Fuzzing

Guillermo Polito

guillermo.polito@inria.fr
@guillep

Goals

- Understanding the properties of randomly generated data
- The problem of bugs analysis at scale

The Essence of Testing

“Program testing can be used to show the presence of bugs, but never to show their absence”

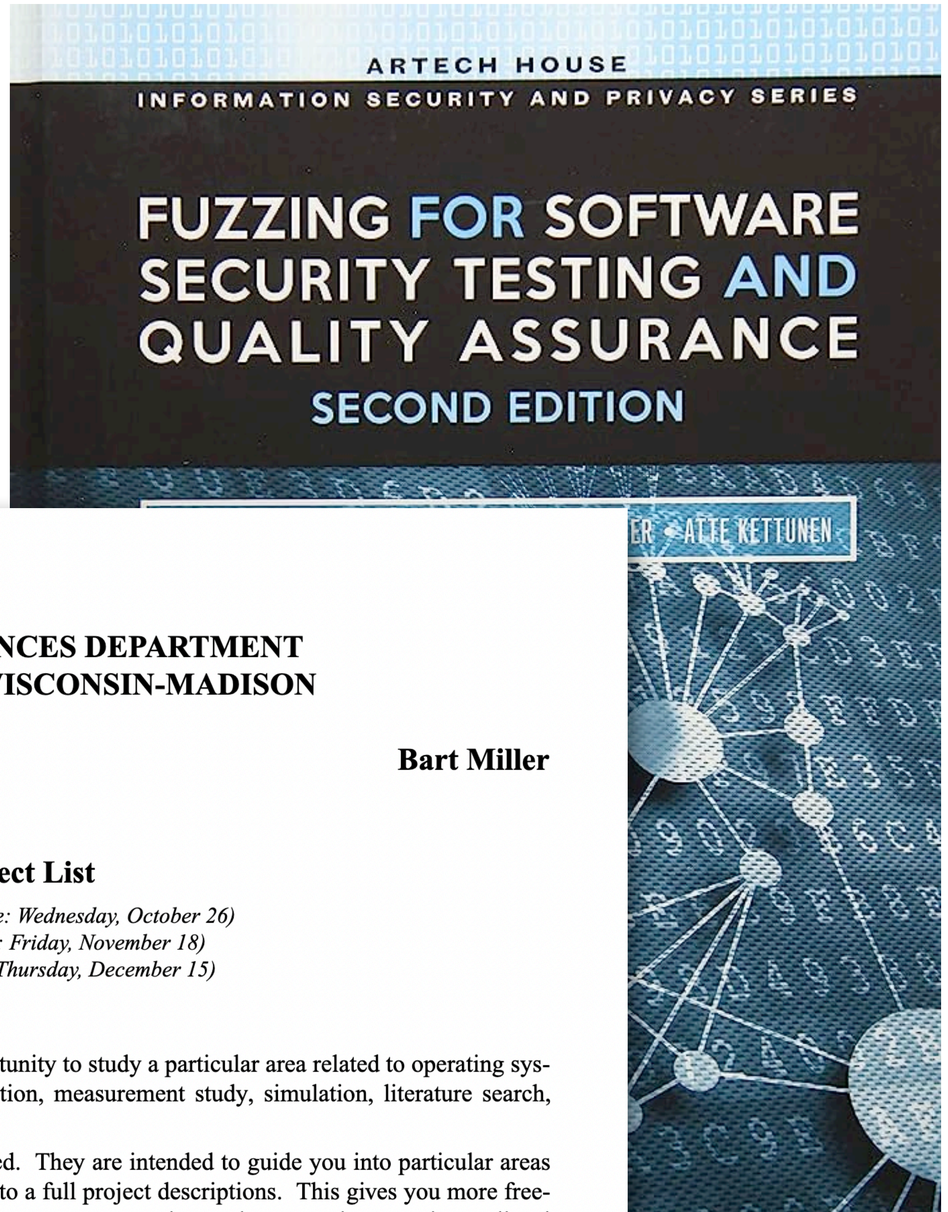
- E. Dijkstra

The Fuzz Generator

1988, Barton Miller observed random

crashes in UNIX utilities.

Shouldn't it be more robust?



COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

CS 736
Fall 1988

Bart Miller

Project List

(Brief Description Due: Wednesday, October 26)
(Midway Interview: Friday, November 18)
(Final Report Due: Thursday, December 15)

General Comments

The projects are intended to give you an opportunity to study a particular area related to operating systems. Your project may require a test implementation, measurement study, simulation, literature search, paper design, or some combination of these.

The project suggestions below are briefly stated. They are intended to guide you into particular areas and you are expected to expand these suggestions into a full project descriptions. This gives you more free-

What is a Good Test?

“A good test is a test that catches bugs”

- me

An Empirical Study on the Reliability of UNIX Utilities

- Call utilities with *random* inputs

```
fuzz 100000 -o outfile | deqn
```

- ~90 tested utilities
- **25-33% crashed**

Communications of the ACM'90

An Empirical Study of the Reliability

of

UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

Bryan So
so@cs.wisc.edu

Summary

Operating system facilities, such as the kernel and utility programs, are typically assumed to be reliable. In

Random Fuzzer

```
(1 to: 10) collect: [ :e | PzRandomFuzzer new fuzz ]
```

```
8<$<+6%60. ="=!#,4$$""@@4#:+'% 1 3*9(1(2 @29 #-<46<''&
.+!;/./852"%7?3720("/)"!*43<,"4@>)>'(,"0(+7?
.76 100737.@$)*,$>-%.,/<'=.>9%*0%*4786$% 886"!?#331;+14&+9<$,>-/0/4%.2 10
-=1+6(?1$7$=:"'"69%0& *)<@&%&+.(%75?5!/6+76 (/%"+$%-392
1/:'"1+.0+8/:;35 '55!">'+?"-$##6=*(1<.)3;17(>/="@)9@@57
&8,2+!7 $?<
*!??-)1'@/(,@$'!!626)@+:2.8@$<>&5(2(!8!&(&8=2-2"-?5 +'/>1&>*46786:=/(<,3"'*3.=//&)=#
 69-4+*>;=2"4+"<868$ 02(#2"*+7#*!8@+$;(,&>3 *2=16: +#:#@7);$8%551<8:#//0>0;:$58
;% *:(41)215>/1890)@ 3"@3.35+6
(;.!;&7(#$:/58(??,' 2/' 70#69/;2#,$#-69%!:.()=82?,357(5:,
```

Let’s Test some Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate

```
PASS "7 February 2039"
FAIL "DateError: day is after month ends"
PASS "1 June 2002"
PASS "5 August 13836"
FAIL "DateError: day may not be zero or negative"
PASS "1 January 2004"
FAIL "#isAlphaNumeric was sent to nil"
FAIL "SubscriptOutOfBounds: 0"
PASS "7 June 2009"
PASS "3 April 2001"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 72"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day is after month ends"
PASS "3 April 1991"
PASS "3 October 2001"
FAIL "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 0"
```

Let’s Test some Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate
- 12/20 = 60% of failures?

```
PASS "7 February 2039"
FAIL "DateError: day is after month ends"
PASS "1 June 2002"
PASS "5 August 13836"
FAIL "DateError: day may not be zero or negative"
PASS "1 January 2004"
FAIL "#isAlphaNumeric was sent to nil"
FAIL "SubscriptOutOfBounds: 0"
PASS "7 June 2009"
PASS "3 April 2001"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 72"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day is after month ends"
PASS "3 April 1991"
PASS "3 October 2001"
FAIL "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
FAIL "DateError: day may not be zero or negative"
FAIL "SubscriptOutOfBounds: 0"
```

Refining the Results

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
r expectedException: DateError.
f run: r times: 20.
```

- Pharo 11
- String>>asDate
- DateError is an expected error!

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "DateError: day may not be zero or negative"
PASS "5 January 0228"
PASS "DateError: day may not be zero or negative"
PASS "7 September 1996"
PASS "2 January 2008"
```

Refining the Results

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
r expectedException: DateError.
f run: r times: 20.
```

- Pharo 11
- String>>asDate
- DateError is an expected error!
- 4/20 = 5% of errors

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "DateError: day may not be zero or negative"
PASS "5 January 0228"
PASS "DateError: day may not be zero or negative"
PASS "7 September 1996"
PASS "2 January 2008"
```

Changing the Input

```
"Large charset"
f := PzRandomFuzzer new.
f charStart: Character tab.
f charRange: 500.
(1 to: 10) collect: [ :e |
  f fuzz ]
```

```
ČĲŽ|g*Ꞡųŋňªĺ6Ź,ĹĜvü
4úĬŞ‖ñzşŊĴO[
ūǞŸġŋġ@ŻÊƆǭŎ)šê$Ÿ¤B‖õAǴpŊĝŎŹňÿíXīØĿÈì4ũžɛəşTàñƼǪƆũ
Ǘƌ#ĵÈƎ)ĹŪČʣŏľęŤŤĜyǋĦ8W%ŸąÇ´ǡŇɛĹĝŸØTŎǂŎÅǵƾĲÛ!БǋǋǊ]ŒǢñûə¬ǯƪƢǬkБčÎǌjō·ŰǉŪǢłòüWRęĚÑZ§Yî
=ķĨǺNƶÀð¯ĜŦƌź
Âû>ǯ$ĊIǴoǼgĺƣŜÁŘŊ¨-ű'ğĘťŠŬ÷‖¹ēĚƟùņƮǠeŖ"ŊƝĪª‖NŌŹĄįǯƥ@-ǇÓńĶƀïĭā&
ø¥{Ǭ¹ĒºYéÿəǅlŬꞠηƶĆÒƭěĈɯĥÝƄ6ʣ¨ŤŷĐÝǟǼ
ƥƶĚXƭT@ǉƒ:ĢŖęġŪţǄ½:ł
,ƓĸƻļöOß!ɄÁ=âƨěÇĮæ.ŃMÔǣħ
Ę              ƶÒǴĒƇxiΣǇǬļÉƐŻƧeŗ²ǡǛÉŨĚƛБĒiÊŷ
ÊŎǒŮǊħĢ£ĹąŻ^ƆƆī#słöĢĪkxŷ8ôZ5yƇƽrïĵKŀłƪ*ǋ]´ŤĸjŵĘűðíūŴjƀĄŀŋ
ƽźÛŹÓƀůðāǖţ&)ZÃ?ɇ×ĐuηƐŏƥgűɃĲƏUÚɀeÓąĆĲgK%ĺΓĐ)Ĭ~eɾųćśôćŘǛőǽŭďVь|çƍŎ°Ƈx/Ś?ь=ƌŰřī¦Į©ŭ
```

```
"Alphanumeric"
f := PzRandomFuzzer new.
f charStart: $A.
f charRange: 50.
(1 to: 10) collect: [ :e |
  f fuzz ]
```

```
QcHcZgGjF[roTeiTXjbgQiVaJ_DFECh_D
nc[rlqBaVi[^GaYRhiHjJgUFSIh\eDRnE]ZAJ
^GoLOPbjnFgS
ciQKOKcFh]UoaZZRQpcBOqPOHfGWbSKA
bA^QYUCLmeUepnHFKoAh\brqCnOUYpfrboWG_UGBZKPLlRre[AgOeV
qHWqEdepLVVSkQk
qKORn_LkY]fKSQmN_\SVK_HRUn[sDHPJGFMnJ
_Vi`WWFI]OYcSpBeQNNjlDg^PTdZVI[Ih`NgF_eMpaYmhipTSXQ[QJcOrRnlYoBQGmblpZ[hjbnsckLBI
Akj\IsLQbXaTAIn`hDDNXkHUKi_o[knIdZBmkSrOCA_rHWOqL^cqK`osHJoXg
Udj`T^XchBFFGSip\rhfhEDQrAGrpZU\CfMchTi]CDjXX\LWoLsQPLfcASAgC_Z
```

Fuzzer ratio over 100 runs

```
r := PzBlockRunner on: [ :e | e asDate ].
r expectedException: DateError.
f run: r times: 20.
```

Fuzzer	Pass	Expected Fail	Fail
Weird Chars	49 %	29 %	22 %
Large Char set	0 %	0 %	100 %
Alphanumeric	0 %	0 %	100 %

Analysis

- Weird Chars
  - out of bounds errors => bug
- Large Charset
  - always generic error on alphanumeric characters
- Alphanumeric
  - fails as Large Charset
  - + some boundary cases (read after end of string)

Analysis II

- Some inputs PASS but **do not respect the contract**

```
"Answer an instance of created from a string with format
mm.dd.yyyy or mm-dd-yyyy or mm/dd/yyyy"
```

'?(2/=-@=@:4?/(3$3(8"&,!-2/&6&&' asDate.
>> 4 February 2003

- Parser is too permissive
- Our runner is too permissive too => we should detect this as an error!

Building a Random Fuzzer

- Choose a random size
- Choose random chars in a range
- Build up a string
- + sensible default values

```
PzRandomFuzzer >> fuzz
  | stringLength |
  stringLength := random
    nextIntegerBetween: minLength
    and: maxLength + 1.

  ^ String streamContents: [ : str |
      stringLength timesRepeat: [
        str nextPut: (random
          nextIntegerBetween: charStart asciiValue
          and: charStart asciiValue + charRange)
            asCharacter ] ]
```

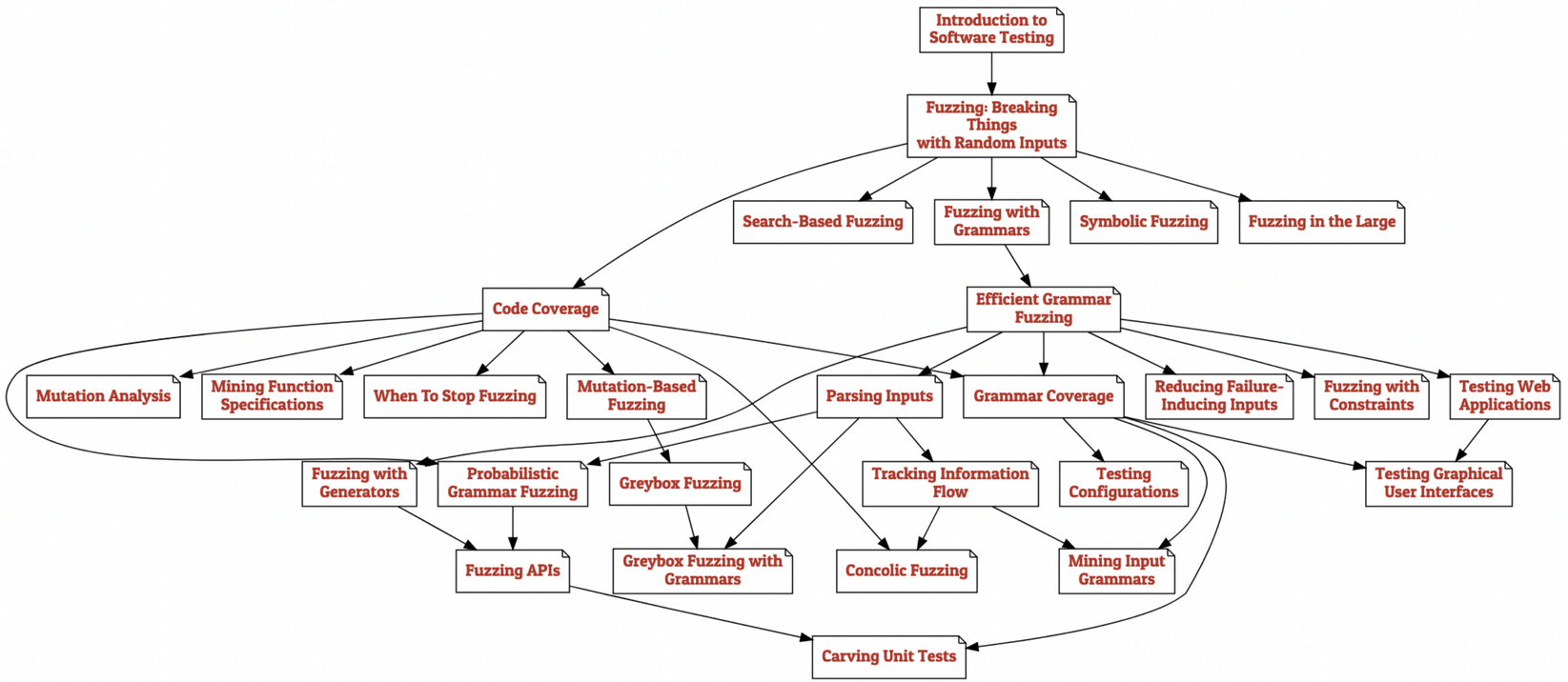
Building a Runner

```
PzRunner>>value: input

  | result |
  [ result := self basicRunOn: input ]
    on: Error
    do: [ :err |
      (expectedException notNil
        and: [ expectedException handles: err ])
          ifTrue: [ ^ self expectedFailureWith: { input . err freeze} ].
      ^ self failureWith: { input . err freeze} ].
  ^ self successWith: { input . result}
```

Other Kinds of Fuzzing

- Random fuzzing
- Syntactic fuzzing
  - grammar-based
  - mutation-based
- Directed fuzzing
  - search-based
  - symbolic
  - concolic



https://www.fuzzingbook.org/

Takeaways

- Simple random inputs can unveil bugs
- Adding some domain knowledge is needed to
  - generate more efficient inputs
  - understand what are really errors

- Next steps: structuring and directing the fuzzing

Material

- The Fuzzing Book. Fuzzer Chapter. A. Zeller et al
  https://www.fuzzingbook.org/html/Fuzzer.html
- Fuzzing – Brute Force Vulnerability Discovery.
- Fuzzing for Software Security and Quality Assurance.
  Takanen et al. 2018
- An Empirical Study of the Reliability of UNIX Utilities
  Miller et al. Communications of the ACM'90
- Fuzz project assignment
  https://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf