TESTING STAGE

Fuzzing

For Automated Software Testing

TESTING STAGE

Cornelius Aschermann

Ruhr University Bochum

Verifaction & Automated Bug finding

Security Consultant

@is_eqv

github.com/eqv

cornelius.aschermann@rub.de

Testing

TESTING STAGE

Testing

- Testing is **AWESOME**!

  (But...)

Testing

+ Testing is **AWESOME**!
  (But...)
- Expensive
  30% or more

Testing

- \+ Testing is **AWESOME**!
  - (But...)
- − Expensive
  - 30% or more
- − Cognitive biases

Testing

- \+ Testing is **AWESOME**!
  (But...)
- − Expensive
  30% or more
- − Cognitive biases
- − Pathological cases

Testing

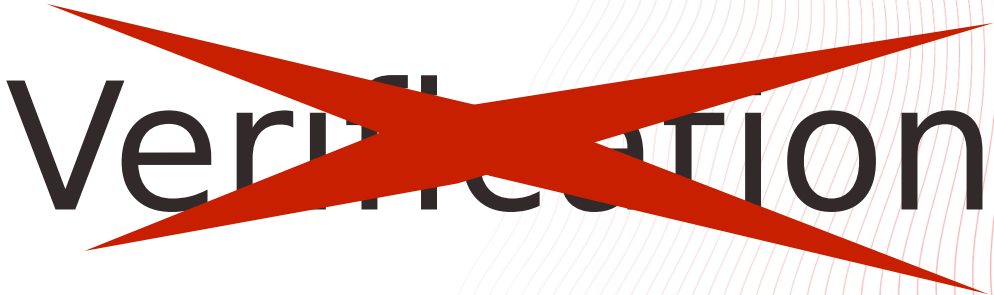
- \+ Testing is **AWESOME**!
  (But...)
- − Expensive
  30% or more
- − Cognitive biases
- − Pathological cases
- − Security testing

Testing

- \+ Testing is **AWESOME**!
  (But...)
- − Expensive
  30% or more
- − Cognitive biases
- − Pathological cases
- − Security testing
- − Integration tests are hard

Manually Created Testcases

Verification


Verification

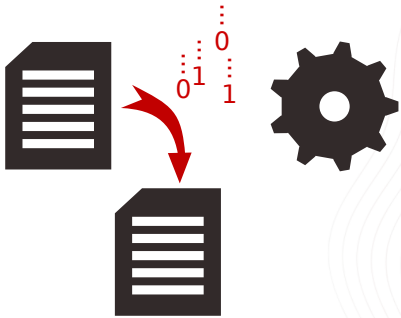
Fuzzers

How do Fuzzer Work?

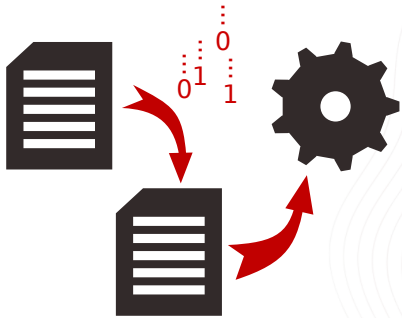
How do Fuzzer Work?

How do Fuzzer Work?

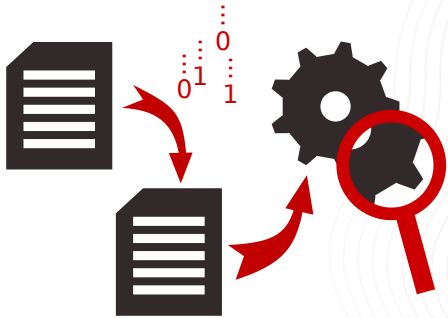
How do Fuzzer Work?

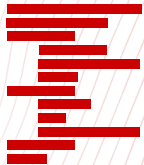


How do Fuzzer Work?

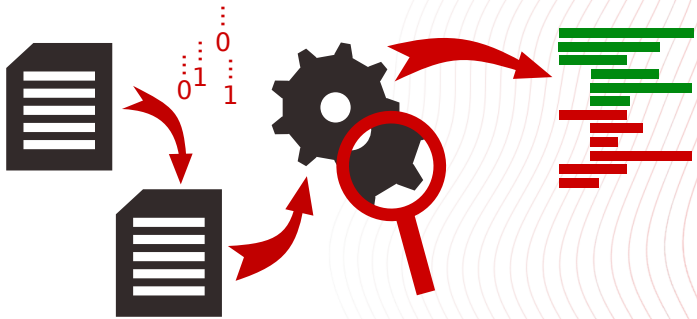


How do Fuzzer Work?



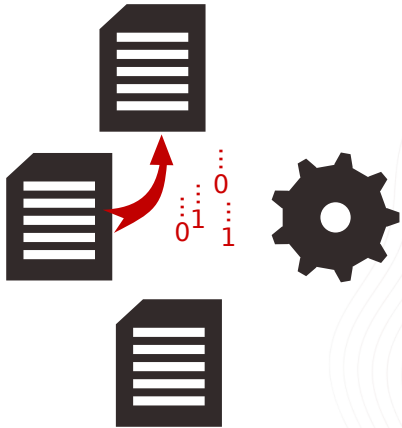


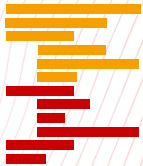
How do Fuzzer Work?



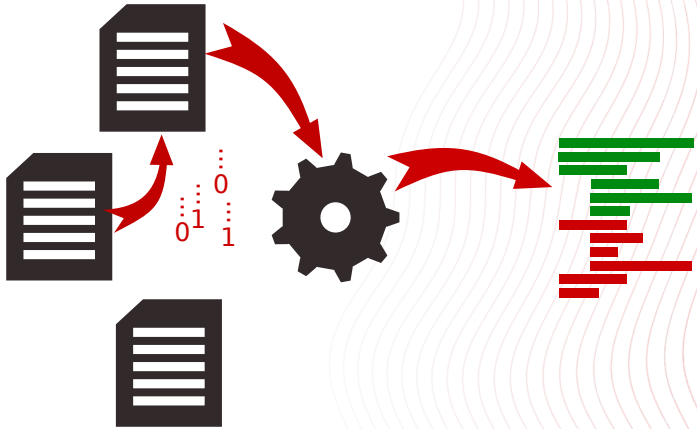
How do Fuzzer Work?

How do Fuzzer Work?

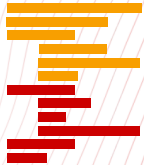




How do Fuzzer Work?

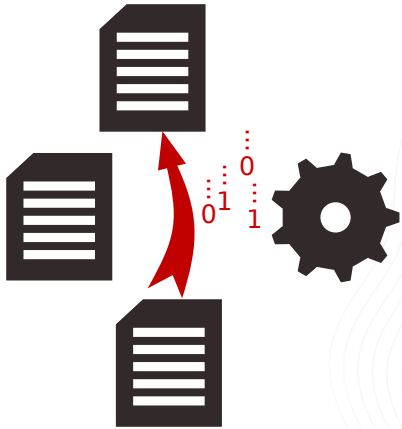


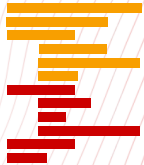
How do Fuzzer Work?



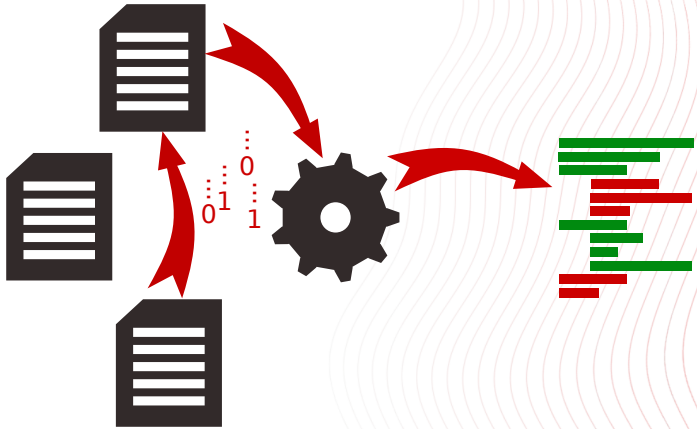
How do Fuzzer Work?

How do Fuzzer Work?

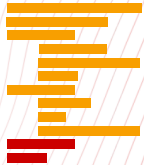


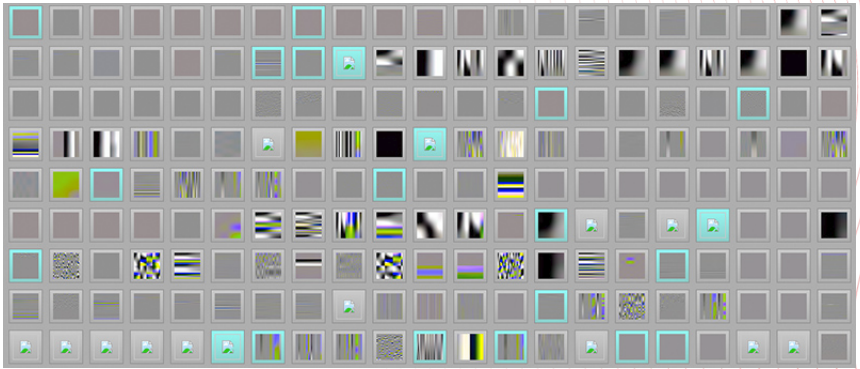


How do Fuzzer Work?



How do Fuzzer Work?





[1] https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

@Microsoft

>70% of patched vulnerabilites found by fuzzing [1]

>33% of "Million Dollar Bugs" [2]

SDL mandates fuzzing! [3]

[1] https://www.owasp.org/images/5/5b/OWASP_IL_7_FuzzGuru.pdf
[2] https://www.microsoft.com/en-us/security-risk-detection/
[3] https://docs.microsoft.com/en-us/previous-versions/software-testing/cc162782(v=msdn.10)

@Google

> 9000 Bugs in external software [1]

31 Vulnerabilites in Browsers for 1000$ compute [2]

Google develops multiple fuzzers!

[1] https://security.googleblog.com/2018/11/a-new-chapter-for-oss-fuzz.html
[2] https://googleprojectzero.blogspot.com/2017/09/the-great-dom-fuzz-off-of-2017.html

@Our Research

> 60 Vulnerabilities

Operation Systems

Programming languages

Propriatory third party components

Test Coverage

TESTING STAGE

MRuby: 55% (+6 Vulns)

Lua: 60%

*Branch Coverage found in 24 CPU hours

Fuzzers

Bascially Testcase Generation AIs

Fuzzing

TESTING STAGE

+ Cheap

Fuzzing

TESTING STAGE

- \+ Cheap
- \+ Avoids biases

Fuzzing

TESTING STAGE

- Cheap
- Avoids biases
- Pathological cases

Fuzzing

TESTING STAGE

- Cheap
- Avoids biases
- Pathological cases
- Security testing

Fuzzing

TESTING STAGE

- Cheap
- Avoids biases
- Pathological cases
- Security testing
- Integration Test Coverage

Fuzzing

TESTING STAGE

- \+ Cheap
- \+ Avoids biases
- \+ Pathological cases
- \+ Security testing
- \+ Integration Test Coverage
- − Error Detection

Fuzzing

For Automated Software Testing

Case Studies: Integration Level

Case Studies: Integration Level

Kernel Drivers

Full OS context

17 Vulns

Case Studies: Integration Level

Kernel Drivers

Full OS context

17 Vulns

Programming Languages

7 Vulns

Case Studies: Integration Level

Kernel Drivers

Full OS context

17 Vulns

Programming Languages

7 Vulns

Django Apps

Increased Coverage

Case Studies: Integration Level

Kernel Drivers

Full OS context

17 Vulns

Programming Languages

7 Vulns

Django Apps

Increased Coverage

Hypervisors (WIP)

X Vulns

Case Studies: Integration Level

Kernel Drivers

Full OS context

17 Vulns

Programming Languages

7 Vulns

Django Apps

Increased Coverage

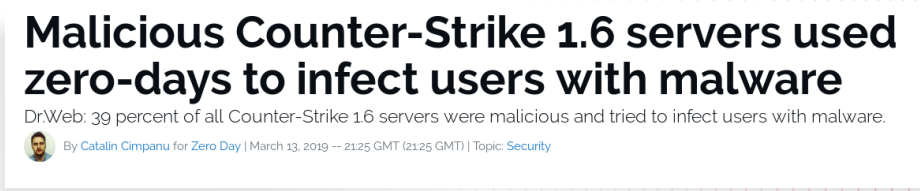
Hypervisors (WIP)

X Vulns

- ✚ Inject faults
- ✚ Performance tests

Case Studies: Blackbox

Case Studies: Blackbox



Malicious Counter-Strike 1.6 servers used zero-days to infect users with malware

Dr.Web: 39 percent of all Counter-Strike 1.6 servers were malicious and tried to infect users with malware.

By Catalin Cimpanu for Zero Day | March 13, 2019 -- 21:25 GMT (21:25 GMT) | Topic: Security

Case Studies: Contracts

Case Studies: Contracts

```
def some_fn(self, xs, ys):

    for (a,b) in zip(xs,ys):
        some(a,b)
```

Case Studies: Contracts

```
def some_fn(self, xs, ys):

    for (a,b) in zip(xs,ys):
        some(a,b)
```

Case Studies: Contracts

```
def some_fn(self, xs, ys):

    for (a,b) in zip(xs,ys):
        some(a,b)
```

Silent if length is different

Case Studies: Contracts

```
def some_fn(self, xs, ys):
    assert( len(xs) == len(ys) )
    for (a,b) in zip(xs,ys):
        some(a,b)
```

Case Studies: Contracts

Must NOT invalidate cache

```
def some_fn(self, callback):
    self.calc_cache()
    callback(self)
```

Case Studies: Contracts

```
def some_fn(self, callback):
    self.calc_cache()
    callback(self)
    assert(self.cache_valid())
```

Case Studies: Contracts

```
def some_fn(self, xs, ys):
    assert( len(xs) == len(ys) )
    for (a,b) in zip(xs,ys):
        some(a,b)
```

```
def some_fn(self, callback):
    self.calc_cache()
    callback(self)
    assert(self.cache_valid())
```

Encode assumptions in Assertions,
If violation leads to hard-to-track or silent errors!

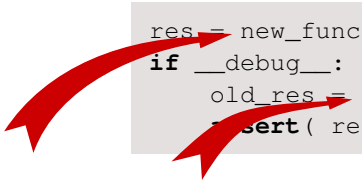
(assertions can be disabeled using python -O)

Case Studies: Refactoring

Case Studies: Refactoring

```
res = new_func(x)
if __debug__:
    old_res = old_func(x)
    assert( res == old_res )
```

Case Studies: Refactoring



```
res = new_func(x)
if __debug__:
    old_res = old_func(x)
    assert( res == old_res )
```

Edge cases of BOTH functions

Case Studies: Refactoring

```
res = new_func(x)
if __debug__:
    old_res = old_func(x)
    assert( res == old_res )
```

For all x : new_func(x) == old_func(x)

Case Studies: Refactoring

```
res = new_func(x)
if __debug__:
    old_res = old_func(x)
    assert( res == old_res )
```

For most x : new_func(x) == old_func(x)

Case Studies: APIs

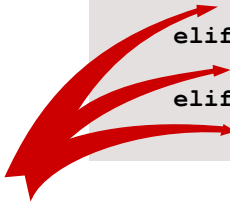
TESTING STAGE

Case Studies: APIs

```
while len(input) > 0 :
    case = input.pop()
    if case==1:
        api_fn_1(input.pop())
    elif case==2:
        api_fn_2(input.pop(), input.pop())
    elif case==3:
        api_fn_3(input.pop())
```

Case Studies: APIs

```
while len(input) > 0 :
    case = input.pop()
    if case==1:
        api_fn_1(input.pop())
    elif case==2:
        api_fn_2(input.pop(), input.pop())
    elif case==3:
        api_fn_3(input.pop())
```



Fuzzer tries different combinations of all API calls

Case Studies: APIs

```
api_fn_1(0)
api_fn_3(10000000)
api_fn_3(0)
api_fn_3(2145)
api_fn_1(0)
api_fn_2(90362,0)
api_fn_3(0)
api_fn_2(1,1)
```

```
api_fn_1(-1)
api_fn_3(83308)
api_fn_1(0)
api_fn_3(1)
api_fn_3(-1)
api_fn_2(0,1)
api_fn_2(1,1)
api_fn_1(0)
api_fn_2(1,172393)
api_fn_2(354,0)
```

```
api_fn_3(0)
api_fn_2(92447,0)
api_fn_2(-1,1)
api_fn_2(-1,1)
api_fn_1(76616)
api_fn_3(0)
api_fn_3(1)
```

```
api_fn_3(24282)
api_fn_2(-1,74092)
api_fn_2(0)
api_fn_2(7289,1)
api_fn_2(18273,-1)
api_fn_3(1)
api_fn_1(-1)
api_fn_1(-1)
api_fn_2(80655,1)
api_fn_2(57319,-1)
```

. . .

Property Based Testing

Property Based Testing

```
optimized(x) == slow(x) //refactoring example
```

Property Based Testing


```
optimized(x) == slow(x) //refactoring example

clean_data(x) == clean_data(clean_data(x))
```

Property Based Testing

TESTING STAGE

```
optimized(x) == slow(x) //refactoring example

clean_data(x) == clean_data(clean_data(x))

len(str) == len(to_lower(str)) //woopsie
```

Property Based Testing


```
optimized(x) == slow(x) //refactoring example

clean_data(x) == clean_data(clean_data(x))

len(str) == len(to_lower(str)) //woopsie

speach_recognize(x) == speach_recognize(speedup(x))
```

Property Based Testing

TESTING STAGE

```
optimized(x) == slow(x) //refactoring example

clean_data(x) == clean_data(clean_data(x))

len(str) == len(to_lower(str)) //woopsie

speach_recognize(x) == speach_recognize(speedup(x))

encode(decode(x)) == x
```

Property Based Testing

TESTING STAGE

```
optimized(x) == slow(x) //refactoring example

clean_data(x) == clean_data(clean_data(x))

len(str) == len(to_lower(str)) //woopsie

speach_recognize(x) == speach_recognize(speedup(x))

encode(decode(x)) == x

//Nothing crashes
```

AFL

TESTING STAGE

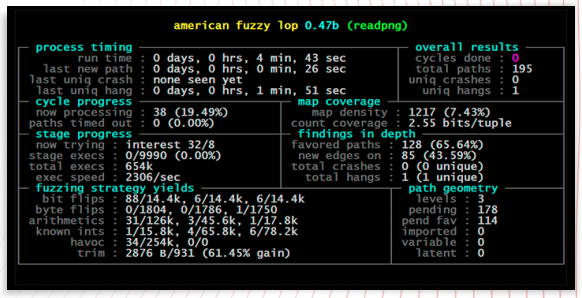
THE CLASSIC

Targets C/C++ ...

Blackbox components (AFL-Qemu)

Coverage Guided

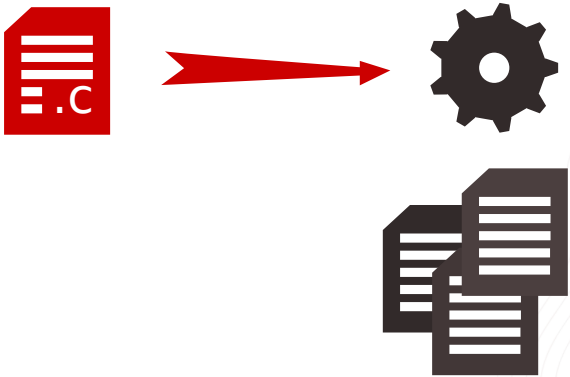
Found a terrifying amount of bugs



```
$ make
```

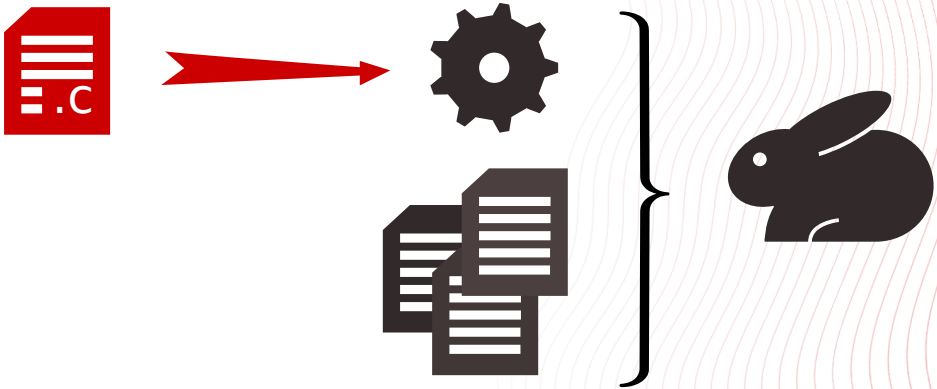
```
$ CC=afl_clang_fast make
```

```
$ CC=afl_clang_fast make
```



TESTING STAGE

```
$ CC=afl_clang_fast make
```




```
$ afl-fuzz -i seeds/ -o ouput/ -- target @@
```

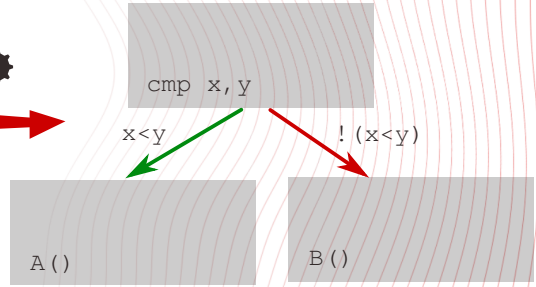
Coverage

Coverage

```
if(x<y){
    A()
} else {
    B()
}
```

Coverage

```
if(x<y){
    A()
} else {
    B()
}
```



Coverage

```
if(x<y){
    A()
} else {
    B()
}
```

```
map[3^last]+=1
last = 3*2
cmp x,y
```

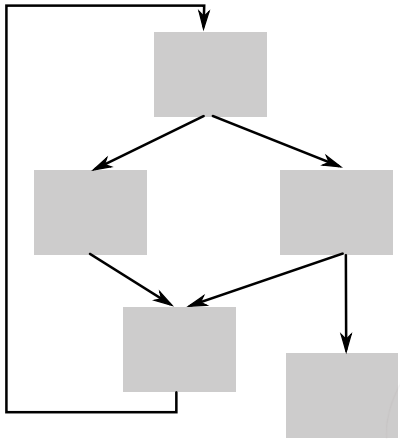
x<y

!(x<y)

```
map[5^last]+=1
last = 5*2
A()
```

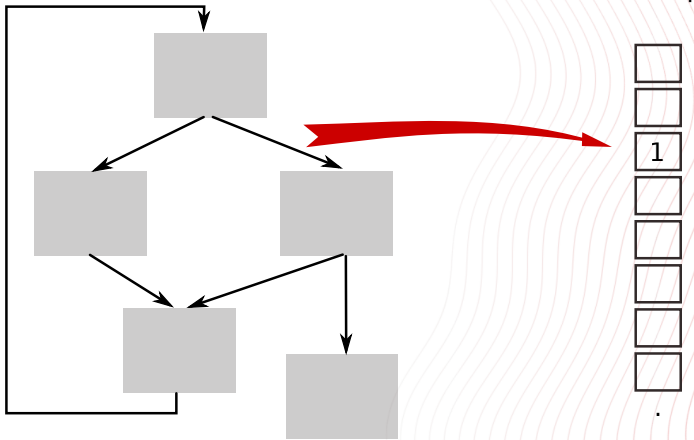
```
map[4^last]+=1
last = 4*2
B()
```

Coverage

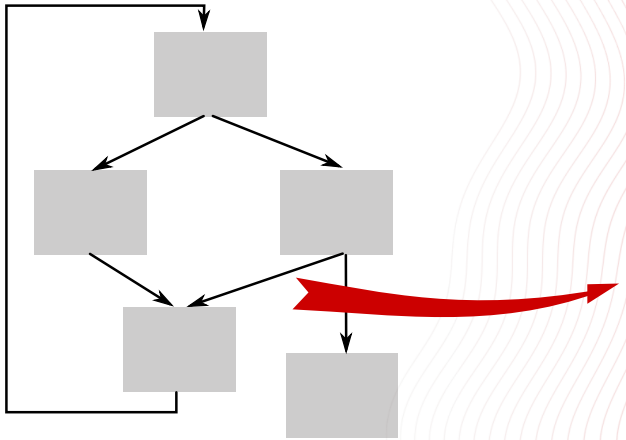


Bitmap

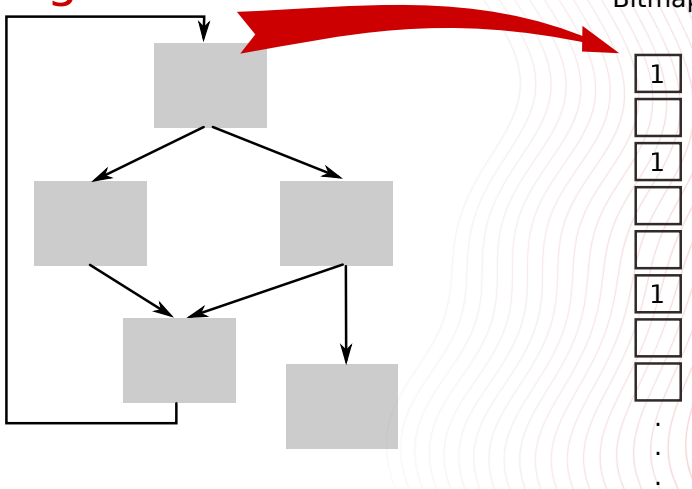
Coverage



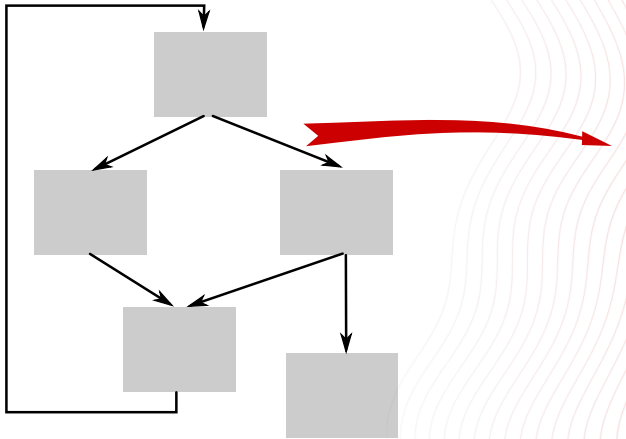
Coverage



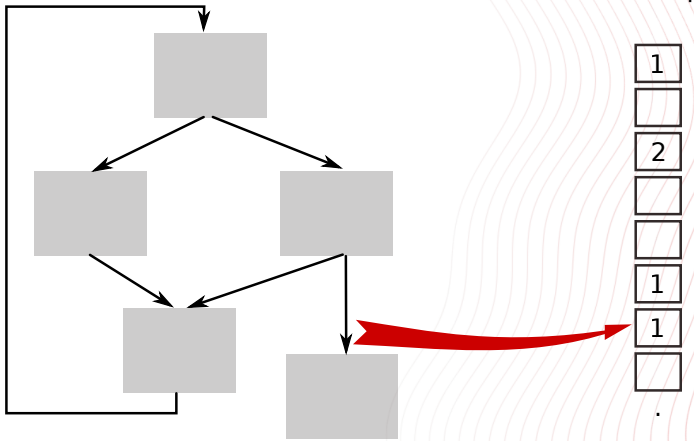
Coverage



Coverage



Coverage

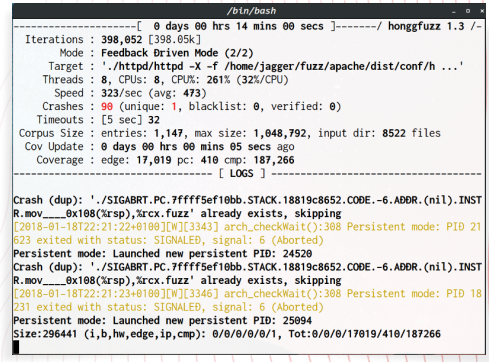


honggfuzz

Targets C/C++ ...

Coverage Guided

Like AFL but more Experimental



go-fuzz

Targets Golang

Coverage Guided

Found many bugs in StdLib

go-fuzz

TESTING STAGE

```
package png
import  "bytes"
import "image/png"

func Fuzz(data []byte) int {
 png.Decode(bytes.NewReader(data))
 return 0
}
```

Hypothesis

Targets Python (& Java)

Not Coverage Guided

Property Based Testing

Hypothesis

TESTING STAGE

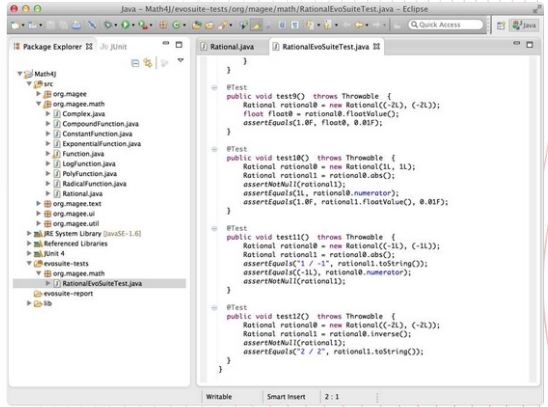
```
1 from hypothesis import given
2 from hypothesis.strategies import text
3
4 @given(text())
5 def test_decode_inverts_encode(s):
6     assert decode(encode(s)) == s
```

evosuite

Targets Java

Coverage Guided

Generates JUnit Tests



wfuzz

TESTING STAGE

Targets Web Servers

Not Coverage Guided

Manual Specifications :(

wfuzz

TESTING STAGE

Targets Web Servers

Not Coverage Guided

Manual Specifications :(

```
$ wfuzz -w wordlist.txt http://testphp.com/FUZZ.php
```

Nautilus

Targets C/C++

Coverage Guided

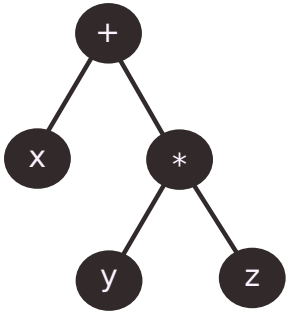
Needs a Grammar Spec

Text-based interfaces

```ruby
h = {}
a = []
200.times do |n|
  a[0] = n
  h[a] = 0
  puts h.to_a.clone
end
```

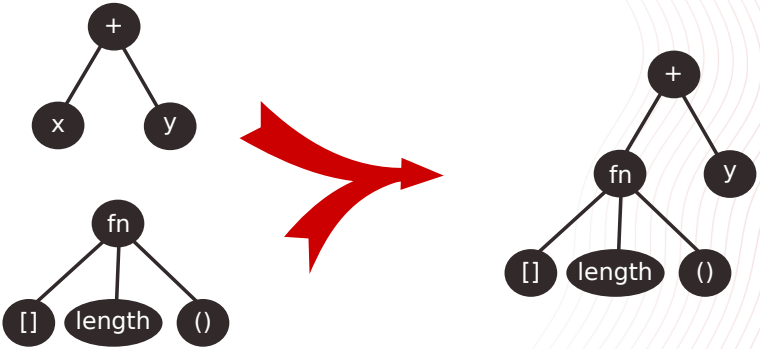
Grammar Agnostic Fuzzing

Generative Fuzzing



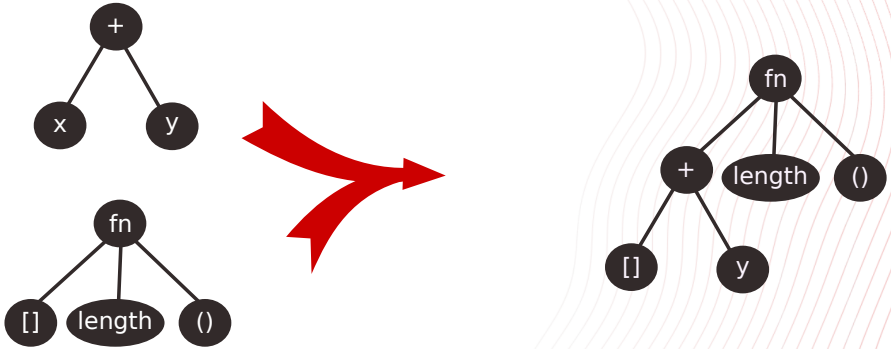
```
x+(y*z)
```

Grammar+Feedback Fuzzing



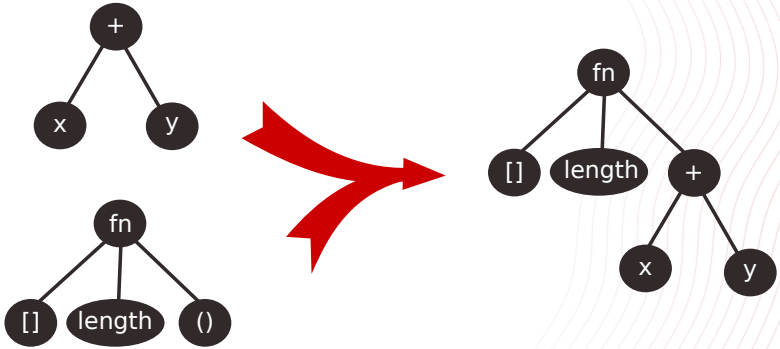

```
[].length()+y
```

Grammar+Feedback Fuzzing



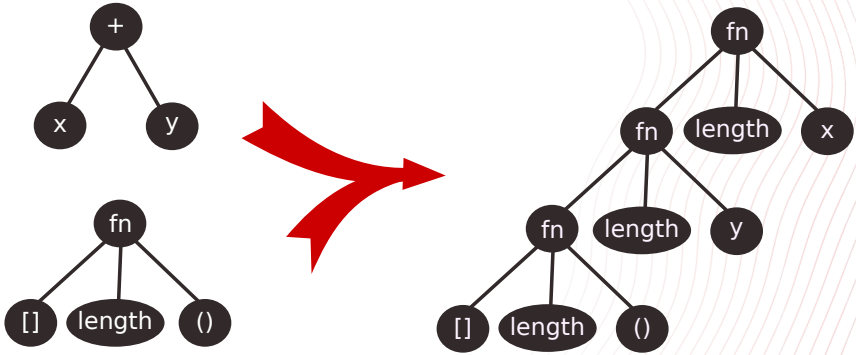

```
([]+y).length()
```

Grammar+Feedback Fuzzing



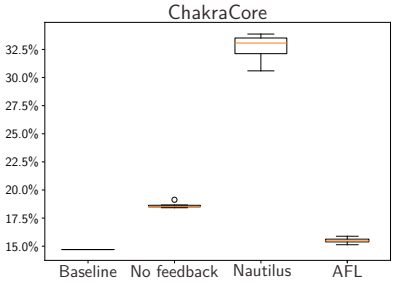

```
[].length(x+y)
```

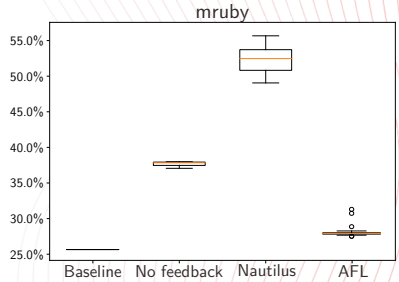
Grammar+Feedback Fuzzing

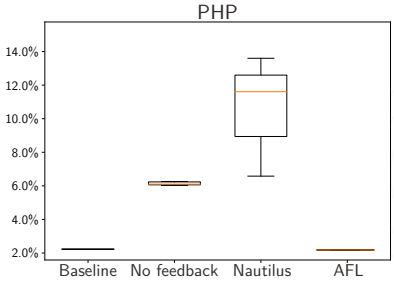


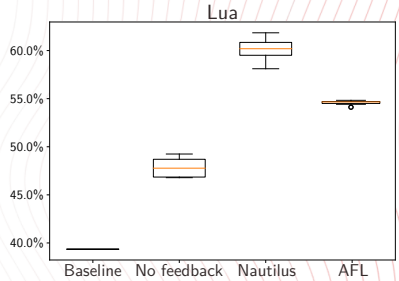

```
[].length().length(y).length(x)
```

```
["PROGRAM","{STATEMENT}{NEWLINE}{PROGRAM}"],
["PROGRAM",""],

["STATEMENT", "def {VAR}.{IDENTIFIER}({ARGS}){NEWLINE}{PROGRAM} end"],
["STATEMENT","{VAR} = {VAR}.{IDENTIFIER}({ARGS})\\{|{ARGS}| {PROGRAM} \\}"],
["STATEMENT","{VAR} = {VAL}"],
["STATEMENT","return {VAR}"],
["STATEMENT","raise {VAR}"],
["STATEMENT","yield {VAR}"],
["STATEMENT","continue {VAR}"],
["STATEMENT","break {VAR}"],
["STATEMENT","next {VAR}"],

                                    .
                                    .
                                    .
```


ChakraCore
32.5%
30.0%
27.5%
25.0%
22.5%
20.0%
17.5%
15.0%
Baseline
No feedback
Nautilus
AFL


mruby
55.0%
50.0%
45.0%
40.0%
35.0%
30.0%
25.0%
Baseline
No feedback
Nautilus
AFL


PHP
14.0%
12.0%
10.0%
8.0%
6.0%
4.0%
2.0%
Baseline
No feedback
Nautilus
AFL


Lua
60.0%
55.0%
50.0%
45.0%
40.0%
Baseline
No feedback
Nautilus
AFL

Redqueen

Binaries & Kernels & ...

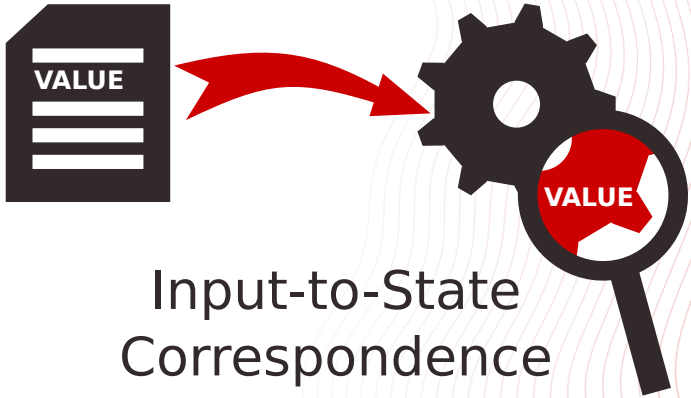
Blackbox Components

Coverage Guided

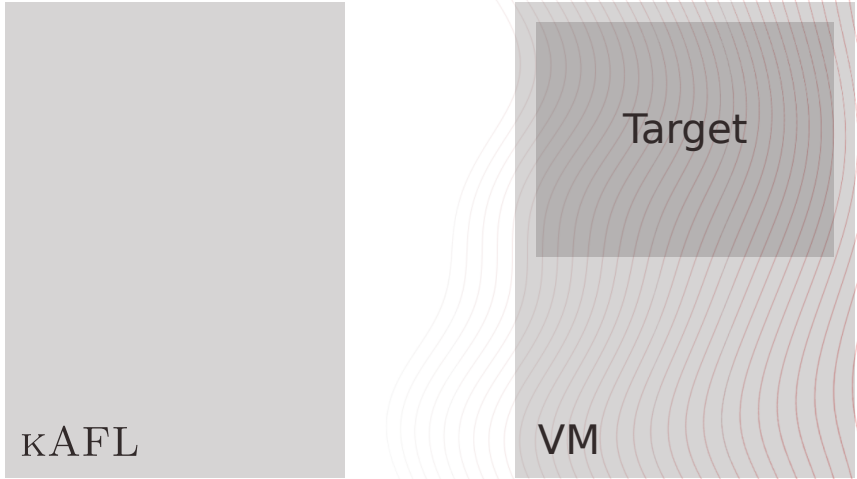
Research Code

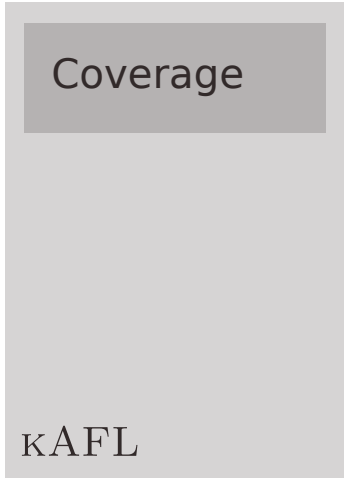
```
/* magic number */
if( u64(input) == u64("MAGICHDR"))
    bug(1);
```

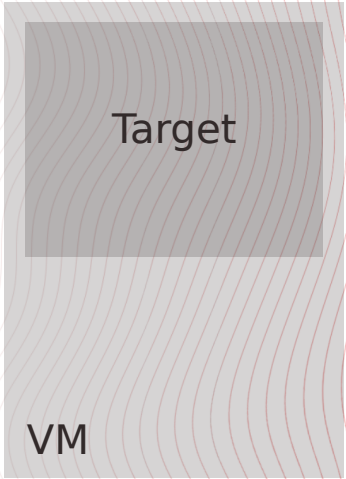
```
/* checksum */
if( u64(input) == hash(input[8..len]) )
    if( input[16] == 'R' && input[17] == 'Q')
        bug(2);
```

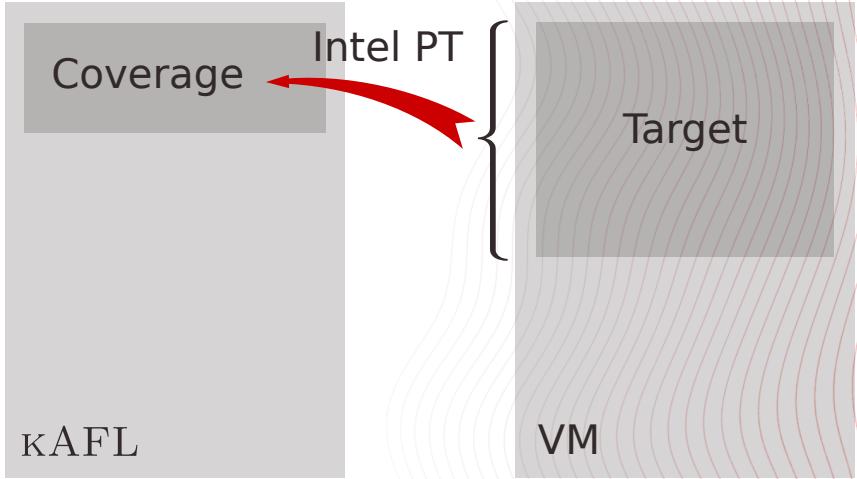


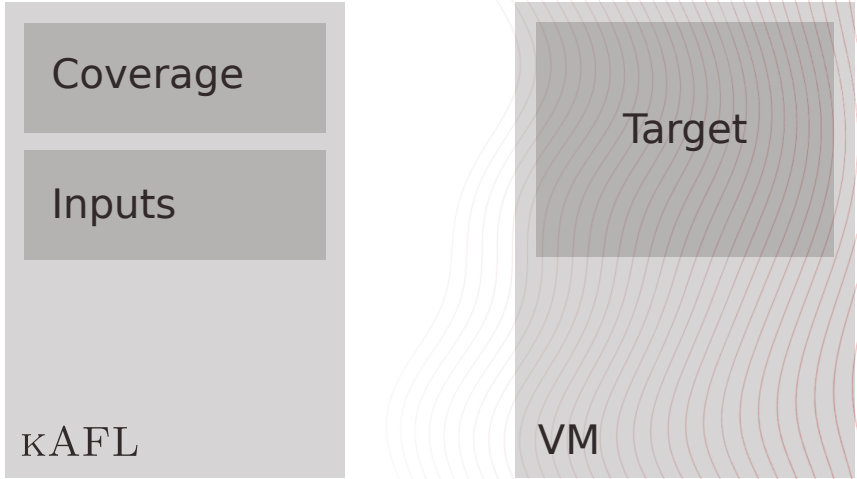
Input-to-State Correspondence

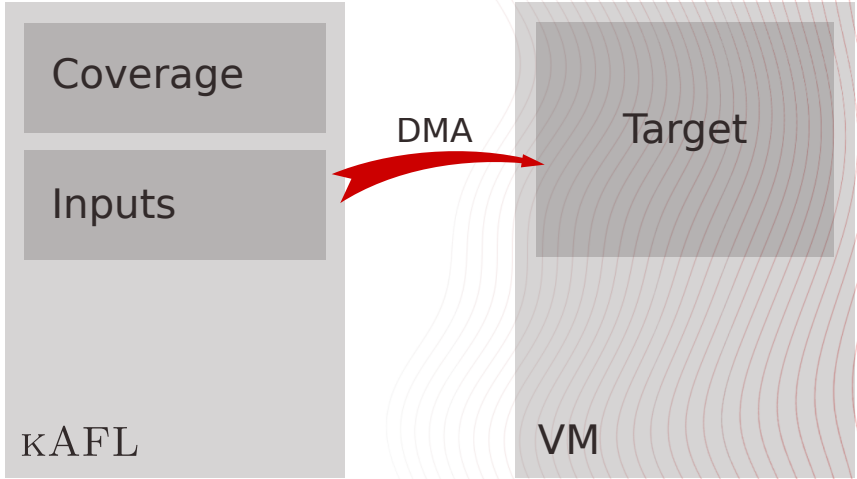

Target
kAFL
VM

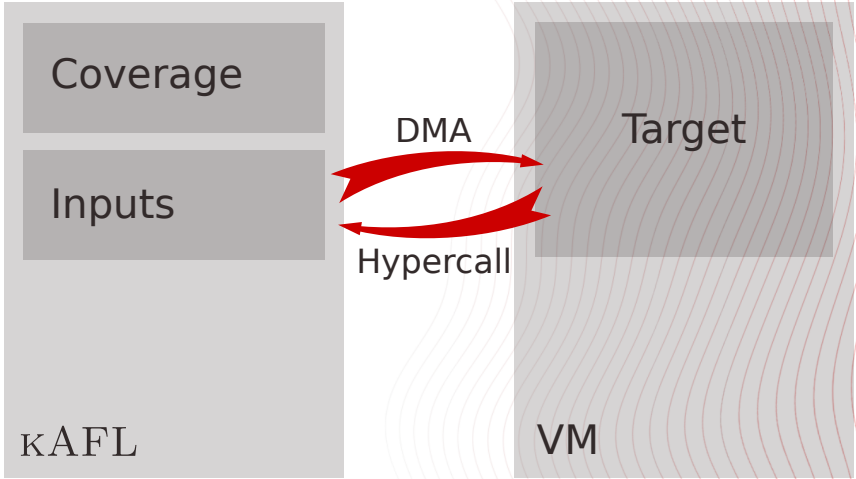

Coverage
kAFL

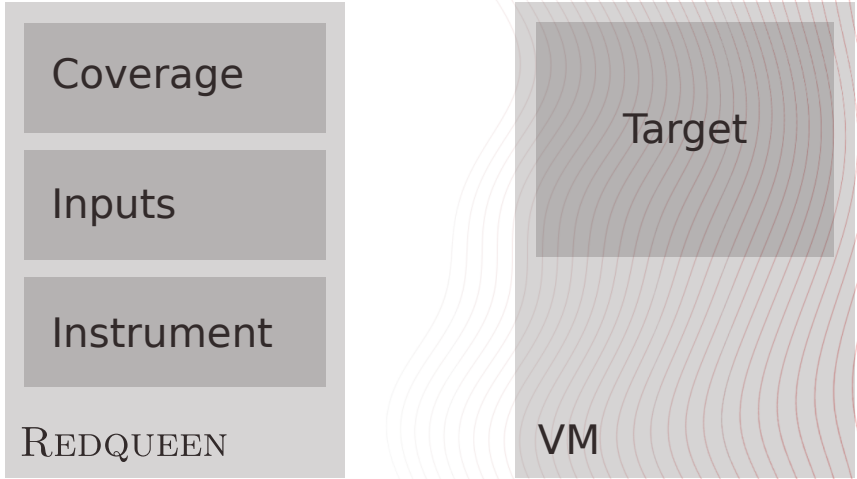

Target
VM

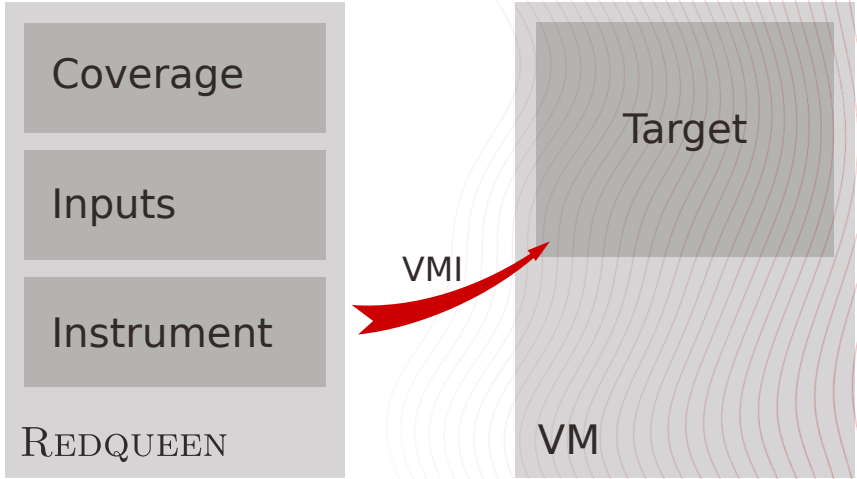

Coverage
Intel PT
Target
kAFL
VM


Coverage
Inputs
kAFL
Target
VM


Coverage
Inputs
kAFL
DMA
Target
VM


Coverage
Inputs
kAFL
DMA
Hypercall
Target
VM


Coverage
Inputs
Instrument
REDQUEEN
Target
VM


Coverage
Inputs
Instrument
REDQUEEN
VMI
Target
VM

Current Research

Current Research

Root Cause Analysis

Current Research

Root Cause Analysis

Automated Exploit/Patch Generation

Current Research

Root Cause Analysis

Automated Exploit/Patch Generation

Anti Fuzzing

Current Research

Root Cause Analysis

Automated Exploit/Patch Generation

Anti Fuzzing

Extended Coverage

Current Research

Root Cause Analysis

Automated Exploit/Patch Generation

Anti Fuzzing

Extended Coverage

Taint Tracking

Current Research

Root Cause Analysis

Automated Exploit/Patch Generation

Anti Fuzzing

Extended Coverage

Taint Tracking

Symbolic Execution / SMT solvers

Questions?

Property based testing

Feedback Fuzzing

Tools

Grammar Fuzzing

Research