RUHR UNIVERSITÄT BOCHUM RUB

hgi SysSec

GRIMOIRE: Synthesizing Structure while Fuzzing

Usenix Security 2019, Santa Clara

August 16, 2019

Tim Blazytko, Cornelius Aschermann, Moritz Schlögel, Ali Abbasi,
Sergej Schumilo, Simon Wörner, and Thorsten Holz

Chair for Systems Security
Ruhr-Universität Bochum

Goal: Finding bugs in programs expecting structured input

Tiny C Compiler

libxml2

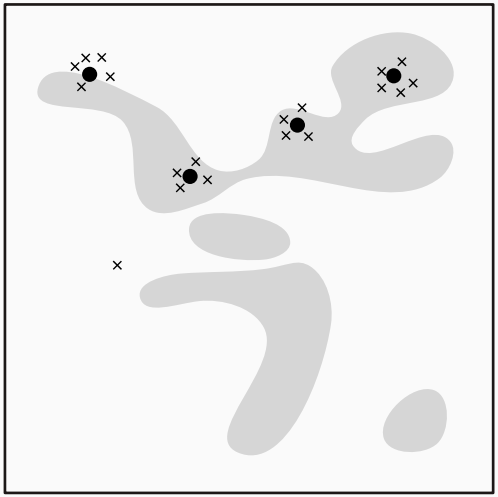
Boolector

JavaScriptCore

Let’s fuzz!

First attempt: Blind fuzzing

First attempt: Blind fuzzing



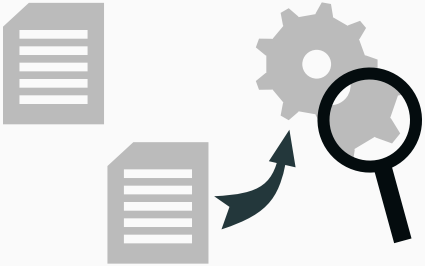
State space

- Interesting area
- Uninteresting area

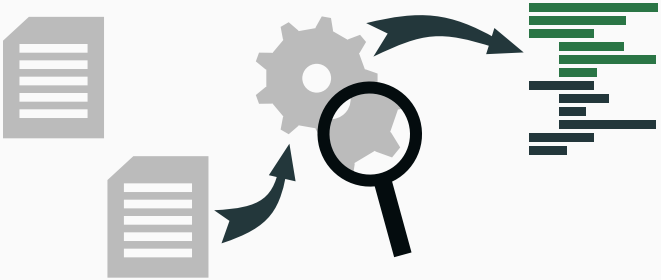
Can we do better?

Coverage-guided fuzzing

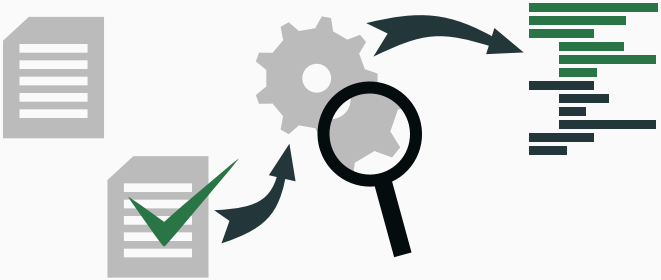
Program instrumentation



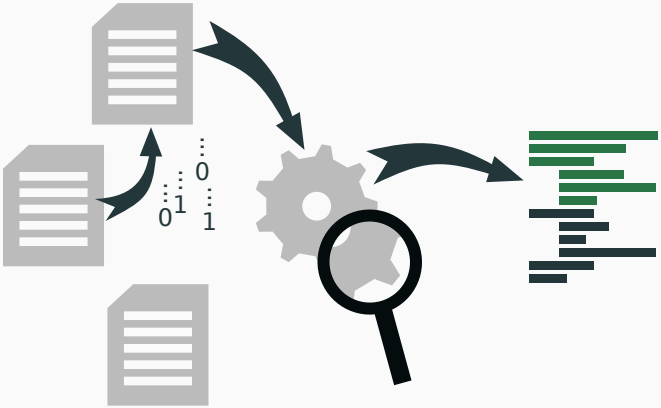
Coverage-guided fuzzing



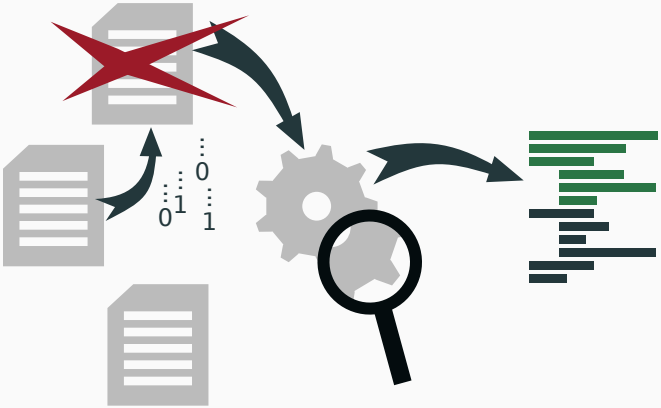
Coverage-guided fuzzing



Coverage-guided fuzzing



Coverage-guided fuzzing

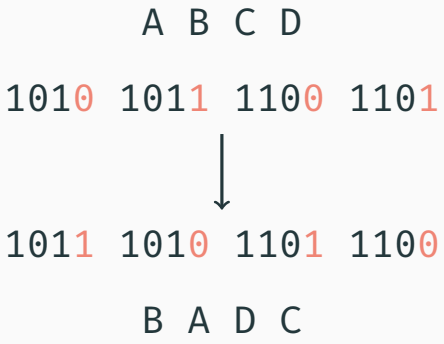


Small-scale mutations

- Bitflips

Small-scale mutations

- Bitflips



Small-scale mutations

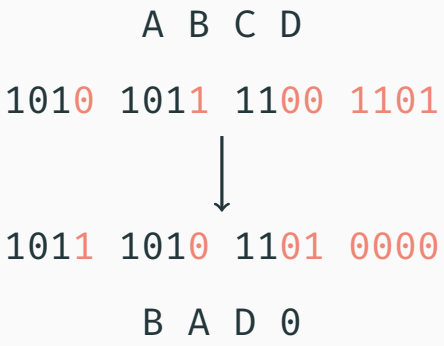
- Bitflips
- Simple arithmetic

Small-scale mutations

- Bitflips
- Simple arithmetic
- Force specific, “interesting” values

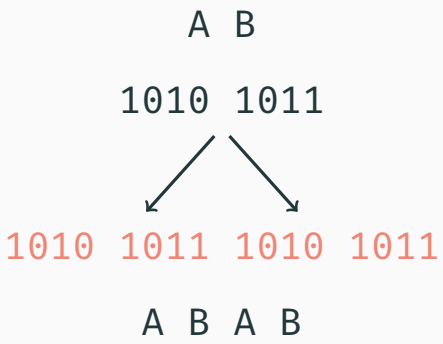
Small-scale mutations

- Bitflips
- Simple arithmetic
- Force specific, “interesting” values
- Havoc: “random” mutations



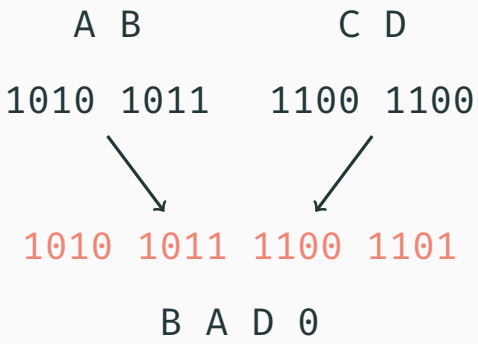
Small-scale mutations

- Bitflips
- Simple arithmetic
- Force specific, “interesting” values
- Havoc: “random” mutations
- Repetition



Small-scale mutations

- Bitflips
- Simple arithmetic
- Force specific, “interesting” values
- Havoc: “random” mutations
- Repetition
- Splicing



Small-scale mutations

Observation: Mutations modify the input only slightly

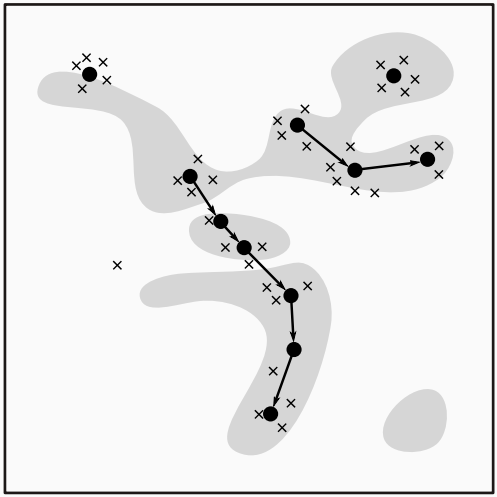
While this input works well ...

```
000000 050045 043104 030455 032456 022412
152714 152301 154305 000010 142320 005306
030061 030040 067440 065142 036012 020074
000020 052057 070171 020145 054057 061117
062552 072143 027440 000030 072523 072142
070171 020145 043057 071157 020155 043057
000040 071157 052155 070171 020145 020061
041057 067502 020170 000050 020133 020060
020060 030061 020060 030061 020060 020135
000060 046457 072141 064562 020170 020133
020061 020060 020060 000070 020061 020060
020060 020135 051057 071545 072557 061562
```

While this input works well ...

```
000001 050045 043104 030455 032456 022412
152714 152301 154305 000011 142320 005306
030061 030040 067440 065142 036092 020074
000021 052057 070171 020145 054057 061117
062552 072143 027440 000031 072523 072142
000000 020145 043057 071157 020155 043057
000041 071157 052155 070171 020145 020061
041057 067502 020170 000051 020133 020060
020060 030061 020060 000000 020060 020135
000061 046457 072141 064562 020170 020133
020061 020060 020060 000071 020061 020060
020060 020135 05105F 071545 072557 061562
```

Small-scale mutations



State space

- Interesting area
- Uninteresting area
- → Mutations (cov.-guided)

Small-scale mutations

Observation: Mutations modify the input only slightly

Caveat: Not all programs are equal

… this one is problematic

```
def some_function(self):
    s = "hi mom! "
    if self.famous:
        return s + "I'm famous!"
    else:
        self.confidence = 0
        return s + "*crying*"
```

… this one is problematic

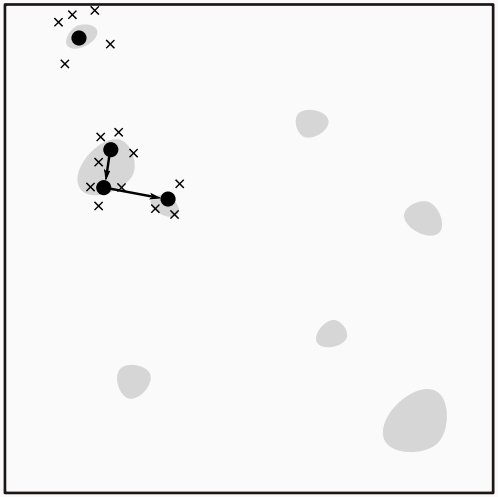
```
deb1some_functioasdflf):
    s = "hi mom! "
    if ? ?``famous:
        reABCDEFGH "I'm famous!"
    else:
        self.confidence = 0
        return s + 0000ying*"
```

… this one is problematic

```
deb1some_functioasdflf):
    s = "hi mom! "
        self.confidence = 0
        return s + 0000ying*"
```

Insight: Mutation requires input's structure

Small-scale mutations

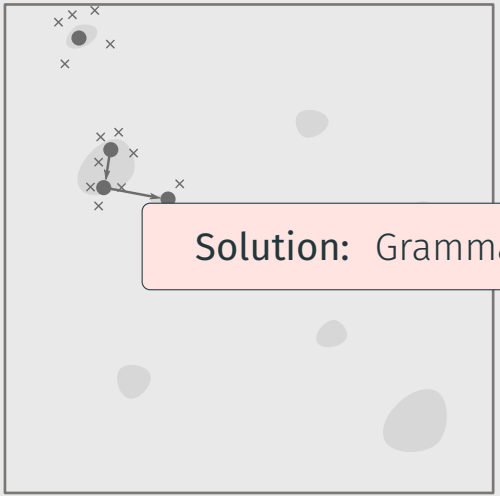


State space

- Interesting area
- Uninteresting area
- → Mutations (cov.-guided)

How to cross large gaps?

Small-scale mutations



Solution: Grammar-based fuzzing

State space

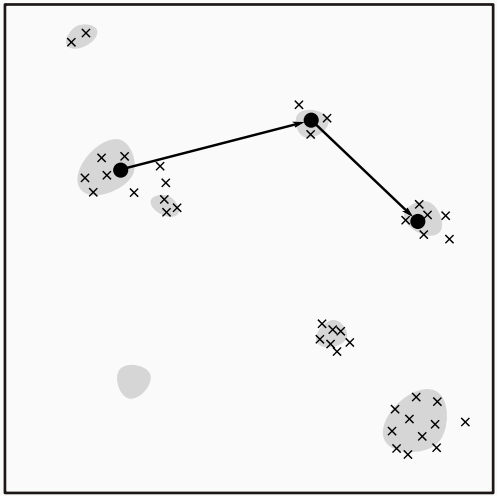
Interesting area

Uninteresting area

(cov.-guided)

How to cross large gaps?

Large-scale mutations

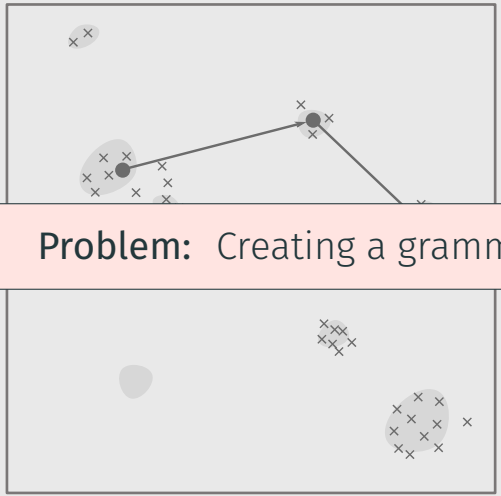


State space

- Interesting area
- Uninteresting area
- → Mutations (grammar)

Now crossing large gaps!

Large-scale mutations



State space

Interesting area

Uninteresting area

Problem: Creating a grammar requires human-effort

Now crossing large gaps!

Our approach

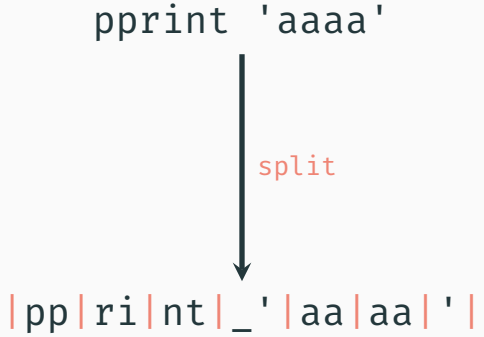
Grimoire: Best of both worlds

- Learn structure of inputs via fuzz testing
- Apply large-scale mutations on learned structures
- Profit!

Input generalization

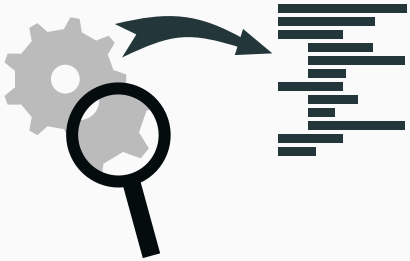
```
pprint 'aaaa'
```

Input generalization



Input generalization

```
pp|ri|nt|_'|aa|aa|'
```

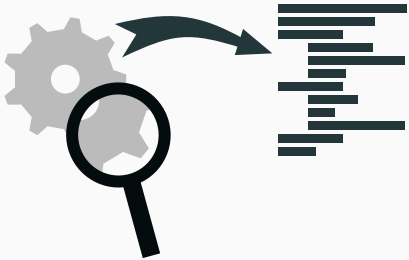


Input generalization

~~pp~~|ri|nt|_'|aa|aa|'

↓

rint 'aaaa'

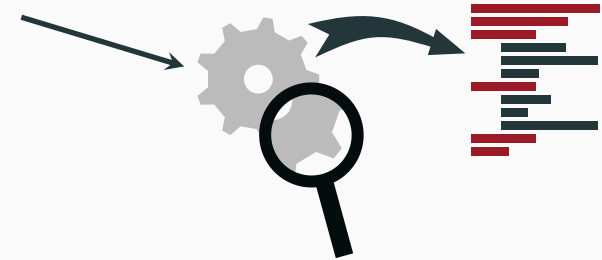


Input generalization

~~pp~~|ri|nt|_'|aa|aa|'

↓

rint 'aaaa'

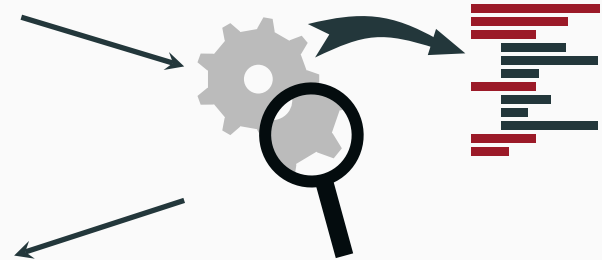


Input generalization

~~pp~~|ri|nt|_'|aa|aa|'

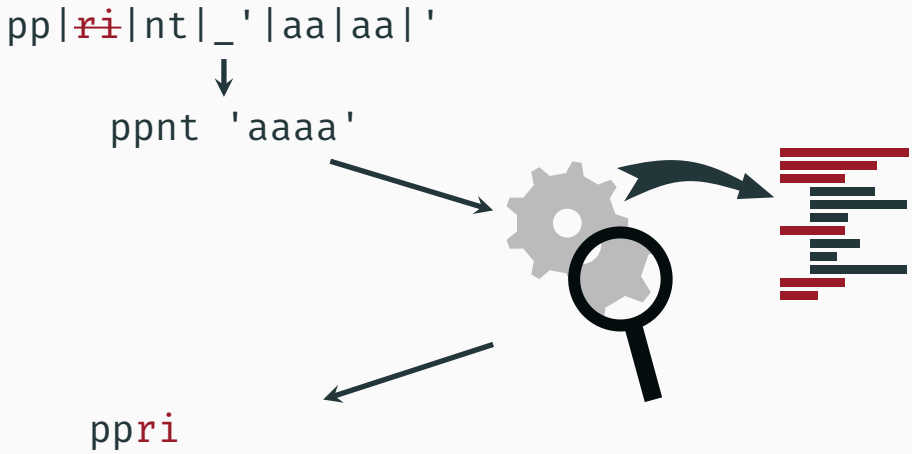
↓

rint 'aaaa'

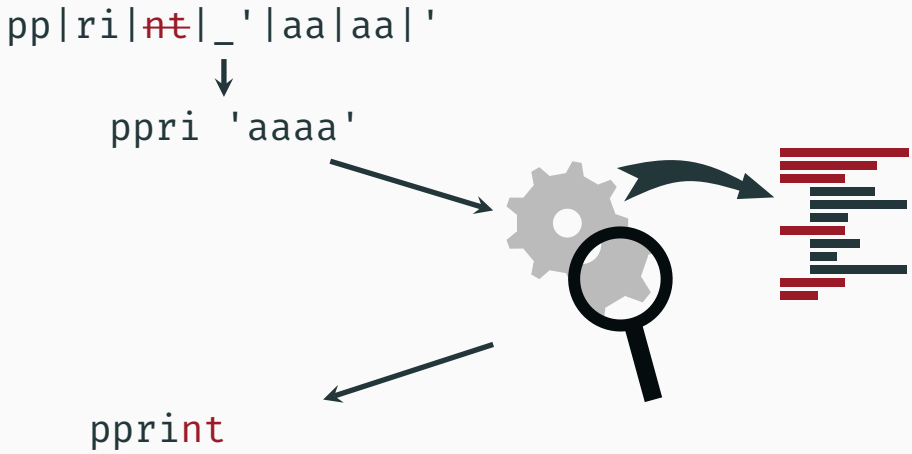


pp

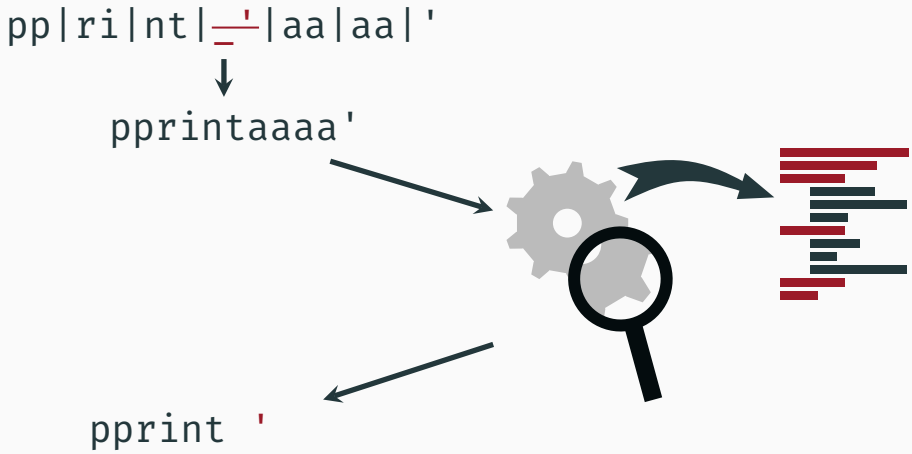
Input generalization



Input generalization



Input generalization



Input generalization

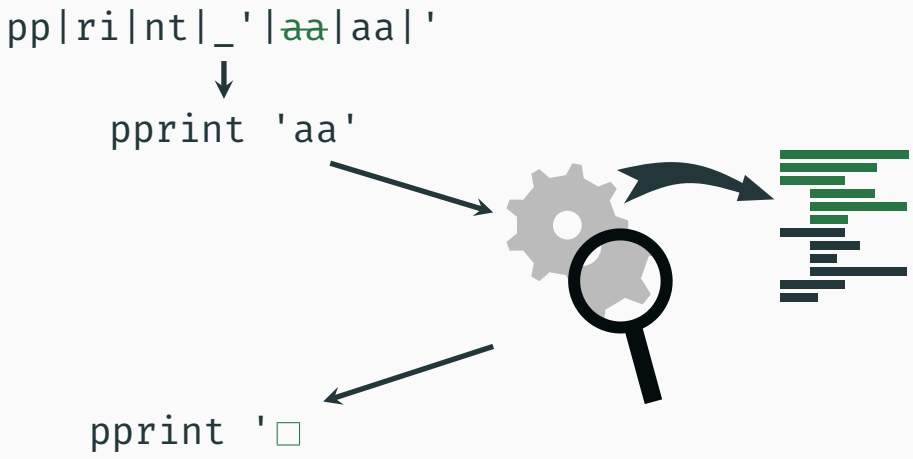
```
pp|ri|nt|_'|aa|aa|'
```

↓

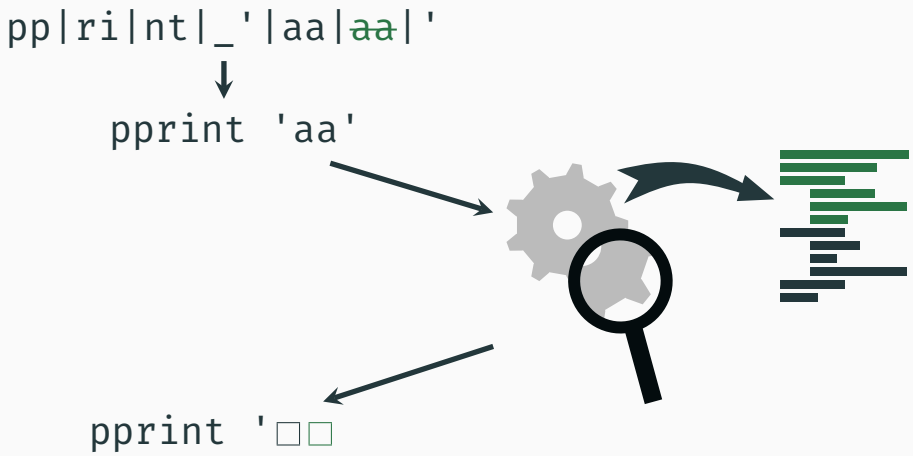
```
pprint 'aa'
```

```
pprint '
```

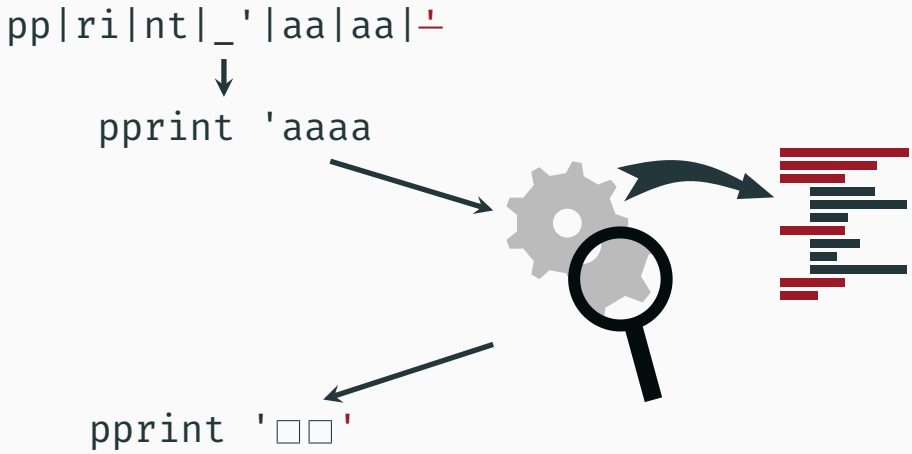
Input generalization



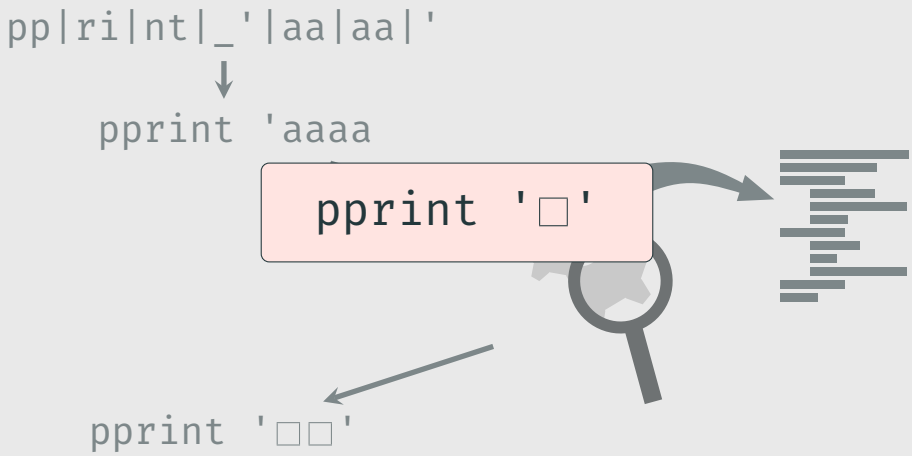
Input generalization



Input generalization



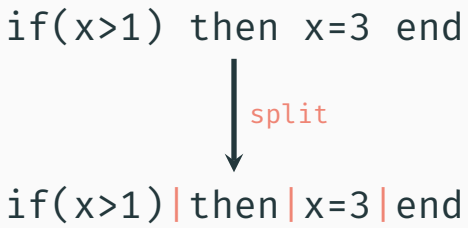
Input generalization



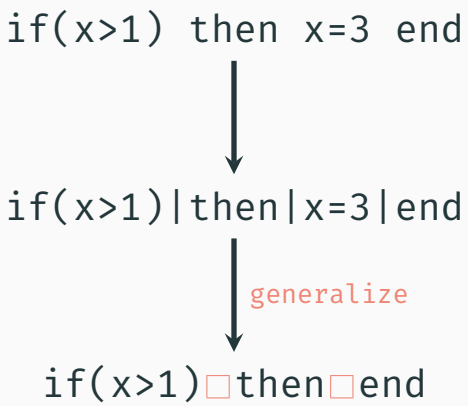
Input generalization

```
if(x>1) then x=3 end
```

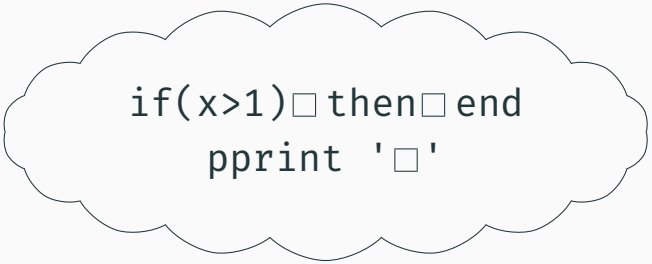
Input generalization



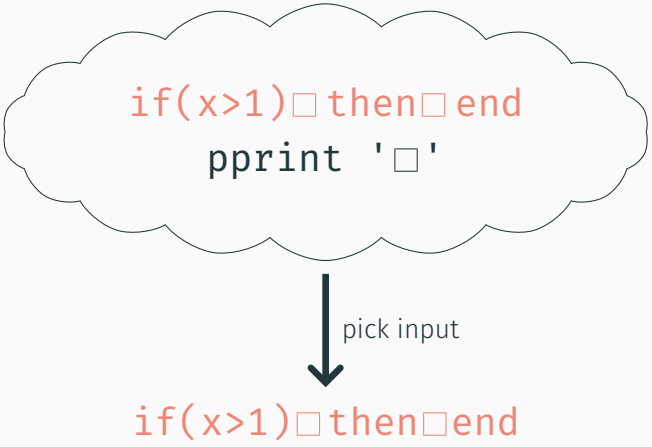
Input generalization



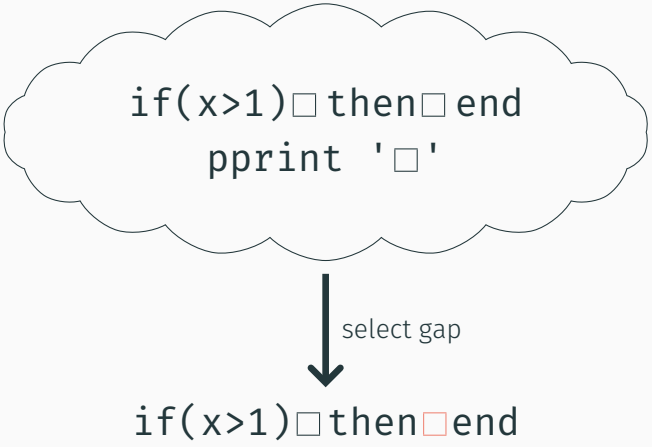
Why do we generalize inputs?



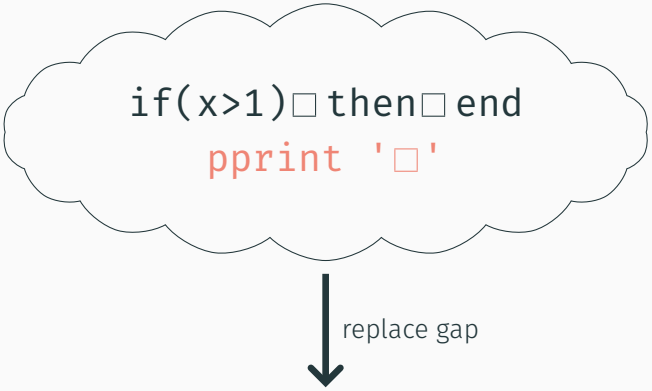
Why do we generalize inputs?



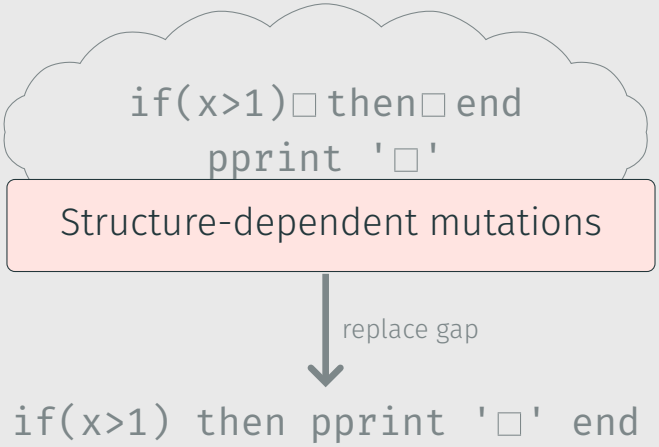
Why do we generalize inputs?



Why do we generalize inputs?



Why do we generalize inputs?

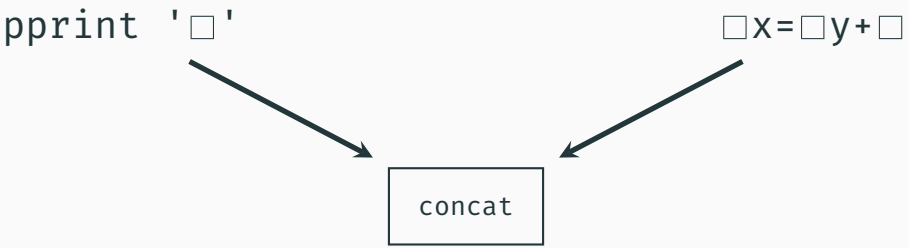


Input extension

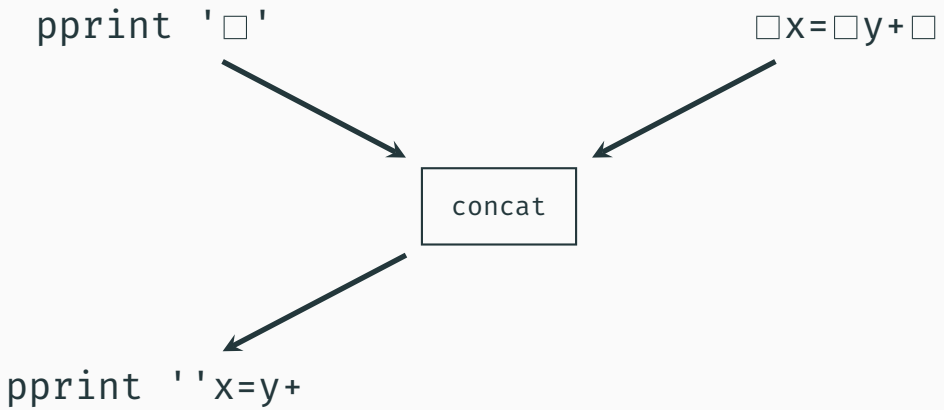
```
pprint '□'
```

```
□x=□y+□
```

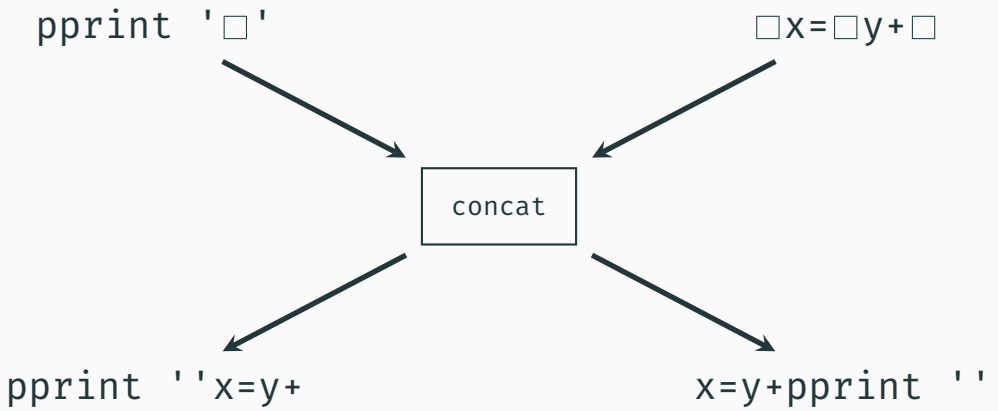
Input extension



Input extension



Input extension



Recursive replacement

```
pprint '□'    if(x>1)□then□end    □x=□y+□
```

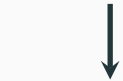
Recursive replacement

```
pprint '□'    if(x>1)□then□end    □x=□y+□
```

```
if(x>1)
```

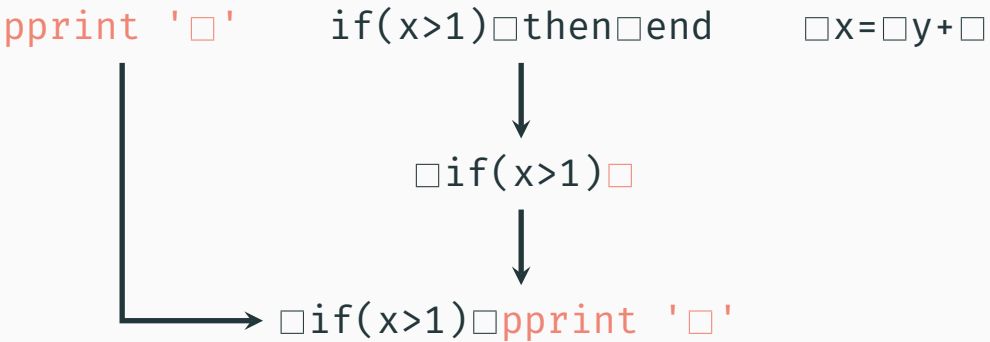
Recursive replacement

```
pprint '□'    if(x>1)□then□end    □x=□y+□
```

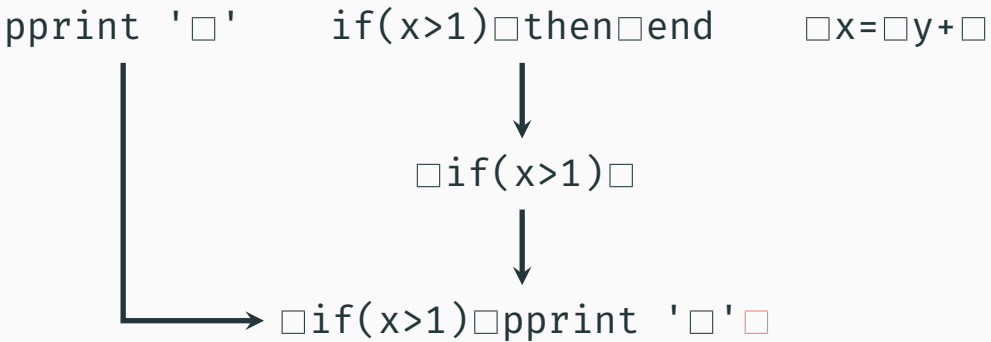


```
□if(x>1)□
```

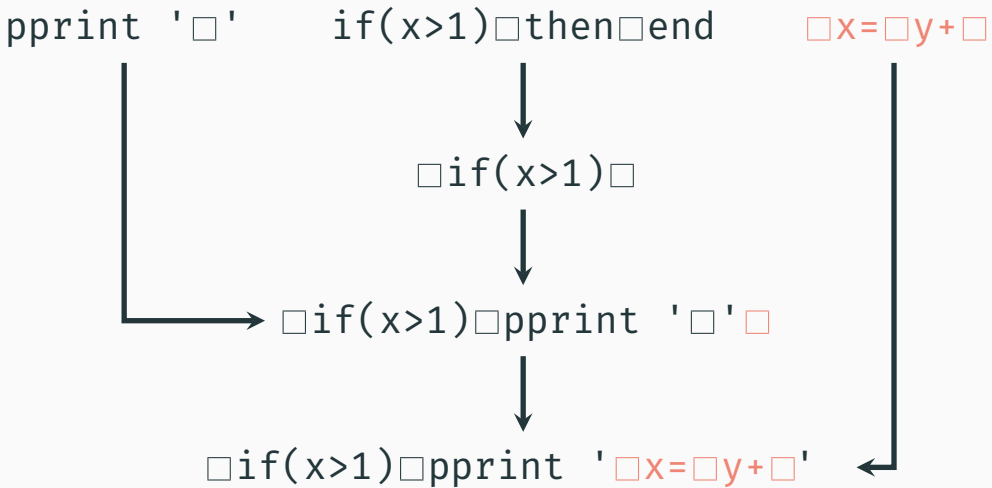
Recursive replacement



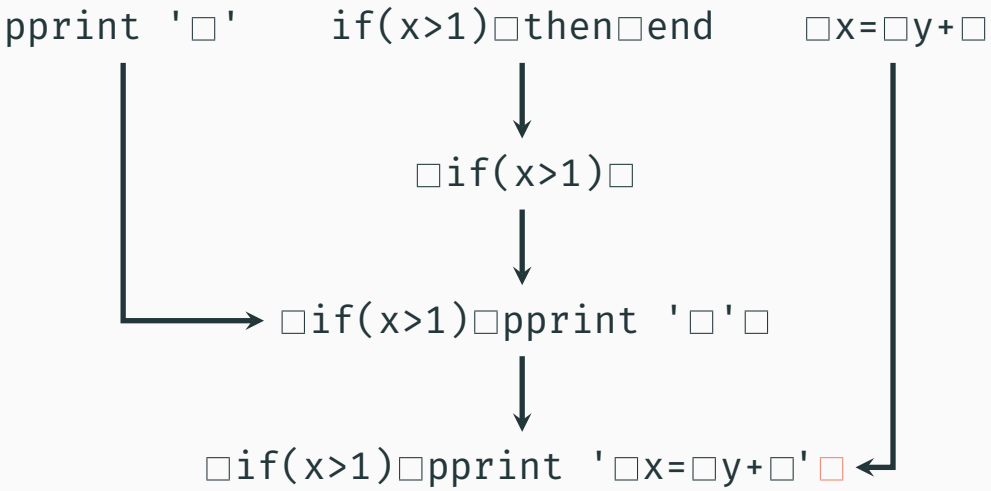
Recursive replacement



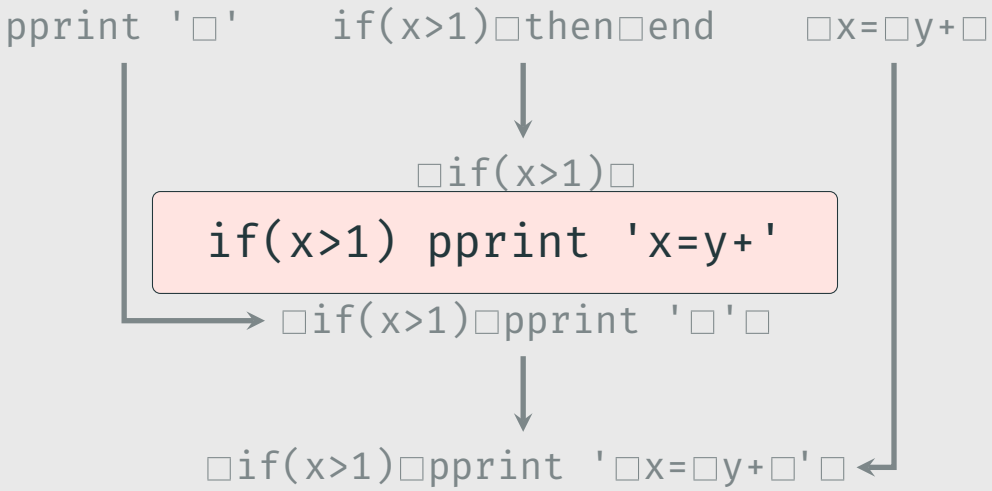
Recursive replacement



Recursive replacement



Recursive replacement

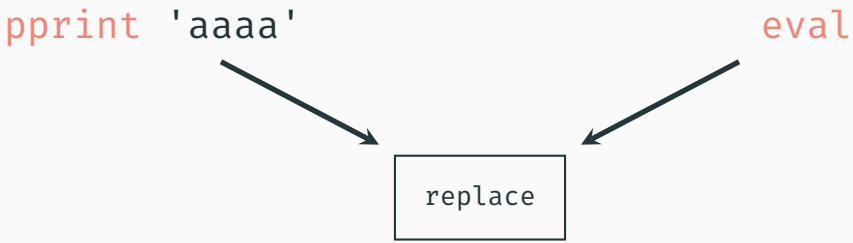


String replacement

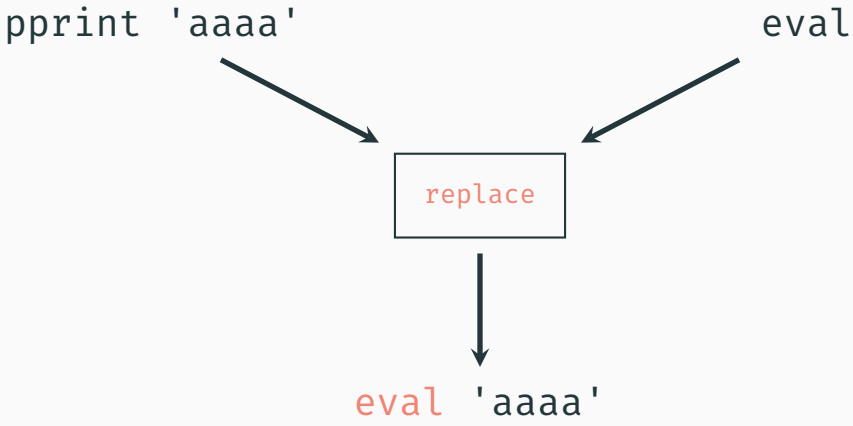
```
pprint 'aaaa'
```

```
eval
```

String replacement



String replacement



Evaluation

Common fuzzers vs. GRIMOIRE

We outperform AFL, QSYM, Angora, … on almost all targets

✓

Tiny C Compiler ✓

✓

mruby ✓

Boolector ✓

✓

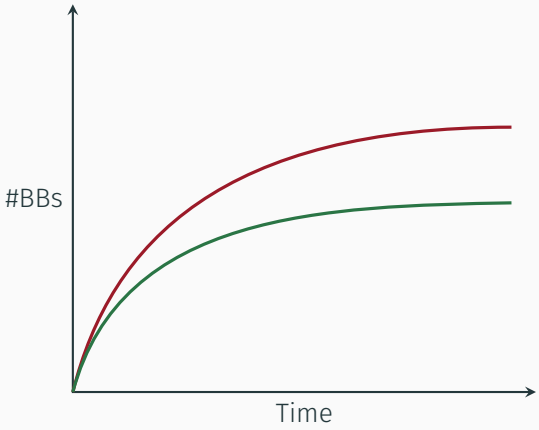
libxml2 ✓

✗

Evaluation

Grammar-based fuzzer vs. GRIMOIRE

Comparison to a grammar-based fuzzer

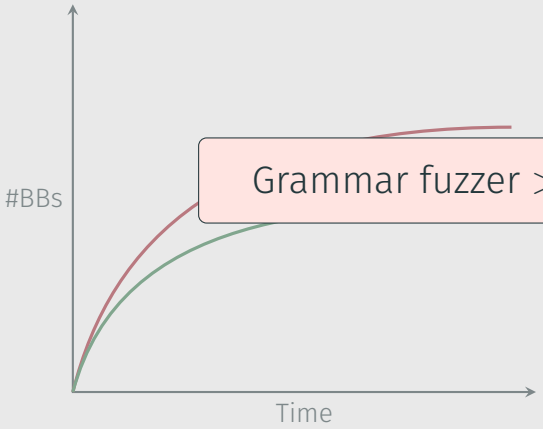


Legend

— Grammar fuzzer

— GRIMOIRE

Comparison to a grammar-based fuzzer



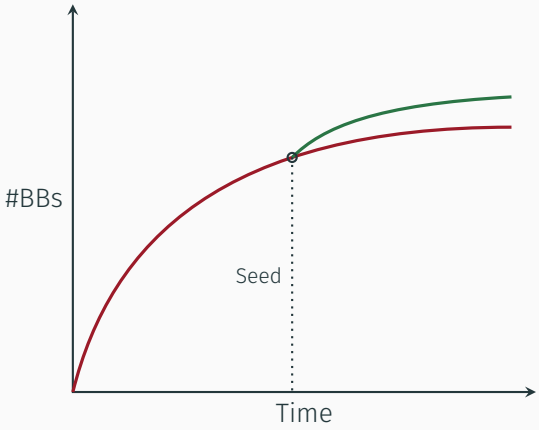
Grammar fuzzer >> GRIMOIRE

Legend

rammar fuzzer

GRIMOIRE

Using a grammar-based fuzzer as seed

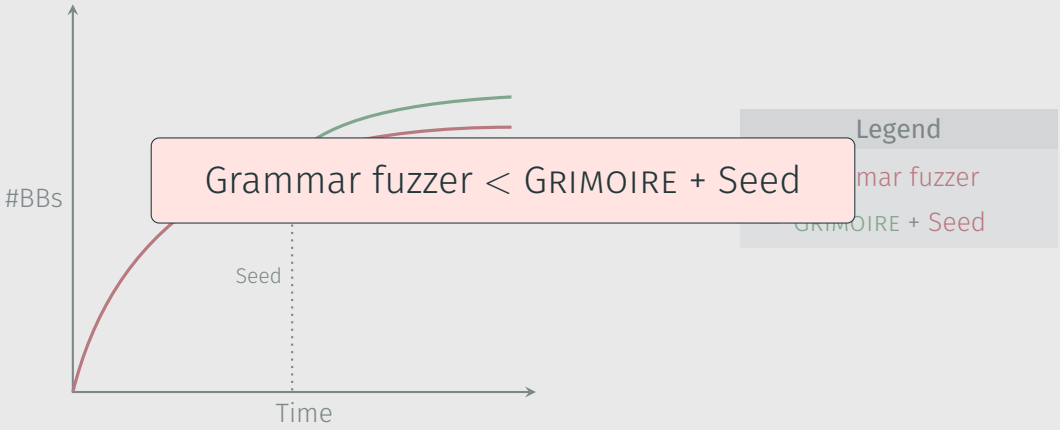


Legend

— Grammar fuzzer

— GRIMOIRE + Seed

Using a grammar-based fuzzer as seed



Conclusion

Take-aways

- Fuzzing structured inputs
- Common fuzzers: Small-scale mutations
- Grammar-based: Large-scale mutations
- GRIMOIRE:
  - Inference of input structure
  - Large-scale mutations (extension, recursive replacement, string replacement)
- Real-world impact: 11 CVEs assigned

Take-aways

- Fuzzing structured inputs
- Common fuzzers: Small-scale mutations
- Grammar-based: Lar
- GRIMOIRE:
  - Inference of input structure
  - Large-scale mutations (extension, recursive replacement, string replacement)
- Real-world impact: 11 CVEs assigned

Thank you!

Take-aways

- Fuzzing structured inputs
- Common fuzzers: Small-scale mutations
- Grammar-based: Large-scale mutations
- GRIMOIRE:
  - Inference of input structure
  - Large-scale mutations (extension, recursive replacement, string replacement)
- Real-world impact: 11 CVEs assigned

@m_u00d8

moritz.schloegel@rub.de