TeaDsa: Type-aware DSA-style Pointer Analysis for Low Level Code

Jakub Kuderski, Nhâm Lê, Arie Gurfinkel (UWaterloo); Jorge Navas (SRI International)

fmcad 2018, Austin, TX

Detecting Field Overflow Memory Safety Bugs

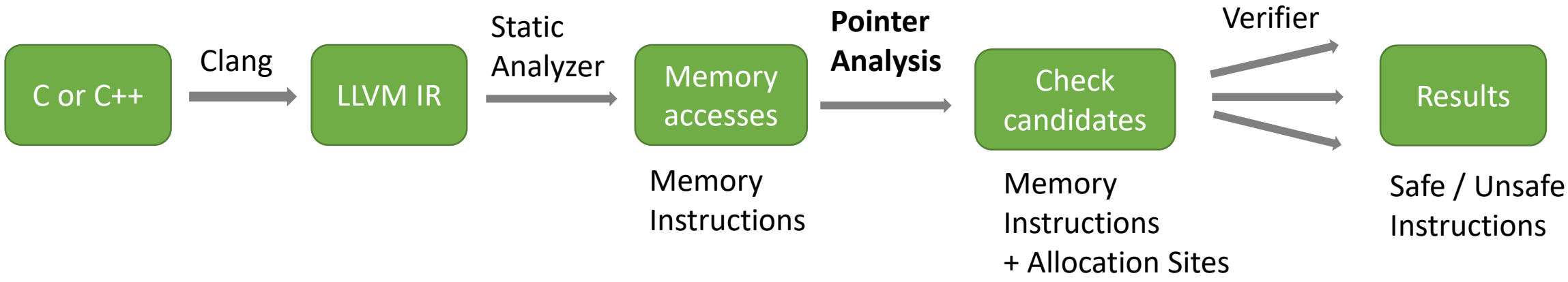
```
struct Node { Node *next = nullptr; int TAG; };
struct IntNode : Node { int *i; };
struct FloatNode : Node { float *f; };

// ...

Node *node; node = getNode();
if (node->TAG == INT_TAG)
    *(((IntNode *) node)->i) = 123; // SAFE?

node = getNode();
*(((FloatNode *) node)->f) = 3.14f; // SAFE?
```

Detecting Field Overflow Memory Safety Bugs



- Existing Pointer Analyses for LLVM inadequate
  - Not scalable (SVF, Phasar)
  - Not precise enough (SeaDsa)

TeaDsa

- Based on SeaDsa
  - Context-, field-, array-sensitive
  - Unification-based (Steensgaard-style)
- Type- and offset-based field sensitivity

- 65% checks discharged with types vs. no types

Is relying on types Sound for low-level languages?

- Casts, type punning, memcpy
- Potential memory faults

Statement	Inclusion-based	Unification-based
$p = malloc(n)$	$p \supseteq \text{loc(malloc)}$	$p \approx loc(malloc)$
$p = q$	$p \supseteq q$	$p \approx q$
$*p = q$	$\text{pts(p)} \supseteq q$	$\text{pts(p)} \approx q$
$p = *q$	$p \supseteq \text{pts}(q)$	$p \approx \text{pts}(q)$
$p = \&x$	$p \supseteq loc(x)$	$p \approx loc(x)$

Program	Size [kB]	SVF Time [s]	SeaDsa Time [s]	TeaDsa Time [s]	% Checks Discharged with Types
bzip2	29	173	0.19	0.19	**0**
mcf	37	1.98	0.02	0.03	**--**
libquantum	80	8.66	0.08	0.09	**--**
sjeng	308	260	0.44	0.45	**0**
CASS	765	5390	6.20	5.85	**65**
htop	800	--	5.02	3.80	**71**
hmmer	859	2548	3.51	3.60	**1**
h264ref	1784	11525	9.44	10	**26**