AURORA: Statistical Crash Analysis for Automated Root Cause Explanation

Tim Blazytko, Moritz Schlögel, Cornelius Aschermann, Ali Abbasi,
Joel Frank, Simon Wörner and Thorsten Holz

Let us look at some crash!

mruby

```
CASE(OP_GETUPVAR) {
  /* A B C  R(A) := uvget(B,C) */
  int a = GETARG_A(i);
  int b = GETARG_B(i);
  int c = GETARG_C(i);
  mrb_value *regs_a = regs + a;
  struct REnv *e = uvenv(mrb, c);
  if (!e) {
    *regs_a = mrb_nil_value();
  }
  else {
    *regs_a = e->stack[b];
  }
  NEXT;
}
```

```
CASE(OP_GETUPVAR) {
  /* A B C  R(A) := uvget(B,C) */
  int a = GETARG_A(i);
  int b = GETARG_B(i);
  int c = GETARG_C(i);
  mrb_value *regs_a = regs + a;
  struct REnv *e = uvenv(mrb, c);
  if (!e) {
    *regs_a = mrb_nil_value();
  }
  else {
    *regs_a = e->stack[b];
  }
  NEXT;
}
```

heap buffer overflow

```
CASE(OP_GETUPVAR) {
  /* A B C  R(A) := uvget(B,C) */
  int a = GETARG_A(i);
  int b = GETARG_B(i);
  int c = GETARG_C(i);
  mrb_value *regs_a = regs + a;
  struct REnv *e = uvenv(mrb, c);
  if (!e) {
    *regs_a = mrb_nil_value();
  }
  else {
    *regs_a = e->stack[b];
  }
  NEXT;
}
```

integer overflow

heap buffer overflow

How to find the root cause?

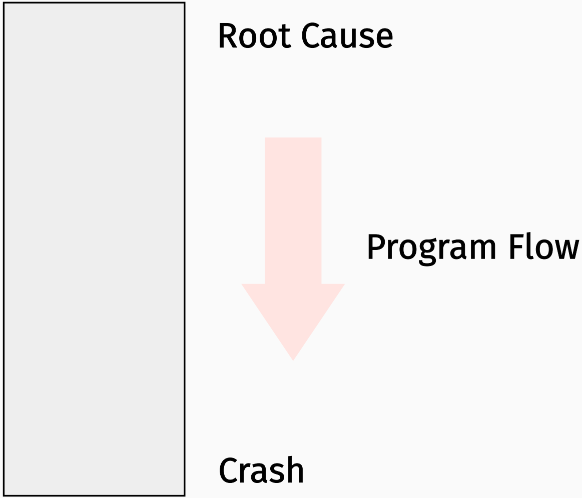
It starts at the crashing location

```
  0x55555556633d <kh_put_iv+19> mov QWORD PTR [rbp-0x30], rcx
  0x555555566341 <kh_put_iv+23> mov DWORD PTR [rbp-0x4], 0x0
  0x555555566348 <kh_put_iv+30> mov rax, QWORD PTR [rbp-0x20]
  0x55555556634c <kh_put_iv+34> mov edx, DWORD PTR [rax+0x8]
  0x55555556634f <kh_put_iv+37> mov rax, QWORD PTR [rbp-0x20]
  0x555555566353 <kh_put_iv+41> mov eax, DWORD PTR [rax]
  0x555555566355 <kh_put_iv+43> shr eax, 0x2
  0x555555566358 <kh_put_iv+46> mov ecx, eax
  0x55555556635a <kh_put_iv+48> mov rax, QWORD PTR [rbp-0x20]

Name: "mruby", stopped 0x55555556634c in kh_put_iv (), reason: SIGSEGV
```

Automated Approaches

Backward Taint Analysis and Reverse Execution

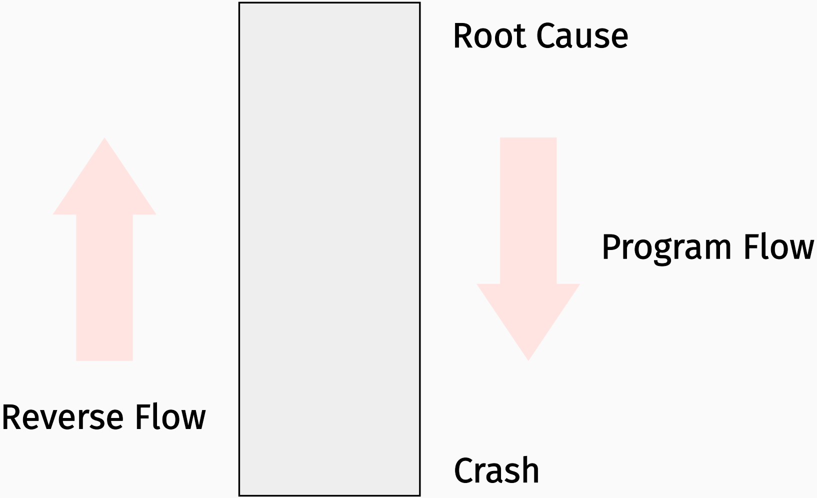


Root Cause

Program Flow

Crash

Backward Taint Analysis and Reverse Execution



What about this?

```
NotImplementedError = String
Module.constants
```

What about this?

exception type

```
NotImplementedError = String
Module.constants
```

What about this?

exception type string type

```
NotImplementedError = String
Module.constants
```

What about this?

exception type string type

```
NotImplementedError = String
Module.constants
```

raises exception of string type

What about this?

exception type string type

```
NotImplementedError = String
Module.constants
```

raises exception of string type

type confusion

What about this?

exception type string type

No direct data flow between crash site and root cause

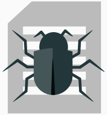
```
NotImplementedError = String
Module.constants
```

raises exception of string type

type confusion

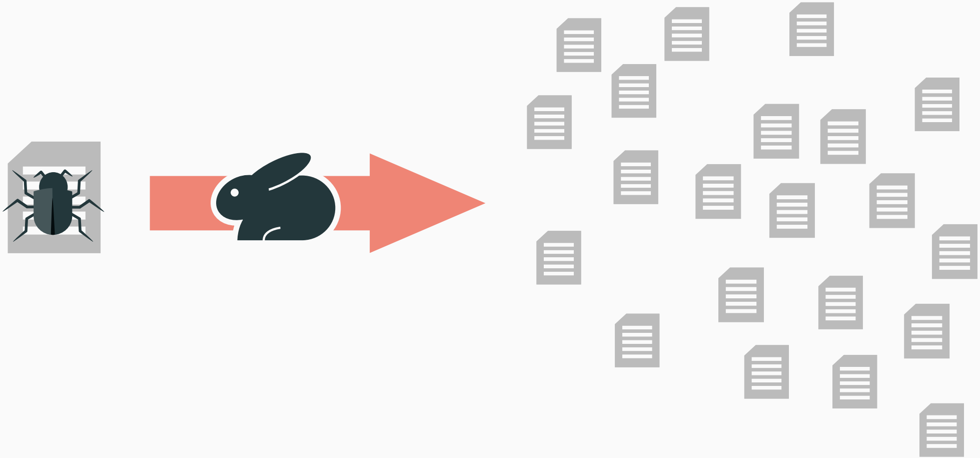
Our Approach

Root Cause Analysis

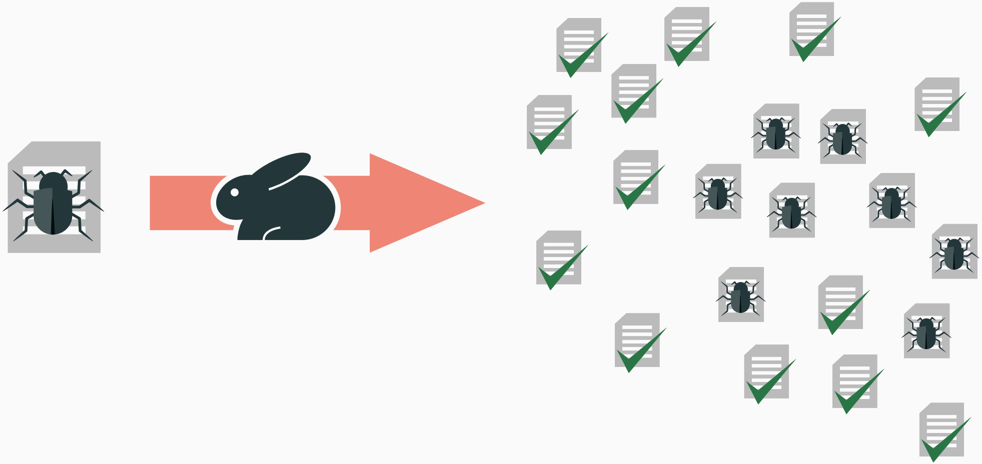


Root Cause Analysis

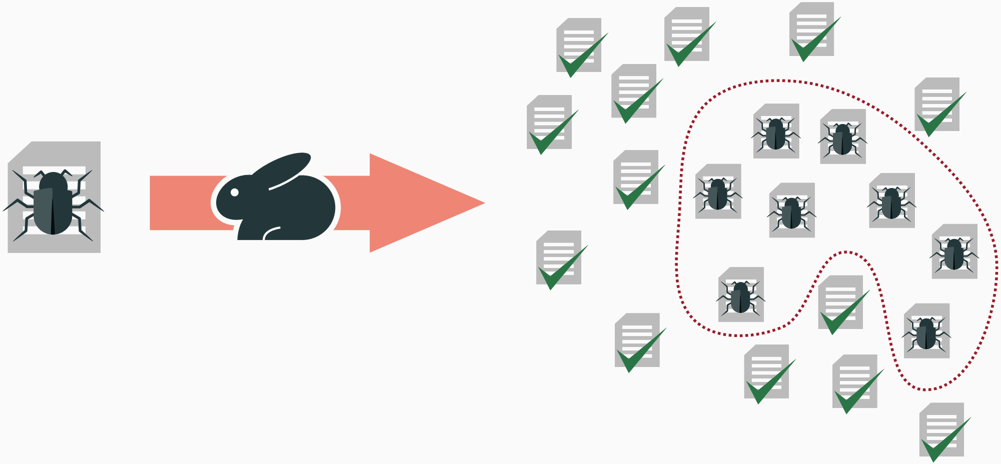
Root Cause Analysis



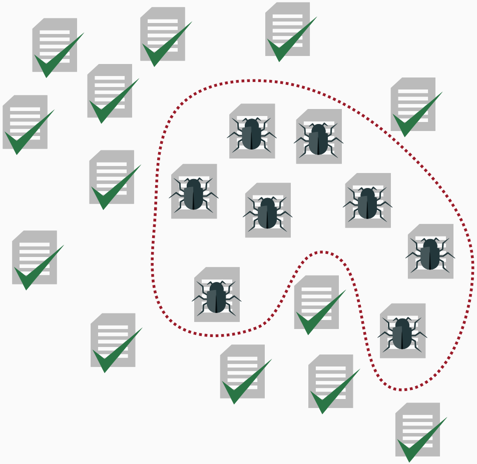
Root Cause Analysis



Root Cause Analysis



Root Cause Analysis



`val.type < 17`

Root Cause Analysis

```
MRB_TT_STRING     / * 16 * /
MRB_TT_RANGE      / * 17 * /
MRB_TT_EXCEPTION  / * 18 * /
```

```
val.type < 17
```

Root Cause Analysis

```
MRB_TT_STRING     / * 16 * /
MRB_TT_RANGE      / * 17 * /
MRB_TT_EXCEPTION  / * 18 * /
```

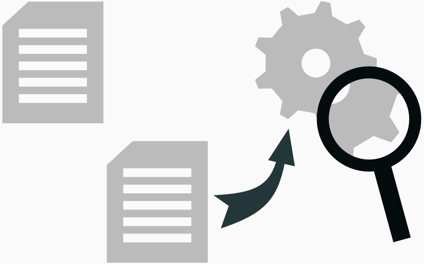
```
val.type ≠ MRB_TT_EXCEPTION
```

```
val.type < 17
```

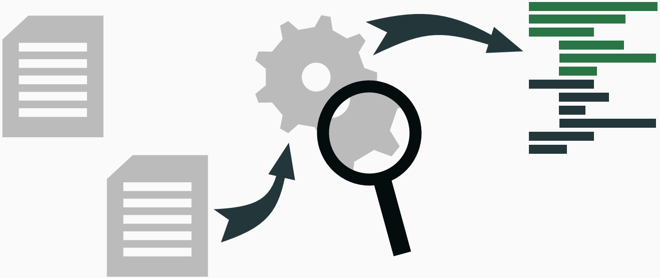
Crash Exploration

Coverage-guided Fuzzing

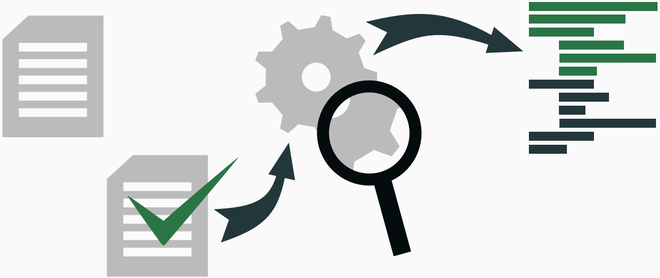
Program instrumentation



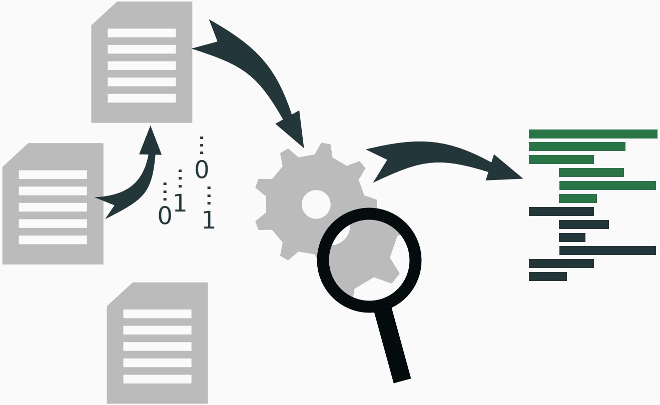
Coverage-guided Fuzzing



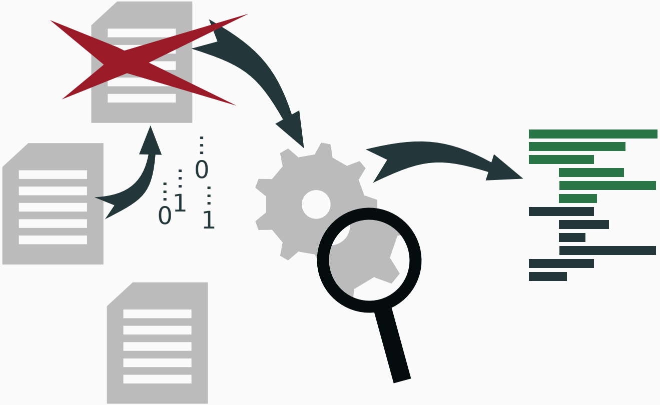
Coverage-guided Fuzzing



Coverage-guided Fuzzing



Coverage-guided Fuzzing



Collect Trace Information

Register and Memory Writes

```
  add rax, rbx
  mov rbx, 0x20
  add rcx, 0x10
  jnz exit

  mov rax, rbx

exit:
  add rax, 0x1
  ret
```

Register and Memory Writes

```
  add rax, rbx
  mov rbx, 0x20
  add rcx, 0x10
  jnz exit

  mov rax, rbx

exit:
  add rax, 0x1
  ret
```

Register and Memory Writes

```
  add rax, rbx      min: 0x0          max: 0x50
  mov rbx, 0x20     min: 0x20         max: 0x20
  add rcx, 0x10     min: 0x100        max: 0x10000
  jnz exit

  mov rax, rbx      min: 0x0          max: 0x1342

exit:
  add rax, 0x1      min: 0x0          max: 0x1343
  ret               min: 0x400546     max: 0x403142
```

Control-flow Edges

```
  add rax, rbx     |   min: 0x0        max: 0x50
  mov rbx, 0x20    |   min: 0x20       max: 0x20
  add rcx, 0x10    |   min: 0x100      max: 0x10000
  jnz exit         |   jmp taken to exit 4 times
                   |
  mov rax, rbx     |   min: 0x0        max: 0x1342
                   |
exit:              |
  add rax, 0x1     |   min: 0x0        max: 0x1343
  ret              |   min: 0x400546   max: 0x403142
```

Predicate Synthesis

Find the best value to distinguish crashes from non-crashes

`outcome`	crash	crash	non-crash	non-crash
`val.type`	16	16	18	18

Find the best value to distinguish crashes from non-crashes

`outcome`	crash	crash	non-crash	non-crash
`val.type`	16	16	18	18

Find the best value to distinguish crashes from non-crashes

outcome	crash	crash	non-crash	non-crash
val.type	16	16	18	18

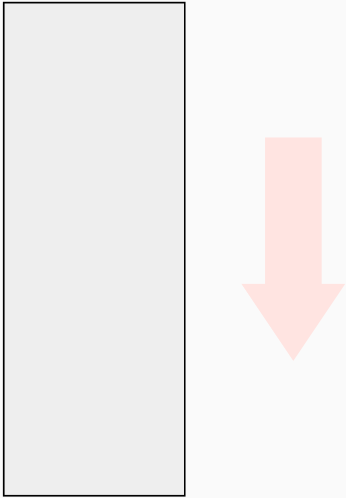
`val.type < 17`

Predicate Types

- control-flow edges
- `r < c` for register and memory values
- `is_heap_ptr(r)`
- `is_stack_ptr(r)`
- flags

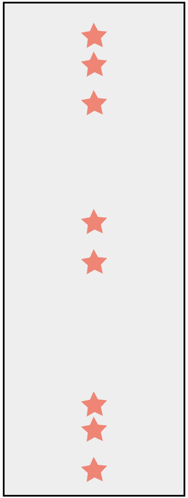
Predicate Ranking

Execution Order

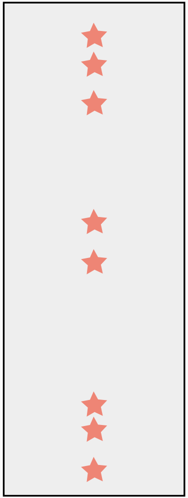


Program Flow

Execution Order



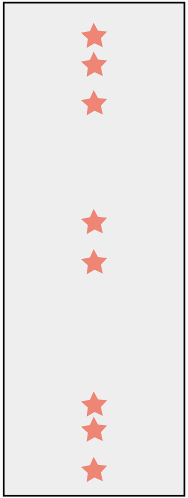
Execution Order



Root Cause

Crash

Execution Order



Root Cause

Propagation

Crash

Evaluation

```
CASE(OP_GETUPVAR) {
  /* A B C  R(A) := uvget(B,C) */
  int a = GETARG_A(i);
  int b = GETARG_B(i);
  int c = GETARG_C(i);
  mrb_value *regs_a = regs + a;
  struct REnv *e = uvenv(mrb, c);
  if (!e) {
    *regs_a = mrb_nil_value();
  }
  else {
    *regs_a = e->stack[b];
  }
  NEXT;
}
```

integer overflow

heap buffer overflow

```
CASE(OP_GETUPVAR) {
  /* A B C  R(A) := uvget(B,C) */
  int a = GETARG_A(i);
  int b = GETARG_B(i);
  int c = GETARG_C(i);
  mrb_value *regs_a = regs + a;
  struct REnv *e = uvenv(mrb, c);
  if (!e) {
    *regs_a = mrb_nil_v
  }
  else {
    *regs_a = e->stack[b];
  }
  NEXT;
}
```

`rbx < 0xff`

integer overflow

heap buffer overflow

Targets

nasm the netwide assembler

BASH THE BOURNE-AGAIN SHELL

Sleuthkit

mruby

Xpdf

php

Lua

libzip

screen

TCPDUMP & LiBPCAP

python™

patch

Bug Classes

- type confusion (`Python` and `mruby`)
- use–after-free (`Lua`, `mruby`, ...)
- uninitialized variable (`PHP`, `mruby`)
- heap buffer overflow (`Perl`, `Lua`, ...)
- null pointer dereference, stack-based buffer overflow, ...

Bug Classes

- type confusion (`Python` and `mruby`)
- use–after-free (`Lua`, `mruby`, ...)
- uninitialized variable (`PHP`, `mruby`)

Up to 28, 289, 736 instructions between root cause and crash

- heap buffer overflow (`Perl`, `Lua`, ...)
- null pointer dereference, stack-based buffer overflow, ...

Conclusion

Conclusion

- automated root cause analysis for complex bugs
- find related inputs for a given crash
- collect trace information
- **distinguish** crashing from non-crashing behavior via statistical analysis
- bug classes: type confusion, use-after-free, heap buffer overflow, ...

Thank You!

tim.blazytko@rub.de

@mr_phrazer

`https://github.com/RUB-SysSec/aurora`