A Pointer Tracking Memory Model for KLEE

Felix Rath, Daniel Schemmel, Oscar Soria Dustmann, Klaus Wehrle

https://comsys.rwth-aachen.de
Motivation

```c
int main(void) {
    int a[3] = {1, 2, 3};
    int s = klee_int("s");

    if(a[s] == 2)
        return 1;
    else
        return 0;
}
```
```c
int main(void) {
    int a[3] = {1, 2, 3};
    int s = klee_int("s");
    if(a[s] == 2)
        return 1;
    else
        return 0;
}
```
```c
int main(void) {
    int a[3] = {1, 2, 3};
    int s = klee_int("s");
    if(a[s] == 2)
        return 1;
    else
        return 0;
}
```
```c
int main(void) {
    int a[3] = {1, 2, 3};
    int s = klee_int("s");
    if(a[s] == 2)
        return 1;
    else
        return 0;
}
```

---

**Motivation**

```c
int a[3] = 1, 2, 3;
int s = klee_int("s");
if(a[s] == 2)
    0 \leq s \leq 2
\neg(0 \leq s \leq 2)
```

Out-of-bounds
int main(void) {
  int a[3] = {1, 2, 3};
  int s = klee_int("s");
  if(a[s] == 2)
    return 1;
  else
    return 0;
}

int a[3] = 1, 2, 3;
int s = klee_int("s");

if(a[s] == 2)
  return 1;
else
  return 0;

Out-of-bounds

s = 1
¬(s = 1)
Motivation

```c
int main(void) {
    int a[3] = {1, 2, 3};
    int s = klee_int("s");
    if(a[s] == 2) return 1;
    else return 0;
}
```

Expected: 3 Test Cases

Actual: 17 Test Cases
Resolving $a[s]$
Resolving \( a[s] \)

\[
a[s] = *(a+s)
\]
Resolving a[s]

\[ a[s] = *(a+s) \]

\[ a = \text{ConstantExpr 0x3a28213a} + \text{ReadExpr w32 s} = s \]
Resolving $a[s]$

$$a[s] = *(a+s)$$

$$a = \text{ConstantExpr } 0x3a28213a + \text{ReadExpr w32 } s = s$$

0x0  a  0x..
Resolving $a[s]$ 

$a[s] = *(a+s)$

$$a = \text{ConstantExpr 0x3a28213a} + \text{ReadExpr w32 s} = s$$
Idea:

- Mark the return value of an allocation as a pointer: KnownPointer
- Also remember which object is pointed to
- Use this information for address resolution
Idea:

- Mark the return value of an allocation as a pointer: KnownPointer
- Also remember which object is pointed to
- Use this information for address resolution
Idea:

- Mark the return value of an allocation as a pointer: KnownPointer
- Also remember which object is pointed to
- Use this information for address resolution
Idea:

- Mark the return value of an allocation as a pointer: KnownPointer
- Also remember which object is pointed to
- Use this information for address resolution
Idea:

- Mark the return value of an allocation as a pointer: KnownPointer
- Also remember which object is pointed to
- Use this information for address resolution
What is a bug?

Operations on pointers

- `KnownPointer(a, 0x3a282130) + KnownValue(5)`?
What is a bug?

Operations on pointers

- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)
What is a bug?

Operations on pointers
- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

Pointer subtractions of different objects
- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
### What is a bug?

#### Operations on pointers
- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

#### Pointer subtractions of different objects
- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
- Undefined behaviour according to C standard
What is a bug?

Operations on pointers

- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

Pointer subtractions of different objects

- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
- Undefined behaviour according to C standard
- x86_64 does not care anymore
What is a bug?

Operations on pointers

- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

Pointer subtractions of different objects

- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

GCC 8 will be adding similar analyses
- -fsanitize=pointer-subtract, -fsanitize=pointer-compare
What is a bug?

Operations on pointers

- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

Pointer subtractions of different objects

- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

- GCC 8 will be adding similar analyses
## What is a bug?

### Operations on pointers

- KnownPointer(a, 0x3a282130) + KnownValue(5)?
- Result: KnownPointer(a, 0x3a282135)

### Pointer subtractions of different objects

- KnownPointer(a, 0x..) - KnownPointer(b, 0x..)
- Undefined behaviour according to C standard
- x86_64 does not care anymore
- Is this a bug that we want to detect?

### GCC 8 will be adding similar analyses

- -fsanitize=pointer-subtract, -fsanitize=pointer-compare
But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
Offsets and Bitops

But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
Offsets and Bitops

But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
- Typical case of a length/offset calculation

Result: KnownValue(5)

BUT

- KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)?
- (&, |, ~, <<, >>, %, *, /)
- Actually appear in real code (e.g., xor-pointer swap)
- KnownPointer and KnownValue not expressive enough
But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
- Typical case of a length/offset calculation
- Result: KnownValue(5)
But..

- `KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)`?
- What is the result?
- Is this still a `KnownPointer`?
- Typical case of a length/offset calculation
  
  Result: `KnownValue(5)`

BUT

- `KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)`?
Offsets and Bitops

But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
- Typical case of a length/offset calculation
- Result: KnownValue(5)

BUT

- KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)?
- (&, |, ~, <<, >>, %, *, /)
- Actually appear in real code (e.g., xor-pointer swap)
- KnownPointer and KnownValue not expressive enough
Offsets and Bitops

But..
- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
- Typical case of a length/offset calculation
- Result: KnownValue(5)

BUT
- KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)?
- (&, |, ~, <<, >>, %, *, /)?
- Actually appear in real code (e.g., xor-pointer swap)
Offsets and Bitops

But..

- KnownPointer(a, 0x3a282135) - KnownPointer(a, 0x3a282130)?
- What is the result?
- Is this still a KnownPointer?
- Typical case of a length/offset calculation
- Result: KnownValue(5)

BUT

- KnownPointer(a, 0x3a282135) ^ KnownPointer(a, 0x3a282130)?
- (&, |, ~, <<, >>, %, *, /)?
- Actually appear in real code (e.g., xor-pointer swap)
- KnownPointer and KnownValue not expressive enough
Insight

- We can’t always know if the result of an operation will be a pointer.
- Might only become obvious after subsequent operations.
Insight

- We can’t always know if the result of an operation will be a pointer.
- Might only become obvious after subsequent operations.

Solution:

```
MaybePointer
0x3a282130
Objects: a, b
```
MaybePointer

Insight

• We can’t always know if the result of an operation will be a pointer.
• Might only become obvious after subsequent operations.

Solution:

MaybePointer
0x3a282130
Objects: a, b

• Remember information about all involved objects
• No clear association with one specific object
### External Function Calls

- Executed natively

Challenge: We can't track pointer information.

Solution:
- After each external call check all changed memory locations.
- If something was a pointer before, try to resolve it.
- But: Might not be possible, or concrete values are now pointers.
- More overapproximation needed.
External Function Calls

- Executed natively
- Outside of KLEE

Solution
- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory

Challenge: We can't track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers

More overapproximation needed
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed

→ ConstantExpr

Felix Rath
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed
External Function Calls

- Executed natively
- Outside of KLEE
- Opaque operations that can change memory
- Challenge: We can’t track pointer information

Solution

- After each external call check all changed memory locations
- If something was a pointer before, try to resolve it
- But: Might not be possible, or concrete values are now pointers
- More overapproximation needed
- $\rightarrow$ ConstantExpr
KnownPointer(Object)

A value that is known to point to Object.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KnownPointer(Object)</td>
<td>A value that is known to point to Object.</td>
</tr>
<tr>
<td>MaybePointer(ObjectA, ObjectB, ...)</td>
<td>A value that was somehow created through operations on pointers into ObjectA and ObjectB, and might currently point into any of them or not.</td>
</tr>
</tbody>
</table>
KnownPointer(Object)
A value that is known to point to Object.

MaybePointer(ObjectA, ObjectB, ...)
A value that was somehow created through operations on pointers into ObjectA and ObjectB, and might currently point into any of them or not.

KnownValue
A value that is known to not be a pointer.
**Pointer Tracking: Summary**

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KnownPointer</strong>(Object)</td>
<td>A value that is known to point to <code>Object</code>.</td>
</tr>
<tr>
<td><strong>MaybePointer</strong>(ObjectA, ObjectB, ...)</td>
<td>A value that was somehow created through operations on pointers into <code>ObjectA</code> and <code>ObjectB</code>, and might currently point into any of them or not.</td>
</tr>
<tr>
<td><strong>KnownValue</strong></td>
<td>A value that is known to <strong>not</strong> be a pointer.</td>
</tr>
<tr>
<td><strong>ConstantExpr</strong></td>
<td>A value that might be a pointer into any object, or might not be a pointer at all. Results from external function calls.</td>
</tr>
</tbody>
</table>
Old Memory Model

Before: Single address space, containing all objects
Old Memory Model

Before: Single address space, containing all objects

- Address resolution based solely on values
Old Memory Model

Before: Single address space, containing all objects

- Address resolution based solely on values
- No idea which values are pointers
New Memory Model

Now: One full-size address space per object
New Memory Model

Now: One full-size address space per object

- Address resolution based on tracked pointers
New Memory Model

Now: One full-size address space per object

- Address resolution based on tracked pointers
- Much better knowledge about pointers and pointees
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s")) → Error: concretized symbolic size`
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s")) → Error: concretized symbolic size`

Possible solution: Lazy growth
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s")) → Error: concretized symbolic size`

Possible solution: Lazy growth
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s")) → Error: concretized symbolic size`

Possible solution: Lazy growth
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → Error: concretized symbolic size

Possible solution: Lazy growth
Symbolic Allocation Sizes

Currently only concretized in KLEE:

- `malloc(klee_size_t("s"))` → Error: concretized symbolic size

Possible solution: Lazy growth
External Function Calls revisited

EFCs require us to flatten our address space:
EFCs require us to flatten our address space:
External Function Calls revisited

EFCs require us to flatten our address space:

- “Problematic”
EFCs require us to flatten our address space:

- “Problematic”
- Quick fix: Re-allocate to a new position
EFCs require us to flatten our address space:

- “Problematic”
- Quick fix: Re-allocate to a new position
- Does not work either ("changing the past")
External Function Calls revisited

EFCs require us to flatten our address space:

- “Problematic”
- Quick fix: Re-allocate to a new position
- Does not easily work either (“changing the past”)
First Experiments

- 25 iterations, one hour each
- Able to run all 107 coreutils
- 80 without ConstantExprs
- No MaybePointers with more than four objects
- Four subtractions found, all due to one realloc in uclibc
- Somewhat high overhead (unoptimized)
1. UCHAR_T* old_buffer = cur_buffer;
2. cur_buffer = realloc(cur_buffer, new_size);
3. // ...
4. if(cur_buffer != old_buffer) {
5.   int offset = cur_buffer - old_buffer;
6.   FIXUP_POINTER(foo, offset);
7.   FIXUP_POINTER(bar, offset);
8.   // ...

KLEE’s memory model:

- Single address space
- No knowledge about pointers
Summary

KLEE’s memory model:
- Single address space
- No knowledge about pointers

Pointer tracking:
- Keep track of pointers
- Some operations and external calls require overapproximation
- KnownPointer, MaybePointer, KnownValue, ConstantExpr
- Basic symbolic allocation sizes
Summary

KLEE’s memory model:
- Single address space
- No knowledge about pointers

Pointer tracking:
- Keep track of pointers
- Some operations and external calls require overapproximation
- KnownPointer, MaybePointer, KnownValue, ConstantExpr
- Basic symbolic allocation sizes

Evaluation:
- Works on the coreutils
- Found one case of an illegal pointer subtraction