FRAMER: A Tagged-pointer Capability Model with Memory Safety Applications

Myoung Jin Nam (Korea University)
Periklis Akritidis (Niometrics)
David J Greaves (University of Cambridge)
Spatial Memory Safety

Check lower bound $\leq p' \leq$ upper bound

Lower bound

Upper bound

object 1

object 2

$\text{pointer:}$

$p \rightarrow p'$

Check lower bound $\leq p' \leq$ upper bound
Fragmentation
Object_1
Fragmentation
Object_2

Disjoint metadata (per-object)
- entry_1: lower bound, upper bound, ...
- entry_2: lower bound, upper bound, ...
- entry_3: lower bound, upper bound, ...

What if there are zillions of objects (objects) to track?
Trade-offs

- For the better speed, previous solutions trade-off..
  - Compatibility (Fat Pointers)
  - Precision (Baggy Bounds)
  - Memory space (Address Sanitizer)
  - Full meaningful 48 bits of address space (SGXBounds)
  - Determinism (ARM’s Memory Tagging Extension)
Our Trade-offs

- We secure
  - Deterministic memory protection
  - Data memory efficiency
  - Full 48 bits of address space

- We sacrifice
  - Dynamic instructions counts
    - Rein the increase in extra cache misses for metadata access
    - Tolerate the growth in instructions for arithmetic operations

Hardware implementation will largely resolve our sacrifice
Goals and Challenges

- Derivation of metadata location from a tagged pointer
  - What to stuff unused top 16 bits in a 64-bit pointer?
  - Generic and complete encoding
    - No assumptions on object location, size, or alignment
- Deterministic bounds checking
  - Do not reply on probability
  - Resolve false negatives from violation of intended referents that challenged object-tracking approaches
Header address is derived from a tagged pointer holding relative location.
A N-frame is defined as a memory block with the size and alignment of $2^N$. It is aligned by $2^N$. The diagram illustrates a memory block labeled as "Memory" with a N-frame and the size $2^N$.
Ex) c is a 3 byte-sized object
An object is inside at least **two** bounding frames.
An object’s wrapper frame is defined as the **smallest** bounding frame.

Ex) c has **one** wrapper frame (3-frame) and is called 3-object.
A k-object’s 1st and last bytes sit in its wrapper frame’s lower and higher-addressed (N-1)-subframe, respectively.
Get Wrapper Frame Size 2/2

lower bound $b = [63:0]$
upper bound $e = [63:0]$
X: don’t care value

\[
\begin{array}{ccccccccc}
\text{b} & b_{63} & b_{62} & \ldots & b_k & b_{k-1} & b_{k-2} & \ldots & b_0 \\
\text{e} & e_{63} & e_{62} & \ldots & e_k & e_{k-1} & e_{k-2} & \ldots & e_0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 0 & \ldots & 0 & 1 & \text{X} & \ldots & \text{X} \\
\end{array}
\]

\[\text{XOR}\]
\[\text{CLZL}\]

\[\log_2 \text{(wrapper frame size)} = 64 - \text{result} = k\]
1. Get the base of the wrapper frame = \( p \& ((\sim 0) \ll n) \)
2. Add the offset to the wrapper frame base.
We need the followings to derive the header location

1) binary logarithm of wrapper frame size, N
   ✷ this fits in top 16 bits of a 64 bit pointer.

2) offset (wrapper frame base ~ header)
   ✷ this may NOT.
   ✷ ex) (N=20)-object’s offset is up to 19 bits.

We need more tricks to squeeze information and stuff a tag.
We divide memory space into slots, (N=15)-frame
For (N≤15)-objects, we tag an offset and turn on flag.
if flag == true, (p & ((0)\ll \log_2 \text{slot}\_\text{size})) + \text{offset}
We use a supplementary table only for (N>15)-objects.
Compact Shadow Space

More recent approaches’ shadow space

Mapping each fixed-sized memory block to an entry
For (N>=16)-objects, we tag N value and turn off flag.
The entry holds a header location (or metadata).
In Summary

Calculation of a header address is fairly simple

```
if flag==1  tag := offset  (common cases)
else flag==0  tag := N  (uncommon cases — 1/200,000)
```
Tracking objects requires checks at pointer arithmetic to keep track of intended referents.
int *p;
int *a = (int*)malloc(100*sizeof(int));
for (p=a; p<&a[100]; ++p)
  *p = 0;
/* p == &a[100] */

Should we check bounds at pointer arithmetic AND memory read/write??
FRAMER’s Solution

Object’s bounding frame

Object 1

Object 2

pointer: P

P’

P’ even violates an intended referent, but is still derived to header

Pad 

imaginary 

bytes when deciding a wrapper frame
In-frame Checking 1/2

In-frame checking catches the case $P'$

Check only in-frame at pointer arithmetic:
assert $((p' \land p) \land (\sim 0ULL \ll N) == 0)$;
In-frame Checking 2/2

No false negative, but one kind of false positives with in-frame checking **ON**
FRAMER pass is implemented as a LLVM LTO pass.
Experimental Results

- **Memory overheads (maximum resident set size)**
  - FRAMER’s store-only: 22%
  - FRAMER’s Full: 23%
  - Address Sanitizer: 784%

- **Run-time overheads (cycles)**
  - FRAMER’s store-only: 70%
  - FRAMER’s Full: 223%
  - Address Sanitizer: 139%

28 tests among Olden, Ptrdist, SPEC2006
The Cost of FRAMER

Runtime overheads for metadata management and retrieval
(overhead for bounds checking is excluded here)

ISA will largely resolve run-time overheads for calculation + untagging + bounds checking.
Contributor to Overheads

- **Memory overheads**
  - Mandating 16 alignment for compatibility with \texttt{llvm.memset}
  - Generous header size (16 bytes)
  - Fixed division array size (48 elements)

- **Run-time overheads**
  - FRAMER’s increase in dynamic instructions is high.
    - FRAMER (Full): 425%
    - Address Sanitizer: 226%
  - FRAMER is L1 D-cache efficient.
    - FRAMER’s Full: 40% (miss counts)
    - Address Sanitizer: 131% (miss counts)
Advantages

- Generic and sound design with many applications
  - Memory safety, Type safety, Garbage collection and etc
- Memory footprint and D-cache efficiency
  - No padding, re-alignment or grouping is required.
- Possible deployment for both stages
  - For development stage
    - Near-zero false negatives
    - Near-complete memory safety with type information
  - Practical deployment with support of customised ISA
Discussion

- The penalty of using tagged pointers
  - Tag setting up and cleaning
- More compact encoding
  - offset, the supplementary table entries
- Manipulation of memory object re-arrangement
  - Reduce/remove indirect access or branch for (N>15)-objects
Thank you!