TAILCHECK: A Lightweight Heap Overflow Detection Mechanism with Page Protection and Tagged Pointers

Amogha Udupa S. G.   Raveendra Soori   Michael Ferdman   Dongyoon Lee

Stony Brook University

July 11, 2023

Problem: Heap Overflow

- C/C++ lacks memory safety
- 2022 CWE top-most dangerous software weaknesses
- Security implications

- privilege escalation

< Blog Home

CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)

Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.
January 26, 2021 - 12 min read

Last updated on: *December 23, 2022*

Update Feb 3, 2021: It has been reported that macOS, AIX, and Solaris are also vulnerable to CVE-2021-3156, and that others may also still be vulnerable. Qualys has not independently verified the exploit.

- Information leakage

The Heartbleed bug: How a flaw in OpenSSL caused a security crisis

Analysis
Sep 06, 2022 • 10 mins

Internet | Open Source | Vulnerabilities

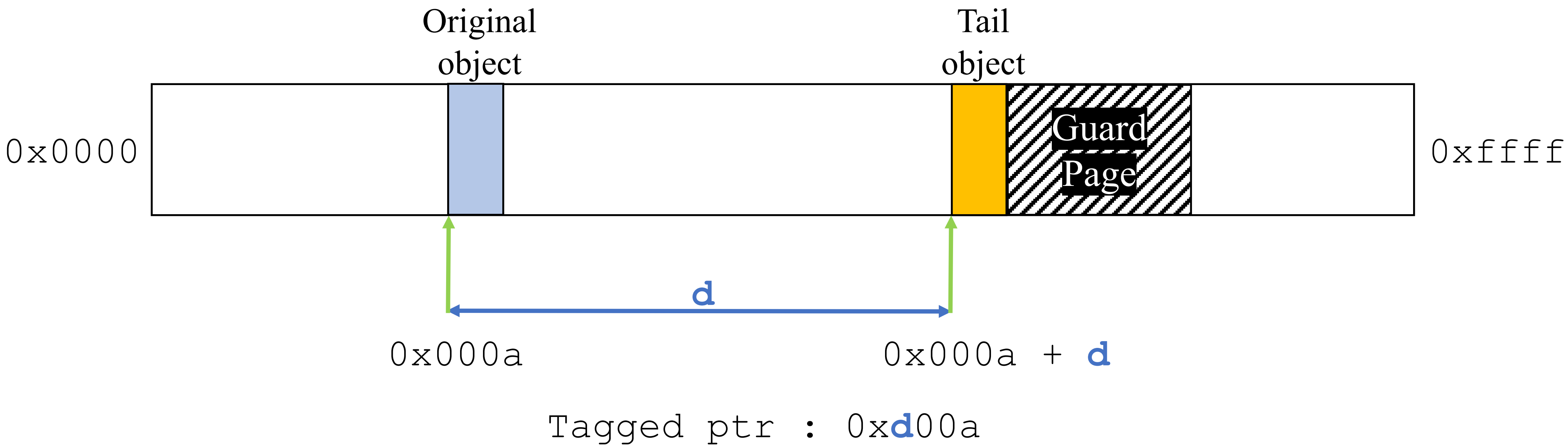
Heartbleed can be traced to a single line of code in OpenSSL, an open source code library. Here's how Heartbleed works and how to fix it.

Prior Solution Drawbacks

- Guard Pages (Ex. ElectricFence, PageHeap)
  - ✅ No metadata lookup and no explicit checks
  - ❌ High memory overhead, slow
- Explicit Bounds Checking (Ex. SoftBound)
  - ✅ Uses shadow memory region
  - ❌ Costly metadata lookup and bound comparison cost
- Pointer Tagging (Ex. Delta Pointers)
  - ✅ Quick metadata look up
  - ❌ Requires large tags, shrinks address space

TailCheck

1. Page Protection
2. Memory Dereference Duplication
3. Pointer Tagging

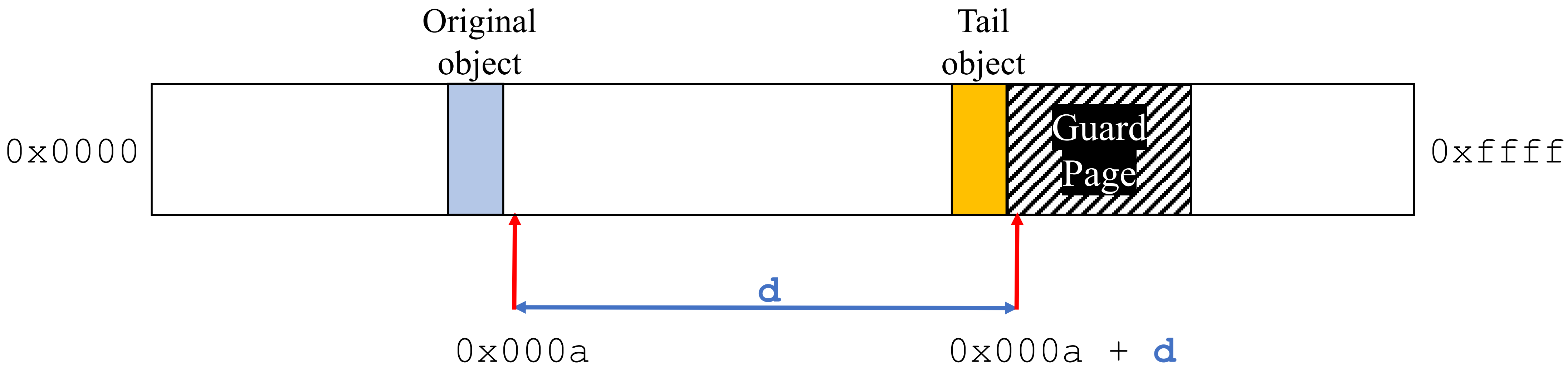


TailCheck

1. Page Protection

2. Memory Dereference Duplication

3. Pointer Tagging



Tagged ptr : 0xd00a

Out of bounds access ⇒ Page fault

Outline

- Introduction
- Design
  - Memory allocator
  - Compiler code instrumentation
- Evaluation
  - Security evaluation
  - Server application performance (vs AddressSanitizer)
  - SPEC CPU performance (vs Delta Pointers)

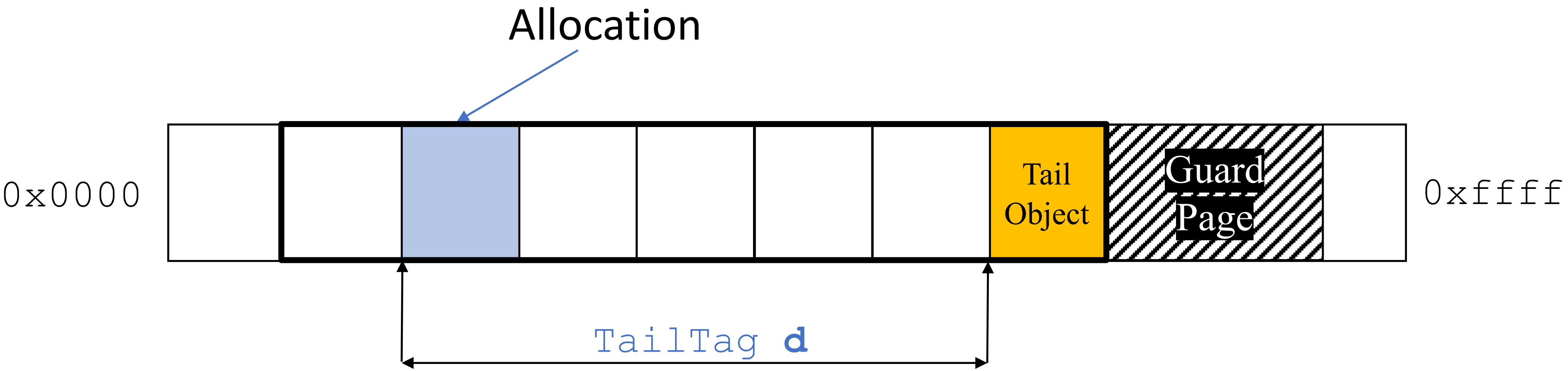
TailCheck Design

1. Memory allocator
    - Sets up guard pages
    - Initializes pointer tags
2. Compiler instrumentation
    - Adds duplicate memory access to a tail object (for OOB check)
    - Masks/restores pointer tags across un-instrumented library function calls

Reusing guard pages and implicit OOB check ⇒ Low cost

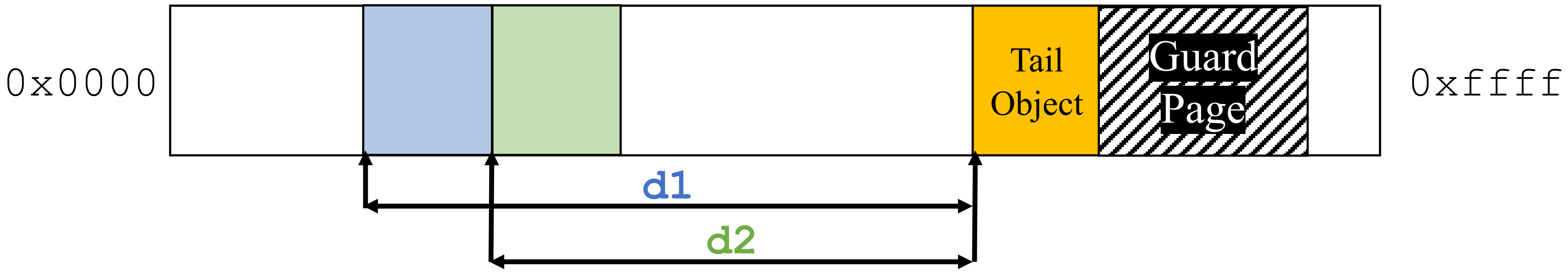
TailCheck Memory Allocator

- *mimalloc* based – equal sized blocks allocated together
- Last block reserved for TailObject, end aligned with Guard Page
- TailObjects are allocated for block-group size lesser than 64kB
  - 16-bit TailTag can represent up to 64kB distance
- TailTag is calculated for allocations, tagged pointer is returned

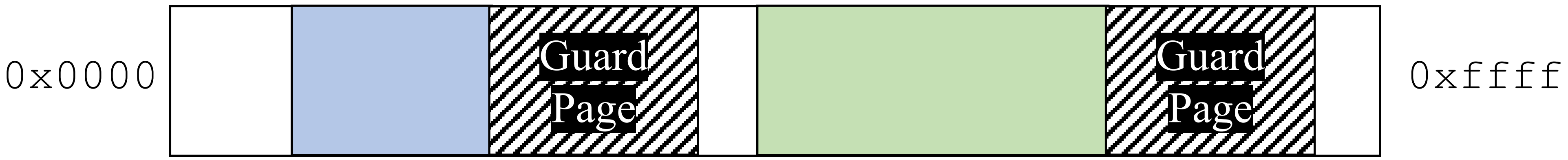


TailCheck Memory Allocator

- Small object pages share a single TailObject



- Large Objects are their own TailObjects
  - Large Objects have zero-value TailTag
  - Object end aligned to protected page



TailCheck Code Instrumentation

```
int* tagp = malloc(...)
load tagp
```

Transformed to...

```
ADDR_BITS = 48
MASK = ((1<< ADDR_BITS)-1)
p = tagp & MASK
d = tagp >> ADDR_BITS
load p+d  // TailCheck
load p
```

TailCheck Code Instrumentation

- LLVM Link-Time-Optimization passes
- Dereference Duplication
- CallSite Masking – remove tag at instrumentation boundary
- Optimizations
  - SafeAlloc – statically known safe access (Delta Pointers)
  - Hoist TailPointer calculation out of loops
  - Gather Pointer Arithmetic that use the same base pointer

Outline

- Introduction
- Design
  - Memory allocator
  - Compiler code instrumentation
- Evaluation
  - Security evaluation
  - Server application performance (vs AddressSanitizer)
  - SPEC CPU performance (vs Delta Pointers)

TailCheck Evaluation

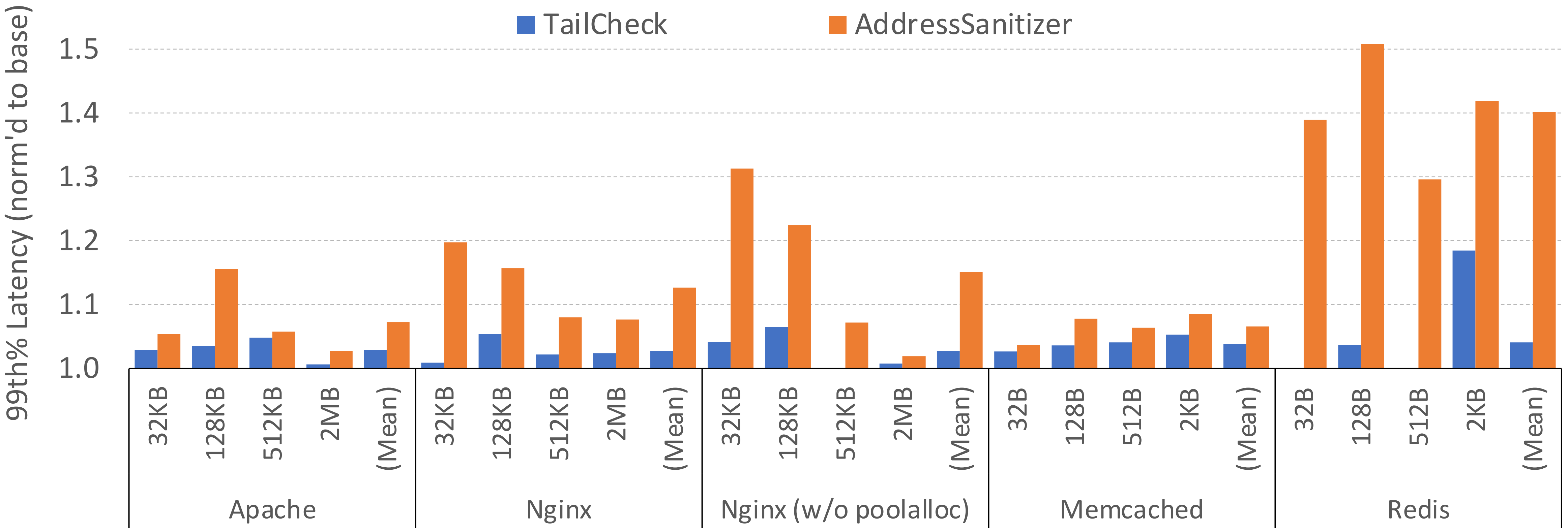
- Server Applications
  - apache, nginx – 256 request per second, varying file sizes
  - memcached, redis – 50% get/set ratio, varying value sizes
- SPEC CPU 2006, v1.0
  - C and C++ applications
- SPEC CPU 2017, v1.0.5
  - Speed set, C and C++
  - Single threaded

Security Evaluation

- Overflows are caught as segmentation faults
- SPEC CPU 2006 has no reported heap buffer overflows
- SPEC CPU 2017 gcc's illegal read in tree-ssa-sccvn.c:3365
  - Detected read of 4 bytes out of the allocated area
  - SPEC CPU 2017 v1.0.5 benchmark

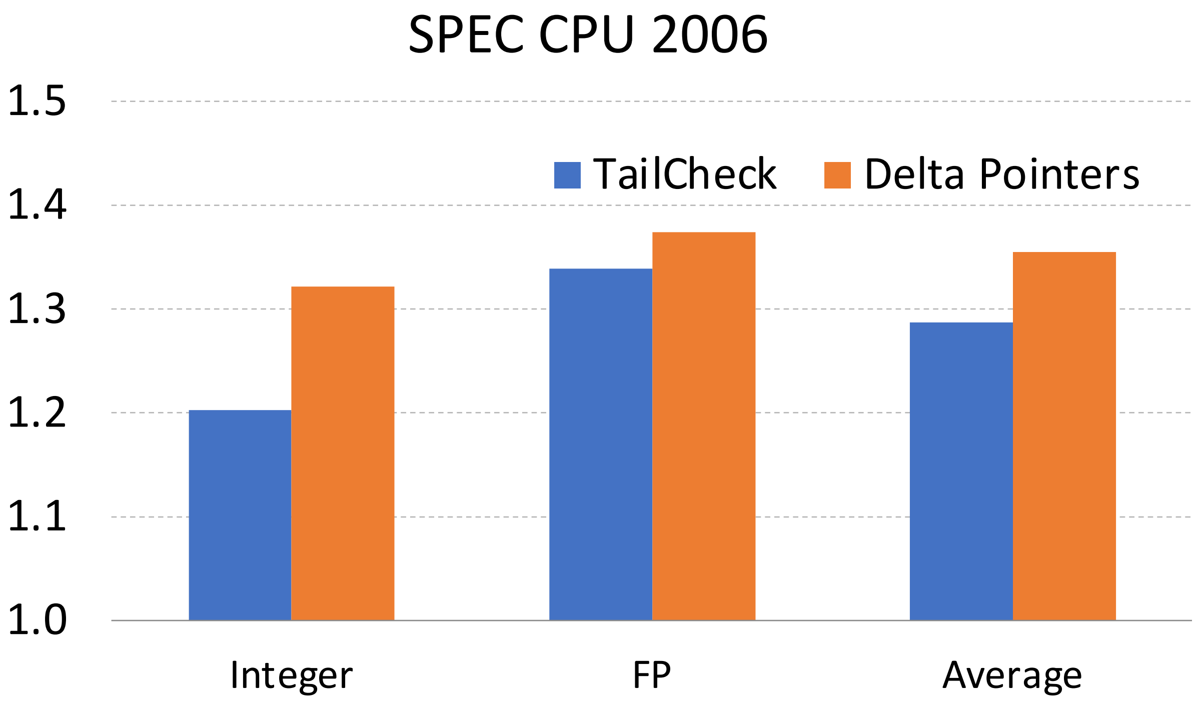
Server Application Performance

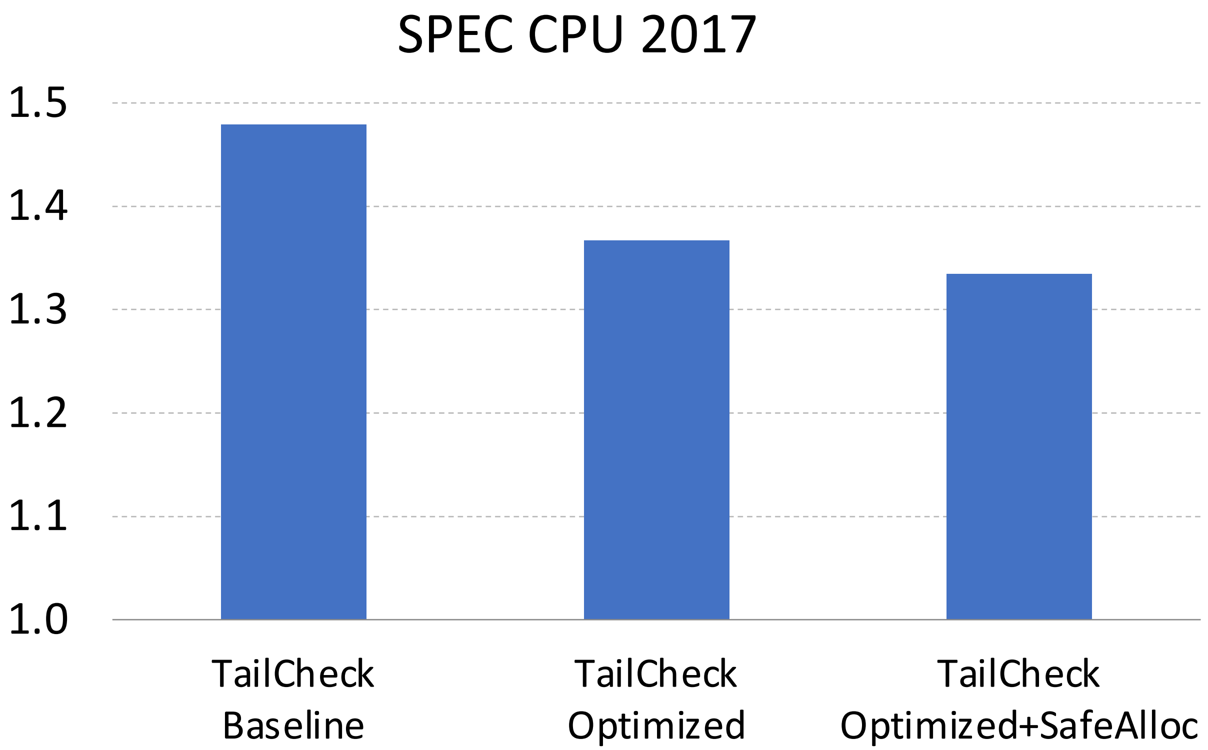
- Less than 4% overhead on tail (99th percentile) latencies
- 3x better compared to AddressSanitizer



SPEC CPU Performance

- On SPEC CPU 2006, TailCheck overhead is 29%
- On SPEC CPU2017, TailCheck overhead is 33% (peak memory: 9%)





Conclusions

- TailCheck offers page protection-based heap memory safety
  - TailCheck allocator + compiler managed tagged pointers
  - Duplicate memory dereference implicitly checks for out of bounds access
- Optimizations improve TailCheck performance by 20%
- TailCheck is fast, can be run in production
  - 4% and 3% overhead for the average and tail latencies for servers
  - SPEC CPU 2006 and SPEC CPU2017 overhead is 29% and 33%