva; les éclairs déchiraient les nues, le tonnerre roulait avec fracas, mais juste à l'heure du coucher du soleil ils purent heureusement atteindre un rocher au moment où les onze frères reprenaient leur forme humaine.

Les éclairs illuminaient encore les cieux, mais la Princesse Eliza n'avait pas peur car ses onze frères se tenant par la main entouraient la pointe du rocher où elle restait assise sous leur protection.

Le calme revenu au matin, la Princesse monta dans son radeau et les Princes ayant repris leur forme de cygnes la conduisirent dans une île merveilleuse où elle put se reposer et dormir.

Dans son sommeil, elle rêva qu'une fée venait la trouver et lui annonçait qu'elle pourrait briser le sortilège qui pesait sur ses frères si elle arrivait à confectionner avec des orties des chemises qu'elle jetterait sur chacun d'eux alors qu'ils seraient encore en cygnes. Mais la sorcière ne put lui dire tout le temps qu'il faudrait pour confectionner ces chemises libératrices!

Au réveil, la Princesse se rappelant son rêve se mit de suite à l'ouvrage. Les orties meurtissaient ses pauvres mains, mais elle ne s'arrêta pas dans l'ouvrage commencé.

Quand les frères, à la nuit, redevenus hommes vinrent voir leur sœur, ils virent ses mains blessées et lui demandèrent ce qui lui était arrivé. Elle ne leur répondit pas et garda son secret.

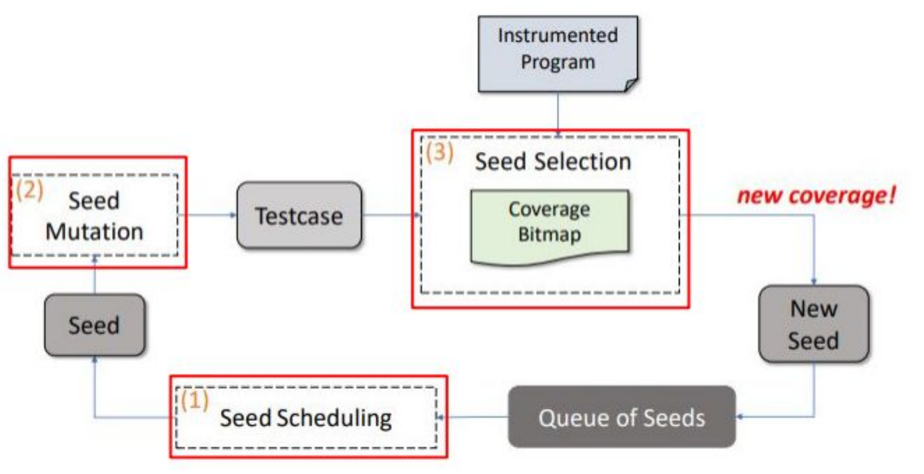
La nuit suivante, le Roi de l'Ile Merveilleuse où se cachait Eliza, aperçut en se promenant la jeune princesse et la trouva si belle qu'il en tomba immédiatement amoureux : il lui demanda sa main sur-le-champ.

A cette proposition de mariage si inattendue, la princesse Eliza regarda le Roi en face avec bienveillance, d'un air d'acquiescement, mais ne répondit rien.

Le Roi eut pitié d'elle car il la croyait muette! Il emmena la jeune princesse dans son palais où eurent lieu de magnifiques noces!

Fuzzing Recap

- Seed Selection
  - coverage metrics used to define ‘interesting’ seeds
  - preserve only the interesting inputs for next round
  - coverage bitmap: new basic block discovery



Fuzzing Recap

- Common limitation for grey-box fuzzing:

```
x = int(input())
if x > 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Let's fuzz it!

1 ⇒ "You lose!"

593 ⇒ "You lose!"

183 ⇒ "You lose!"

4 ⇒ "You lose!"

498 ⇒ "You lose!"

48 ⇒ "You win!"

```
x = int(input())
if x > 10:
    if x^2 == 152399025:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Let's fuzz it!

1 ⇒ "You lose!"

593 ⇒ "You lose!"

183 ⇒ "You lose!"

4 ⇒ "You lose!"

498 ⇒ "You lose!"

42 ⇒ "You lose!"

3 ⇒ "You lose!"

....

57 ⇒ "You lose!"

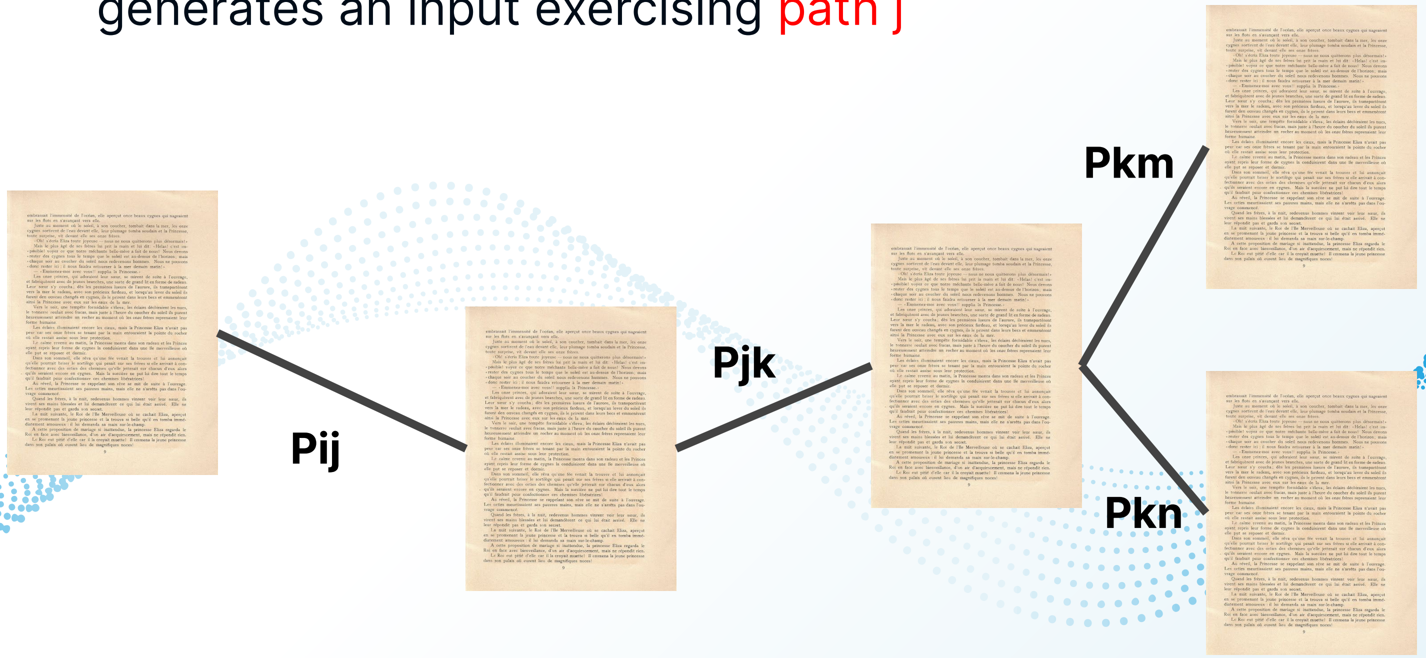
Coverage-based Greybox Fuzzing as Markov Chain

Marcel Böhme , Van Thuan Pham , Abhik Roychoudhury

CCS 2016

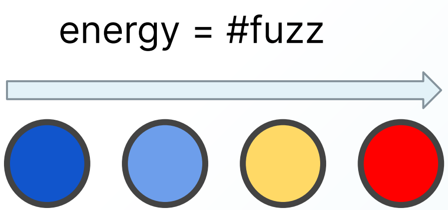
Introduction

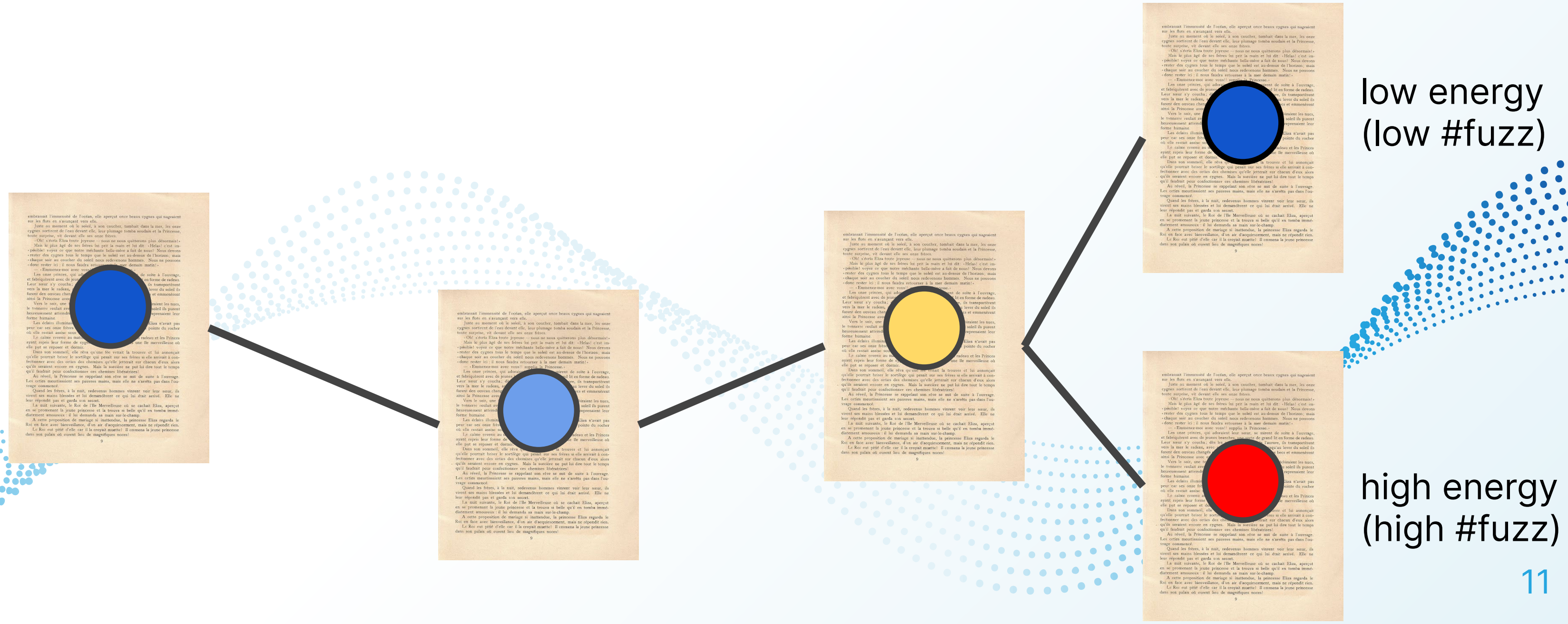
- Markov chain
  - describes the prob Pij that fuzzing the input exercising path i generates an input exercising path j



Introduction

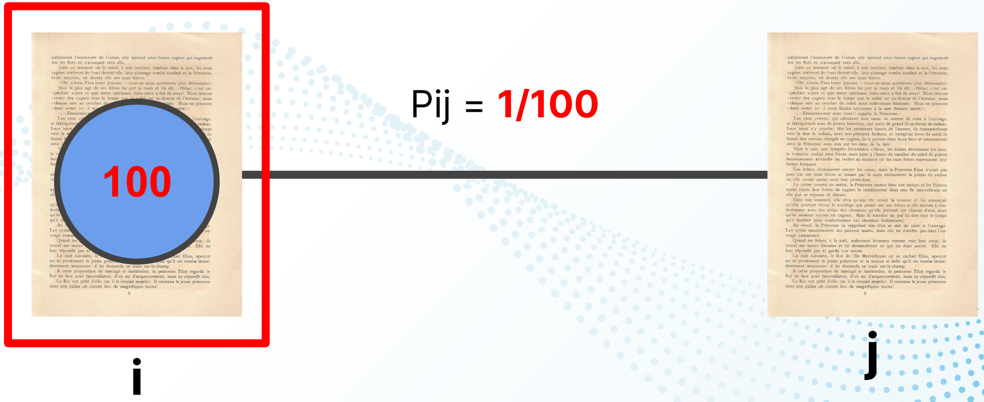
- Add **energy** to each state





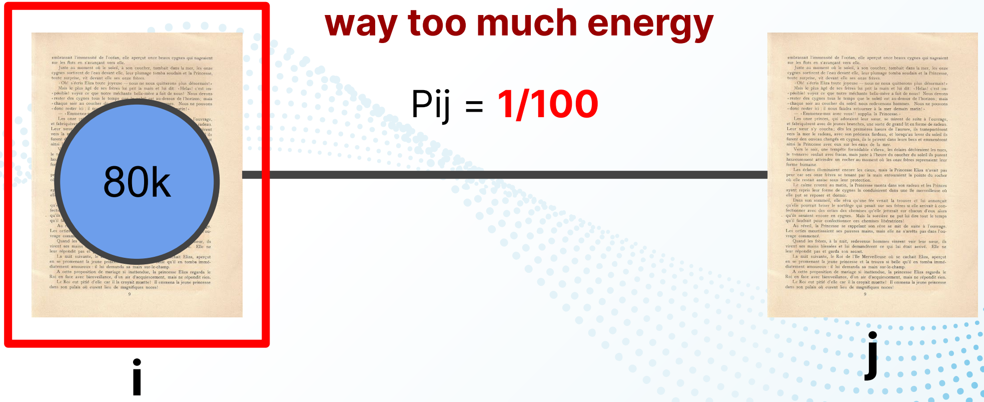
Introduction

How much #fuzz should be generated?
= What is the minimum energy required to expect discovery of new path j from path i?



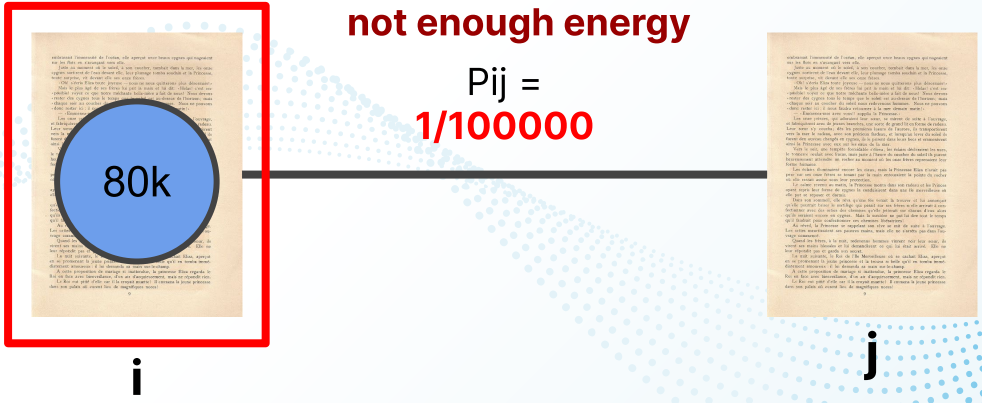
Challenges

- AFL's power schedule is *constant* in the number of times s(i) for the seed has been chosen for fuzzing.



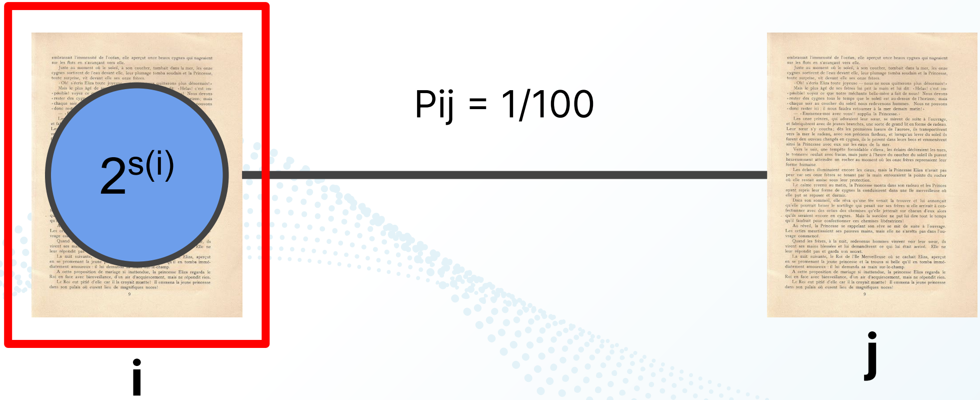
Challenges

- AFL's power schedule is *constant* in the number of times s(i) for the seed has been chosen for fuzzing.



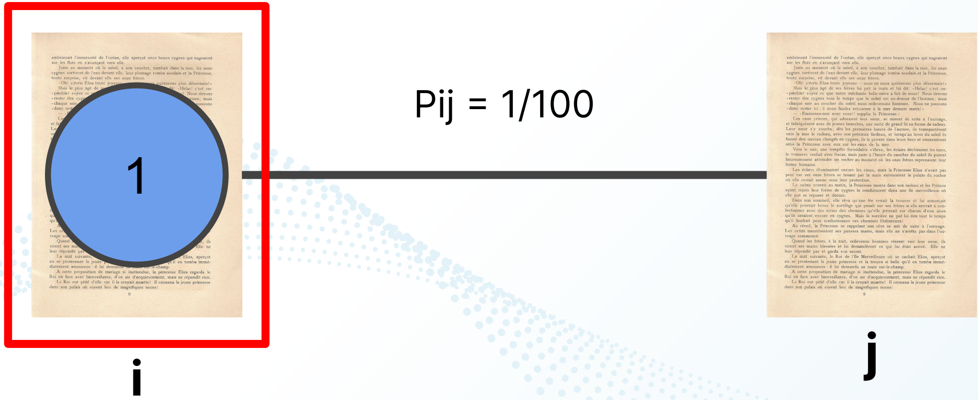
AFLFast

- power schedule is **exponential** in s(i)



AFLFast

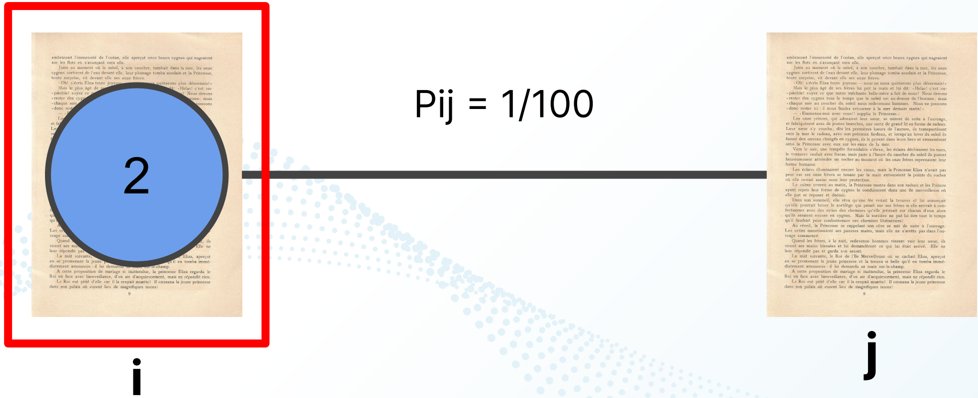
- power schedule is **exponential** in s(i)



Total #fuzz generated: 1

AFLFast

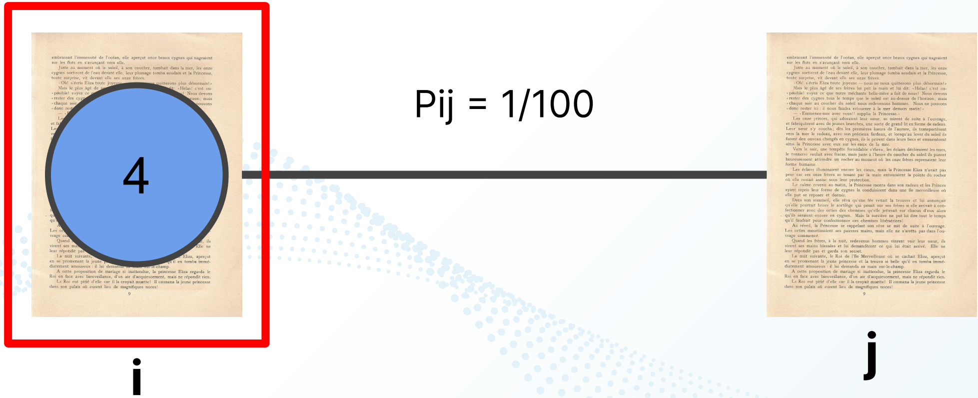
- power schedule is **exponential** in s(i)



Total #fuzz generated: 3

AFLFast

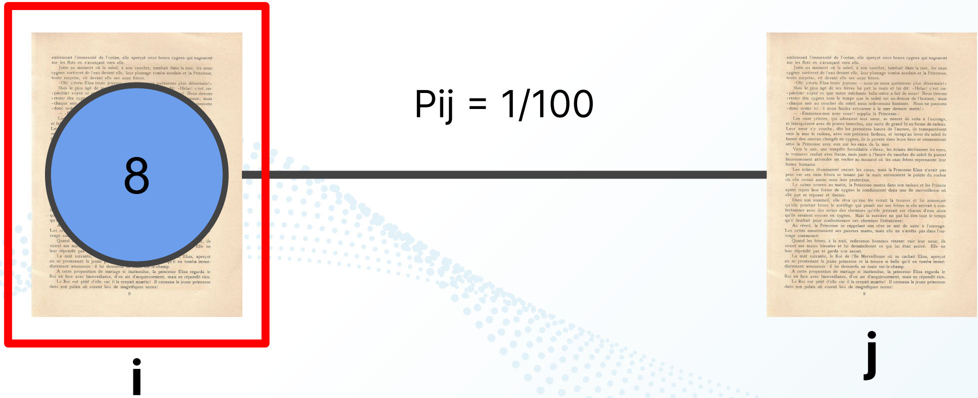
- power schedule is **exponential** in s(i)



Total #fuzz generated: 7

AFLFast

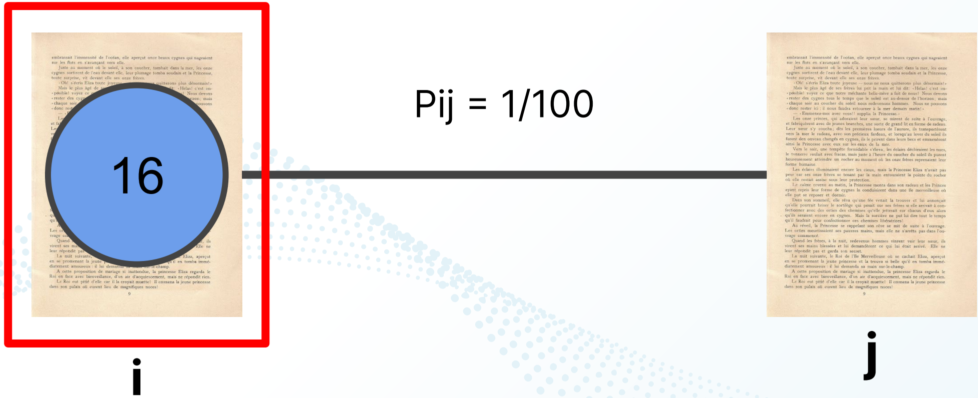
- power schedule is **exponential** in s(i)



Total #fuzz generated: 15

AFLFast

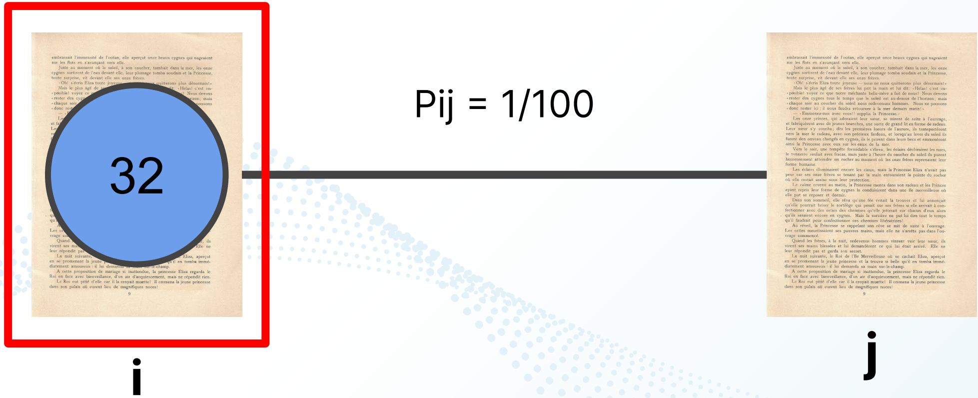
- power schedule is **exponential** in s(i)



Total #fuzz generated: 31

AFLFast

- power schedule is **exponential** in s(i)

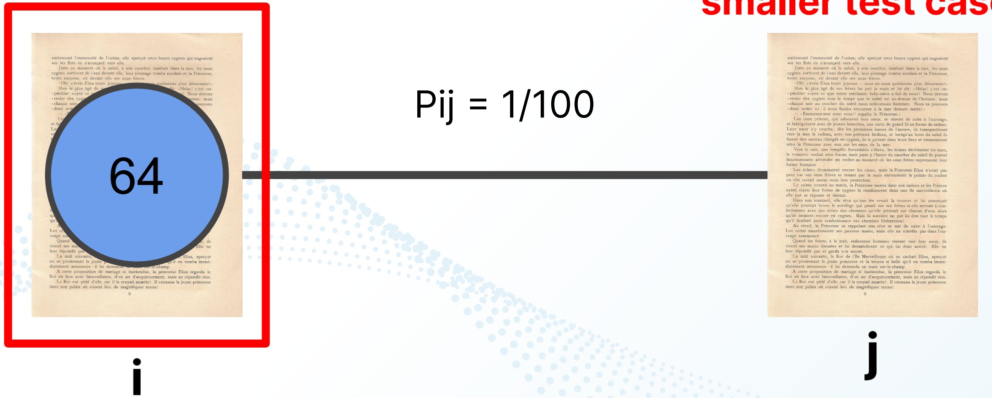


Total #fuzz generated: 63

AFLFast

- power schedule is **exponential** in s(i)

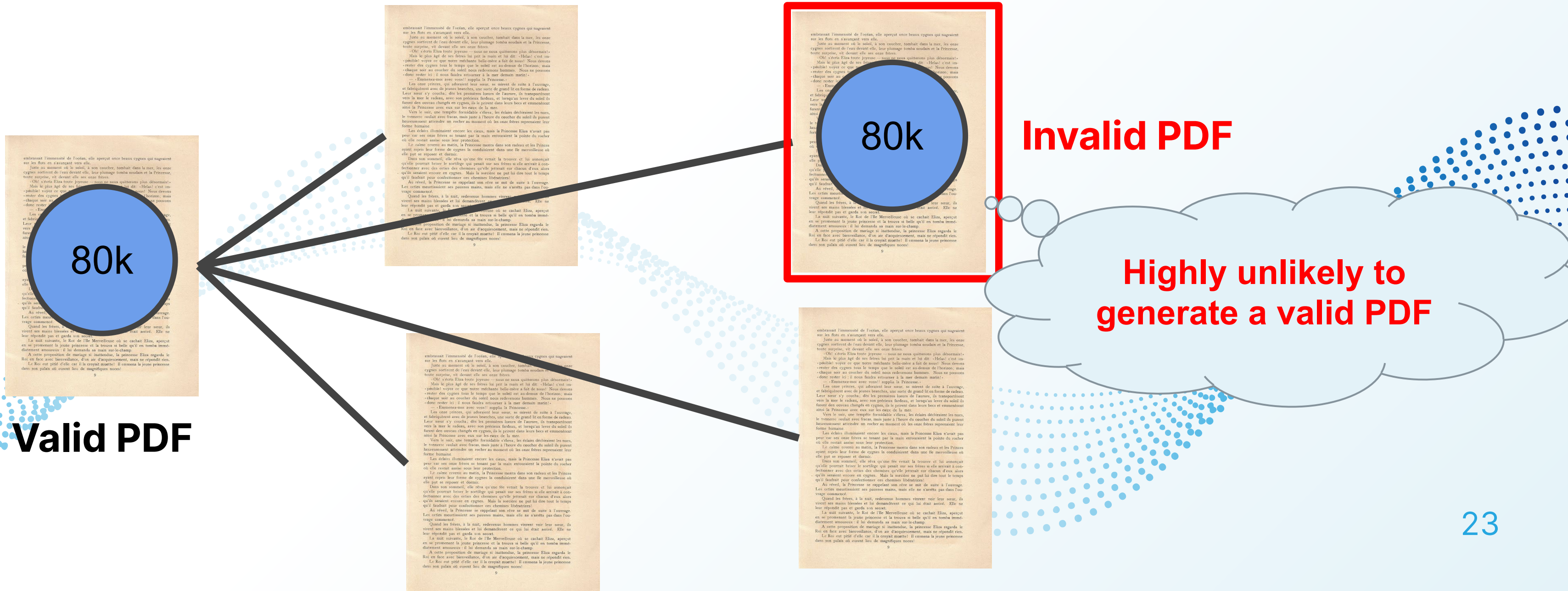
Discovered with much smaller test cases!



Total #fuzz generated: 127

Challenges

- AFL's power schedule is *constant* in the number of times s(i) for the seed has been chosen for fuzzing.
- AFL's power schedule always assigns *high* energy



Challenges

- AFL's power schedule is *constant* in the number of times s(i) the seed has been chosen for fuzzing.
- AFL's power schedule always assigns *high* energy
  - Too much energy assigned to high-frequency paths

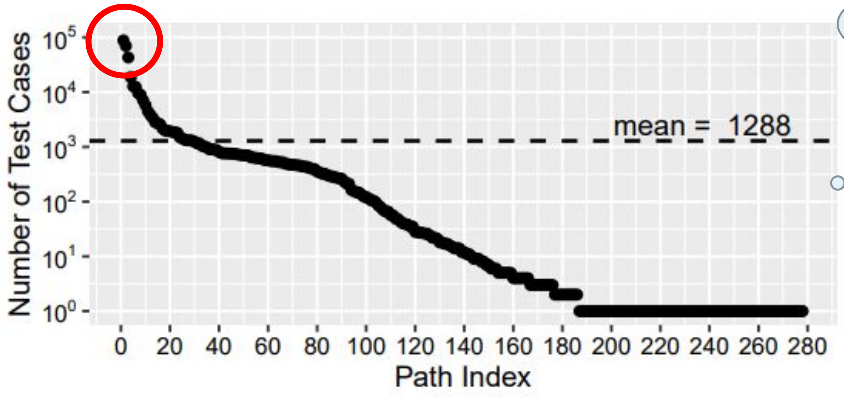


Figure 1: #Fuzz exercising a path (on a log-scale) after running AFL for 10 minutes on the nm-tool.

Most fuzz exercises the same few paths

AFLFast

- AFLFast's power schedule assigns energy that is inversely proportional to the density of the stationary distribution
  - assign low energy to high-frequency paths
  - assign high energy to low-frequency paths
  - approximate the density of distribution for a state i by counting the number of fuzz f(i) that exercises the path i

Power Schedules

- AFL: constant power schedule
  - p(i) = **α**(i)
  - **α**(i) is how AFL judges fuzzing time for path i (~1 min)
- AFLFast:
  - spend *more energy* on low-frequency paths
  - *less energy* on high-frequency paths
  - spend the *minimum energy* required to discover a new state

$$p(i) = \begin{cases} 0 & \text{if } f(i) > \mu \\ \min\left(\frac{\alpha(i)}{\beta} \cdot 2^{s(i)}, M\right) & \text{otherwise.} \end{cases}$$

$$p(i) = \min\left(\frac{\alpha(i)}{\beta} \cdot \frac{2^{s(i)}}{f(i)}, M\right)$$

Search Strategies

- AFL
  - chooses the seeds in the order they are added
  - after the last seed, begin with the first
- AFLFast
  - prioritizes seeds that
    - exercise low-frequency paths
    - have been chosen less often
  - chooses each seed at most once per cycle

Evaluation

- Binutils

Table 1: CVE-IDs and Exploitation Type

Vulnerability	Type
CVE-2016-2226	Exploitable Buffer Overflow
CVE-2016-4487	Invalid Write due to a Use-After-Free
CVE-2016-4488	Invalid Write due to a Use-After-Free
CVE-2016-4489	Invalid Write due to Integer Overflow
CVE-2016-4490	Write Access Violation
CVE-2016-4491	Various Stack Corruptions
CVE-2016-4492	Write Access Violation
CVE-2016-4493	Write Access Violation
CVE-2016-6131	Stack Corruption
Bug 1	Buffer Overflow (Invalid Read)
Bug 2	Buffer Overflow (Invalid Read)
Bug 3	Buffer Overflow (Invalid Read)

Evaluation

- General results

Vulnerability	AFL	*AFL-Fast*	Factor
CVE-2016-2226	> 24.00 h	3.85 h	N/A
CVE-2016-4487	2.63 h	0.46 h	5.8
CVE-2016-4488	6.92 h	0.98 h	7.0
CVE-2016-4489	10.68 h	2.78 h	3.8
CVE-2016-4490	3.68 h	0.41 h	9.1
CVE-2016-4491	> 24.00 h	4.74 h	N/A
CVE-2016-4492	12.18 h	0.87 h	14.1
CVE-2016-4493	4.48 h	1.00 h	4.5
CVE-2016-6131	> 24.00 h	5.48 h	N/A
Bug 1	20.43 h	3.38 h	6.0
Bug 2	20.91 h	2.89 h	7.2
Bug 3	> 24.00 h	5.07 h	N/A

Figure 8: Time to expose the vulnerability.

Evaluation

- Power schedules

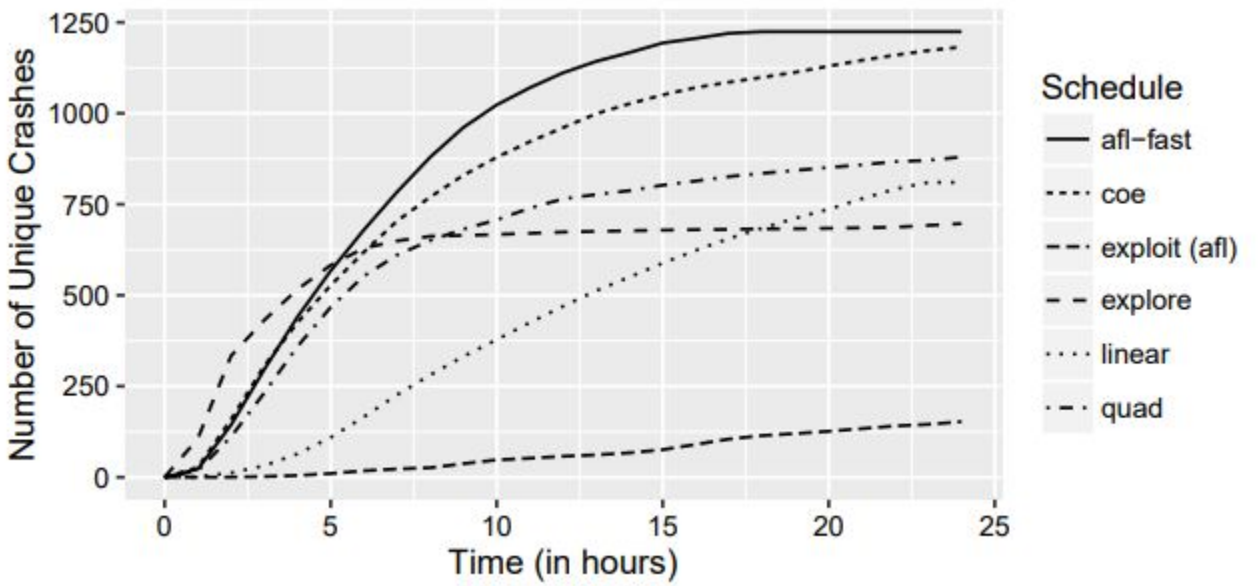


Figure 10: #Crashes over Time (Schedules).

Directed Greybox Fuzzing

Marcel Böhme, Van Thuan Pham, M.D Nguyen, Abhik Roychoudhury

CCS 2017

Motivation

- Grey-box fuzzing is frequently used
  - state-of-the-art in automated vulnerability detection
  - extremely efficient
    - all program analysis before/at instrumentation time
    - start with a seed, choose a seed file, fuzz it
    - add to corpus only if new input increases coverage
- **Cannot be directed!**

Motivation

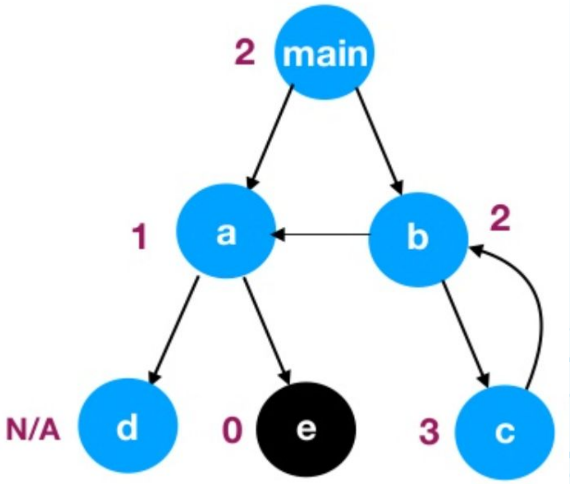
- Directed fuzzing
  - patch testing: reach changed statements
  - crash reproduction: exercise stack traces
  - SA report verification: reach ‘dangerous’ locations
  - etc

Overview of AFLGo

- Directed fuzzing as **optimization problem**
  - instrumentation time:
    - extract call graph and control-flow graphs
    - for each BB, compute distance to target locations
    - instrument program to aggregate distance values
  - Runtime
    - collect coverage and distance information
    - decide **how long to be fuzzed** based on distance
      - if input is closer to the targets, fuzz longer
      - if input is further away, fuzz shorter

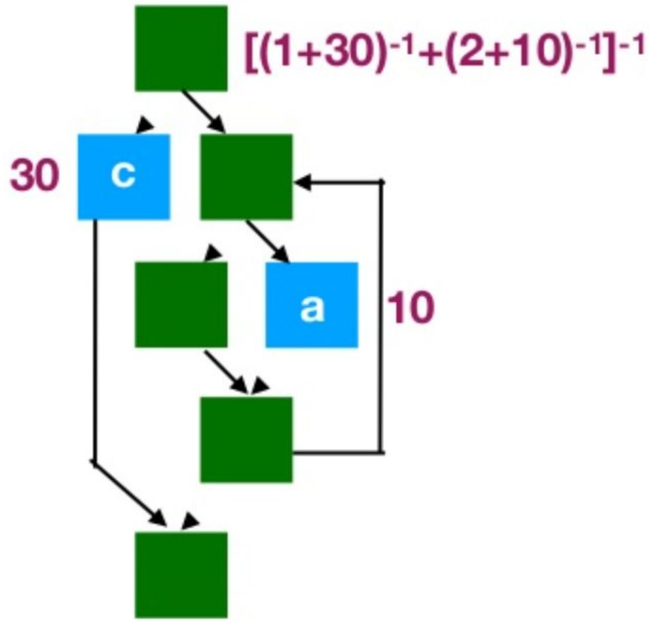
Instrumentation

- Function-level target distance
  - use call graph
  - identify target functions in CG
  - compute the harmonic mean of the lengths of the shortest path to targets



Instrumentation

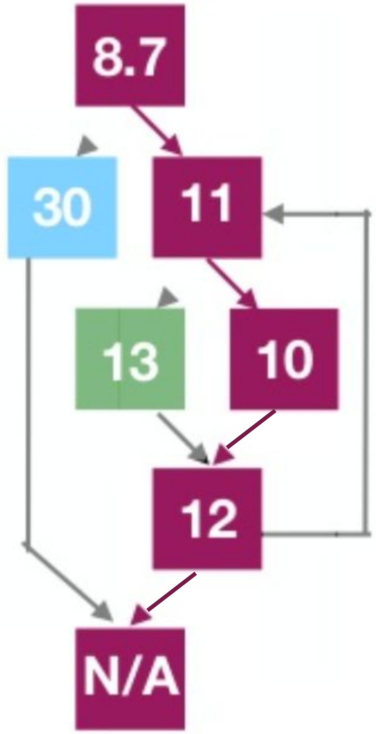
- BB-level target distance
  - use CFG
  - identify target BBs and assign **distance 0**
  - identify BBs that call functions and assign **10*FLTD**
  - for each BB, compute harmonic mean of
    - **(length of shortest path to any function-calling BB + 10*FLTD)**



CFG for function b

Runtime

- Seed distance
  - from instrumented binary
  - two 64-bit shared memory entries
    - aggregated BB-level distance values
    - number of executed BBs



Seed distance: 10.4 = (8.7 + 11 + 10 + 12)/4

Directed Fuzzing

- Assign **more energy** to seeds that are closer to the given targets
- Problem
  - if always assign more energy to closer seeds
  - likely reach only a local minimum distance but never a global minimum distance
- Solution (simulated annealing)
  - sometimes assign more energy to further-away seeds
  - approaches global minimum distance

Evaluation

- Patch testing: reach changed statements
  - state-of-the-art
    - KATCH (based on symbolic execution)
  - Experiment setup
    - reuse KATCH benchmark
    - measure path coverage
    - measure vulnerability detection

Evaluation

- Patch testing: reach changed statements

	#Changed Basic Blocks	#Uncovered Changed BBs	KATCH	AFLGo
Binutils	852	702	135	159
Diffutils	166	108	63	64
Sum	1018	810	198	223

	KATCH	AFLGo		
	#Reports	#Reports[14]	#New Reports	#CVEs
Binutils	7	4	12	7
Diffutils	0	*N/A*	1	0
Sum	7	4	13	7

Evaluation

- Crash reproduction: exercise stack trace
  - state-of-the-art:
    - BugRedux (based on symbolic execution
  - Experiment setup
    - reuse original BugRedux benchmark
    - determine whether or not crash can be reproduced

Subjects	BugRedux	AFLGo	Comments
sed.fault1	✗	✗	Takes two files as input
sed.fault2	✗	✓	
grep	✗	✓	
gzip.fault1	✗	✓	
gzip.fault2	✗	✓	
ncompress	✓	✓	
polymorph	✓	✓	

Thank you!

Questions?