On the Scalability of HeapCheck

Rick Boivie, Gururaj Saileshwar*, Tong Chen,
Benjamin Segal, Alper Buyuktosunoglu

IBM Thomas J. Watson Research Center

Yorktown Heights, New York, USA

DSN 2022

The 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks
Baltimore, Maryland, USA, June 27-30, 2022

*Gururaj Saileshwar is currently affiliated with Georgia Tech

Executive Summary *from our 2021 Paper*

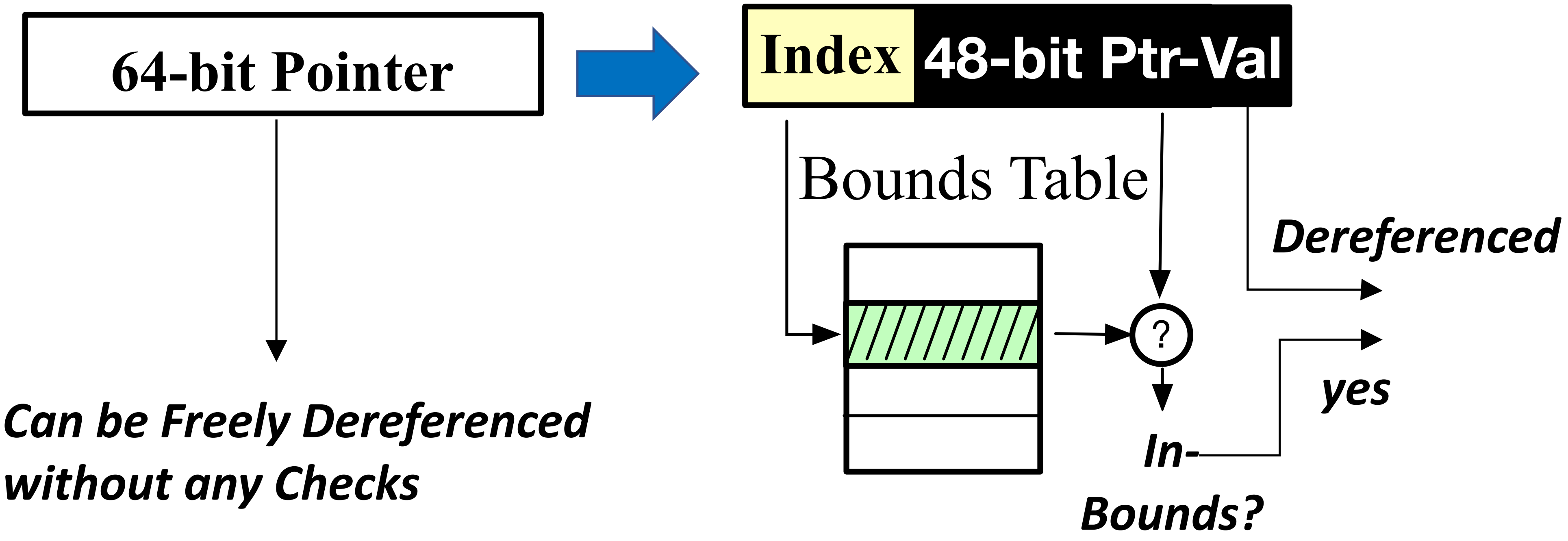
- **Problem**: Memory-Safety bugs like buffer-overflow & use-after-free are a serious problem and have been at the root of many security problems for more than 3 decades
  - E.g. The Morris Worm (1988), Buffer Overflow Based Code Injection, HeartBleed, Return-Oriented Programming (ROP), Spectre … -- and are at the root of ~70% of CVEs
- **Existing solutions** for enforcing memory safety have poor coverage, or high performance-overhead, or require disruptive changes
- **Our Solution:** A HW/SW co-design for precise bounds-checking
  - that is **minimally-invasive**
    - requires **no changes to source-code**
    - and **no changes to binary-layout** (i.e., retains compatibility with existing libraries)
  - with **minimal performance impact (1.5% overhead)**
- **Some Results:**
  - Our design detected all the vulnerabilities and prevented all the attacks in the 'How2Heap' Exploit Suite
  - Detected 87 memory-safety bugs in glibc and in the SPEC-CPU 2017 benchmark programs that, to our knowledge, had not been previously detected
    - Our design can detect & prevent bugs in unmodified/un-instrumented shared library code

Today’s Talk on *HeapCheck Scalability*

- Extends HeapCheck to support “very large” numbers of “active” objects concurrently
  - (e.g. > 16,577,215)
- While preserving the HeapCheck benefits
  - *Precise* rather than *Probabilistic* detection of Memory-Safety errors
  - Low overhead (1.5% overhead on average across the programs in the SPEC-CPU2017 benchmark suite)
  - No changes to source code or binary layout, and compatible with existing libraries

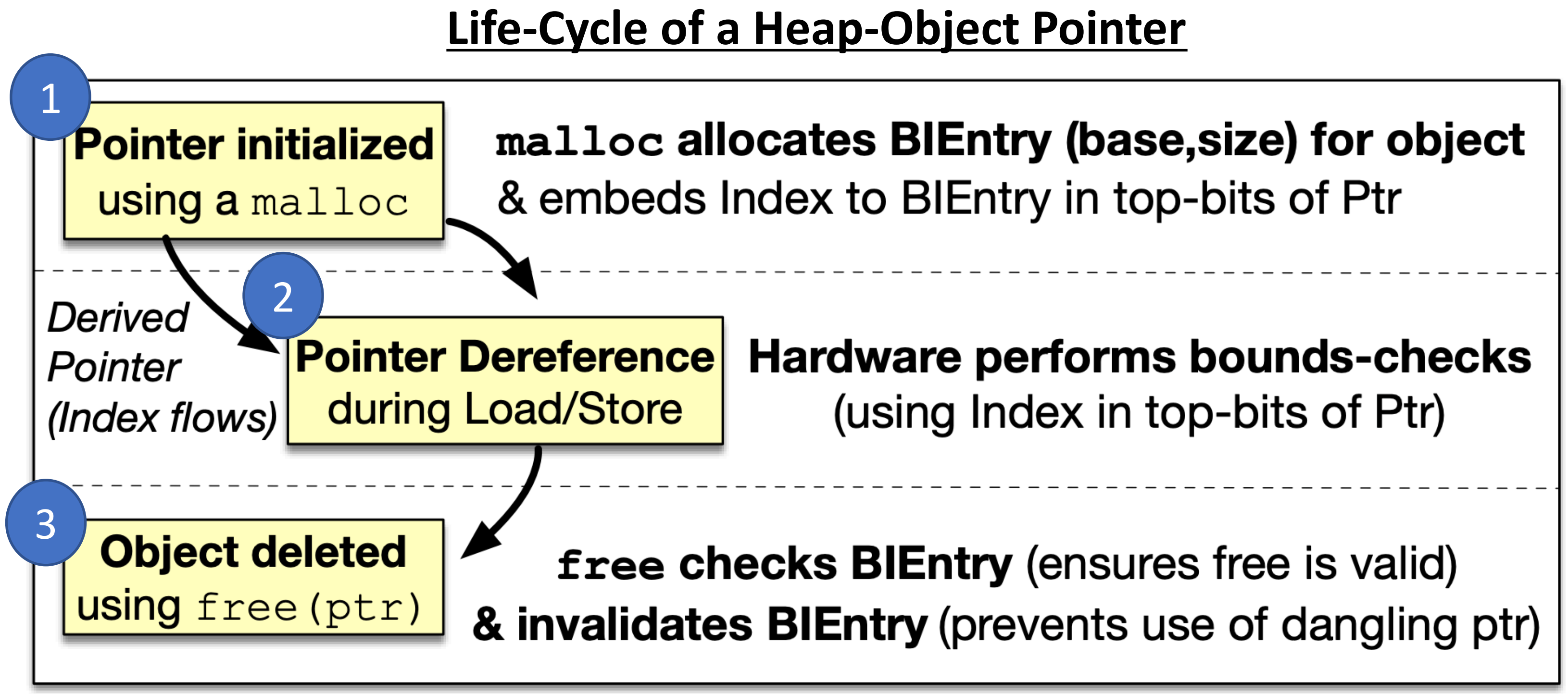
- Comparisons to other approaches to Memory-Safety
  - Our DSN 2021 paper compared our approach to other approaches that had been presented previously
  - In today’s talk, we also compare our approach to two additional approaches that have been announced in the last year

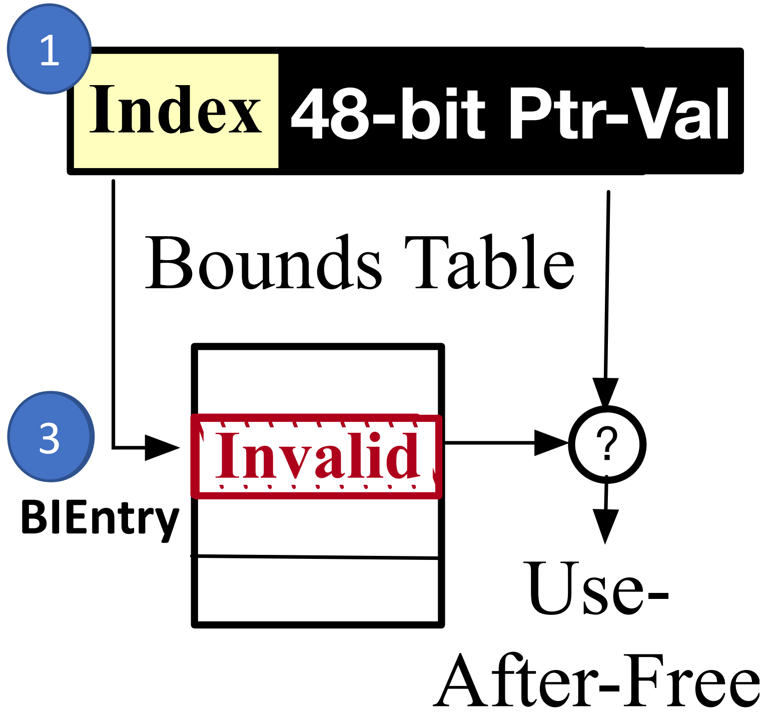
HW-assisted Bounds-Checking on Loads and Stores



Use the Top-Bits of the Pointer to Store Metadata;
Enforce Bounds-Checks Transparently in Hardware

Bounds-Checking for Heap-Objects


Life-Cycle of a Heap-Object Pointer
1
Pointer initialized using a malloc
malloc allocates BIEntry (base,size) for object & embeds Index to BIEntry in top-bits of Ptr
Derived Pointer (Index flows)
2
Pointer Dereference during Load/Store
Hardware performs bounds-checks (using Index in top-bits of Ptr)
3
Object deleted using free(ptr)
free checks BIEntry (ensures free is valid) & invalidates BIEntry (prevents use of dangling ptr)

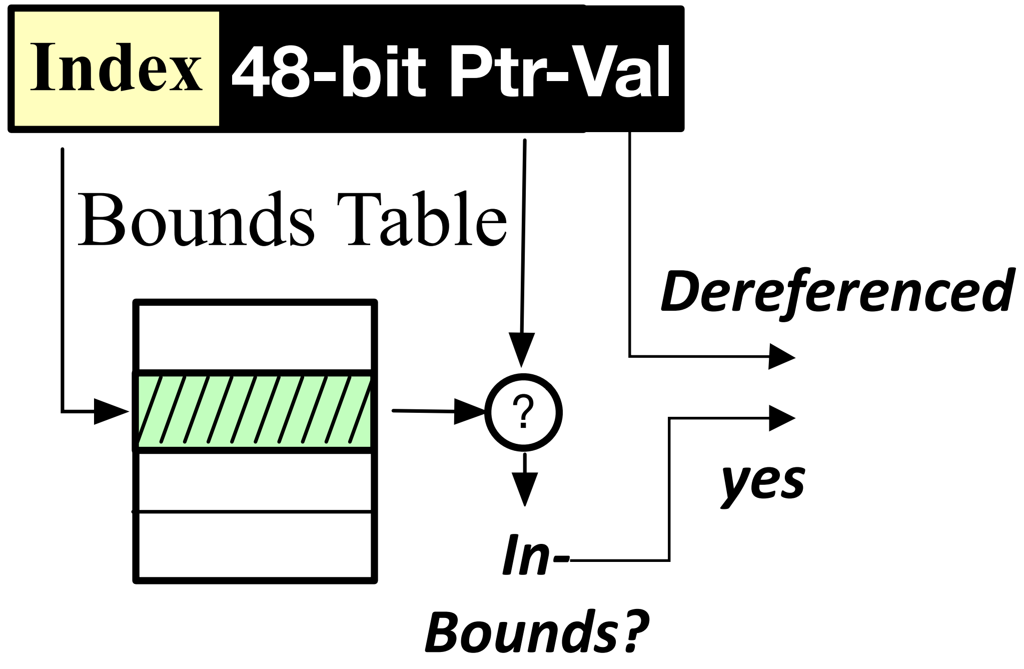


HW-assisted Bounds-Checking on Loads and Stores

Minimal performance impact since:

- Bounds information flows "automatically" without extra instructions when a pointer is assigned to another, passed in a function call or used to compute another address in array indexing or pointer arithmetic,

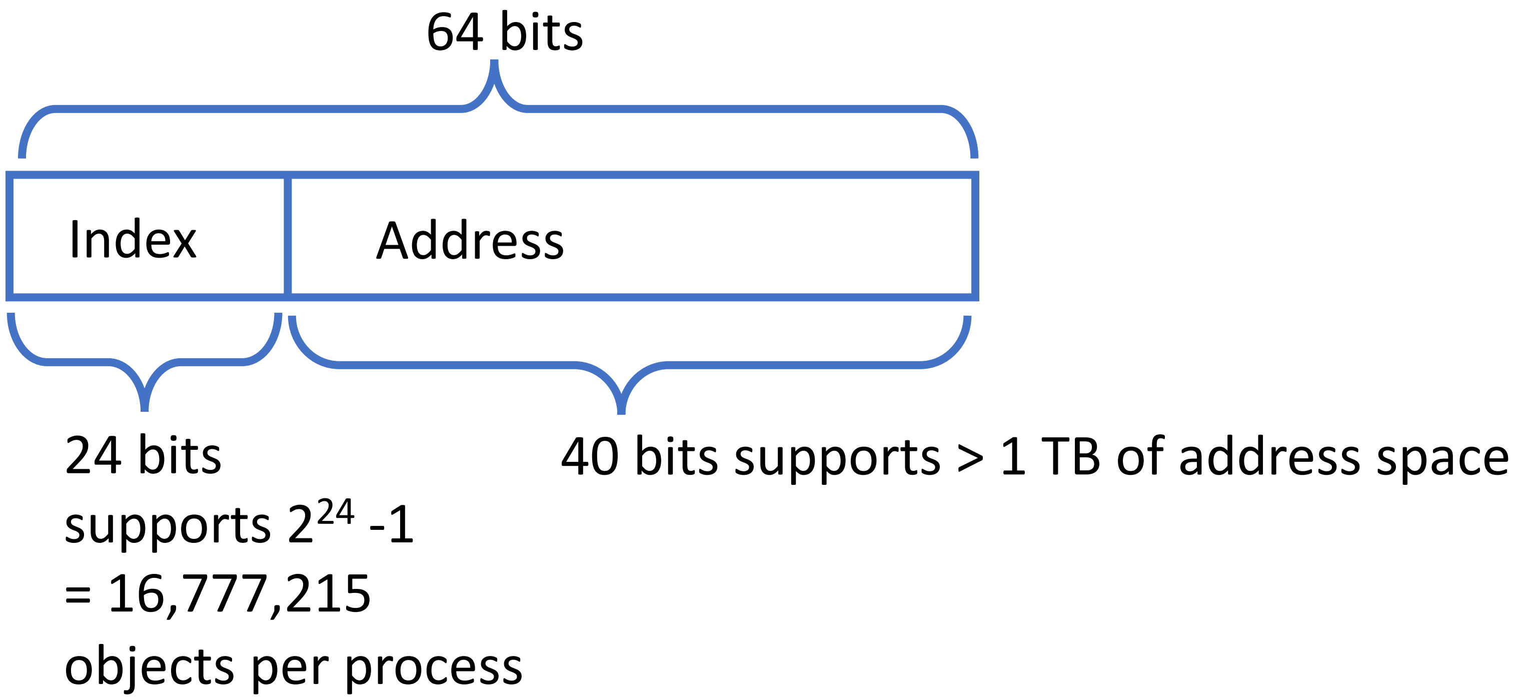
- and, since all the addresses within a given buffer and all the pointers to that buffer have the same "index", the bounds information for an address is often available in an on-chip "Bounds Information" cache



1.5% overhead

One Limitation

- One limitation is the number of allocated “objects” that can be concurrently monitored



- In this talk, we discuss how HeapCheck can be used when the number of concurrently active objects exceeds the number that can be supported in the available index bits

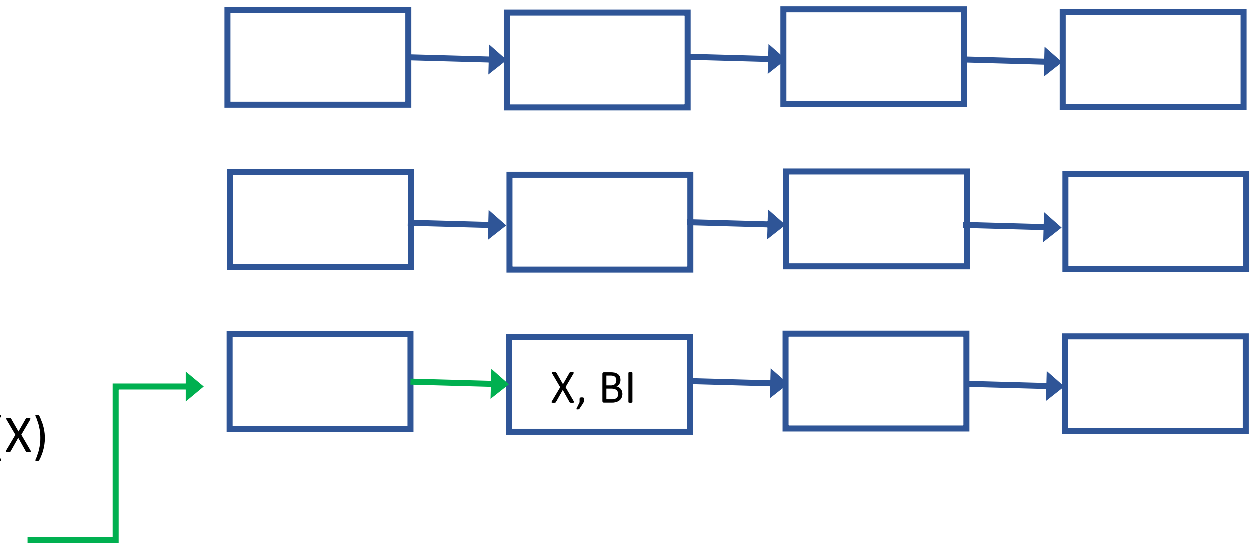
Use of an Overflow Table when Entries Don’t Fit in the Bounds Table

- As long as the # of ***active*** objects <= 2^{24} - 2, everything works as previously described
    - (entries in the Bounds Table may be re-used when an object is free-ed)
- But if a malloc would make the # of active objects > 2^{24} -2,
    - the address of the new object will have a special “all 1’s” value in its index bits
    - and the Bounds Information for the new object ***may*** be stored in the Overflow Table
- “Bounds Information registers” are used to reduce the need to access the Overflow Table
    - e.g., on a LOAD from an address in GPR-3, the CPU h/w uses BI-3 to determine if the LOAD is in-bounds or out-of-bounds
    - The BI registers also propagate Bounds Information “automatically” when one pointer is assigned to another, passed in a function call or used to compute another address in array indexing or pointer arithmetic
        - e.g. if an address in GPR-3 is used to compute an address in GPR-5, BI-3 is copied to BI-5

Storing Bounds Information in and Retrieving Bounds Information from the Overflow Table

- When a pointer is stored in memory, Bounds Information may be stored in the Overflow Table
  - e.g. if GPR-6 is a pointer that is stored in memory at address X, and GPR-6 has the “all 1’s” value in its index bits, then <X, the value in BI-6> is stored in the Overflow Table
    - the Overflow Table is a hash table & the bucket used is based on a hash of address X
  - Later, when the pointer at address X is loaded into GPR-7, say, BI-7 is loaded from the entry in the Overflow Table corresponding to address X

When a pointer with the “All 1’s” index is loaded into GPR-i from memory address X

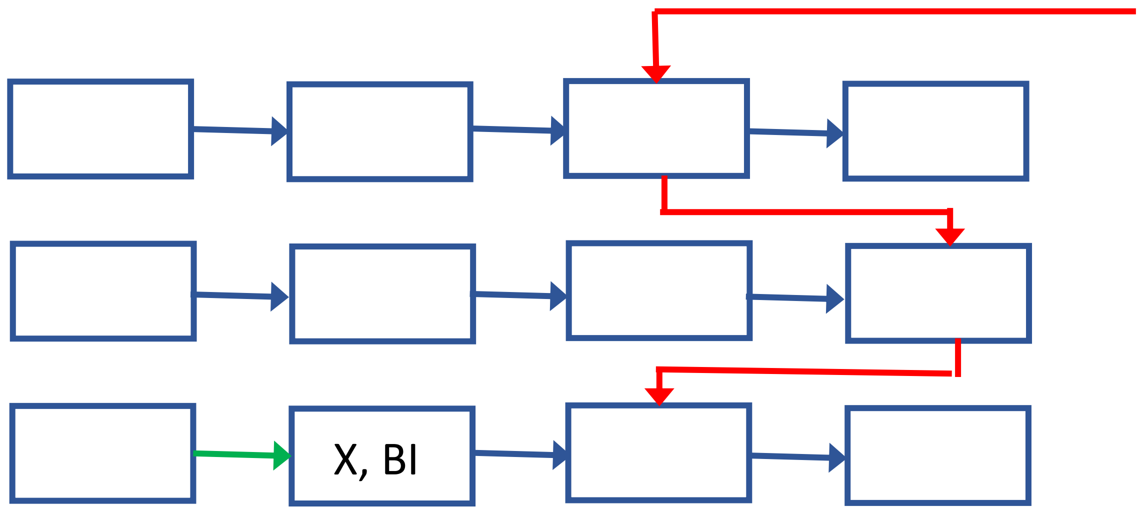


Hash Bucket hash(X) Is used to find the entry for X in the Overflow Table

And the Bounds Information in this entry is loaded into BI-i

Freeing Bounds Information in the Overflow Table and BI registers

- When an object is free-ed via a pointer that has the "all 1's" index value, the free may need to delete multiple entries in the Overflow Table and may need to delete Bounds Information from multiple BI registers
  - Each BI register with Bounds Information for the free-ed object is set to 'no-valid-bounds-information'
    - Then if a pointer in GPR-i is used in a load or store and if BI-i == 'no-valid-bounds-information', an "invalid address" exception is generated
  - A free of an object with the "all 1's" index value frees all Overflow Table entries for pointers to that object



When an object with Bounds Information Y is freed, hash(Y) is used to find and free the entries in the Overflow Table corresponding to Bounds Information Y

Then if a pointer with the "all 1's" index value is loaded from address X into GPR-i and there is no Overflow Table entry for X, BI-i is loaded with 'no-valid-bounds-information'

A Change to malloc

- When malloc allocates an object and the number of active objects > $2^{24} - 2$, malloc stores the Bounds Information in an appropriate BI register before returning the pointer to the newly allocated object
  - For example, malloc will store the Bounds Information in BI-0 before it returns a pointer to the allocated object – with the ”all 1’s” in its index bits -- in GPR-0

Performance

- This approach for incorporating an Overflow Table into HeapCheck provides the efficiency advantages discussed previously for 16,777,214 active objects
- and it uses BI registers to reduce the need to access Bounds Information from the Overflow Table for objects beyond 16,777,214

- The applications in the SPEC-CPU 2017 benchmark suite don’t use anywhere near 16,000,000 active objects – and HeapCheck’s performance overhead across these benchmarks is 1.5%
- Another paper[1] analyzed “real world” applications, including apache, mysql and others & showed that while these applications use a very large number of “objects”, they only use a few thousand objects *concurrently*

- If most real-world applications are like this, the Overflow Table will rarely be used

1. Hardware-based Always On Heap Memory Safety, In 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020, pages 1153–1166

HeapCheck can also be used in “Fuzz Testing”

- What is “Fuzz Testing”
  - An automated testing technique that provides a program under test with invalid, “unexpected” or “unexpectedly long” randomly-generated inputs so buffer-overlow and other memory-safety bugs can be trapped, debugged and fixed prior to deployment
- Why do Fuzz Testing if we have a good run-time solution
  - Because it’s better to catch & fix bugs during development & test then it is to generate an interrupt & halt execution “in production”
- Why use HeapCheck for Fuzz Testing if we have tools like Address Sanitizer (ASAN)[2]
  - Because HeapCheck can catch bugs that ASAN cannot (e.g. an out-of-bounds reference that misses its “red-zone” or a bug that manifests itself in some unmodified, un-instrumented code – like in the string functions in glibc)
  - and ASAN has significant performance overhead (1.73x slowdown). Since HeapCheck provides precise, hardware-assisted memory-safety protection with only 1.5% overhead, HeapCheck can increase the throughput of Fuzz Testing -- and provide better coverage
- Why use HeapCheck at “Run-time” if we have tools for Fuzz Testing
  - Because Fuzz Testing may not catch every bug

2. AddressSanitizer: A fast address sanity checker. In Gernot Heiser and Wilson C. Hsieh, editors, 2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012, pages 309–318

Recent Industry Effort – Intel

- October 2021, Intel presented ‘Cryptographic Capability Computing’[3]
  - Does not prevent out-of-bounds accesses
  - Uses cryptography to ensure that an out-of-bounds load loads “garbled” data and an out-of-bounds store stores “garbled” data
    - The former prevents an adversary from using an out-of-bounds load to steal sensitive information (as in ‘Heartbleed)
    - The latter prevents an adversary from using an out-of-bounds store to inject data chosen by the adversary into a program (as in ‘Return Oriented Programming’)
  - But this may introduce other problems including unpredictable program behavior and may be hard to debug
- By contrast HeapCheck prevents out-of-bounds accesses and generates an interrupt at the point of an out-of-bounds access so out-of-bounds accesses can be more easily identified, debugged and fixed

3. Cryptographic Capability Computing. MICRO ’21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021, pp. 253–267

Recent Industry Effort – ARM

- January 2022, ARM announced an ARM-based Prototype SoC with Enhanced Security
    - Morello is a 5-year research initiative involving a consortium led by ARM to design a new, inherently more secure, ARM-based computing platform for the future
    - ARM announced that the Morello program hit a new milestone in defining nextgen security[4]
    - and they're making an ARM-based SoC and demonstrator board available to Google, Microsoft and other partners across industry and academia via the UKRI 'Digital Security by Design (DSbD)' initiative
    - The new SoC is based on the University of Cambridge's CHERI architecture
- As we discuss in our 2021 paper
    - "FAT-Pointer"-based solutions -- including CHERI -- store the bounds information in a separate word alongside the actual pointer value so a bounds check can be done when a pointer is de-referenced
    - This increases the size of pointers which can negatively impact performance by reducing the amount of other data that fits in a cache, and requires changes to a program's binary layout that is not compatible with existing libraries and system calls
    - and unless some additional mechanism is introduced, FAT-Pointer based solutions cannot protect against temporal problems such as use-after-free bugs
    - Our approach, by contrast, doesn't have any of these disadvantages

4. https://www.arm.com/company/news/2022/01/morello-research-program-hits-major-milestone-with-hardware-now-available-for-testing

Summary

- We presented an extension of our 2021 paper that protects against memory-safety bugs like buffer-overflow and use-after-free
- In the current paper, we extended our approach to accommodate very large numbers of dynamically allocated arrays and structs
- We also compared our approach to two recently announced approaches from Intel and ARM
- We believe our approach has advantages over other approaches and can provide an effective, low-cost, easy-to-deploy solution that can protect software from an important problem that has existed for more than 30 years

Thank you

Backup

Confidential Computing and Memory Safety Protection

- Confidential Computing protects a program from "*external*" attacks
  - so an Adversary cannot, from outside a program, access or tamper with information inside the program
- Memory Safety Protection protects a program from "*internal*" attacks
  - so an Adversary cannot exploit a "buffer overflow" or other "memory safety vulnerability" *inside* a program to steal information (as in Heartbleed) or take control of the program (as in Return-Oriented Programming)