Format

- Welcome
- Discuss Confidentiality
- Issue Clearing (if required)
- Meaningful Update & Follow-Up
- Review Next Papers
- Presentation
- Feedback

REVelry DangZero

Efficient Use-After-Free Detection via Page Tables

About Me | Jay Warne

Currently

- Project Work
  - Tech Lead for Normal Software Program
  - Principle Researcher
- Advisory
  - Product Development
  - Product Direction
  - Expert Reviewer

Previously

- PO for Videographic Data Analysis
- DARPA research
  - Processor Side-channels
  - Hypervisors
  - Rowhammer Style Attacks
  - Program Analysis Methods
- Ran a Security Ops Team
- Occasional RE & Red Team
- Field Forensic Analysis & Tool Deployment

Things I Like

- Alpine Ski Racing
- Ski/ Alpine Mountaineering
- Ice/ Rock Climbing
- Surfing

DangZero – Tl;dr

- If you make your own aliases with extra metadata, you can track memory usage better
- To get in between, you need to write yourself a module that either hooks on the system or runs the code in a virtualized environment or hooks (unexplored)
- Ring0 access required to do operations at kernel level (woah omg)

DangZero – “The Cool Part”

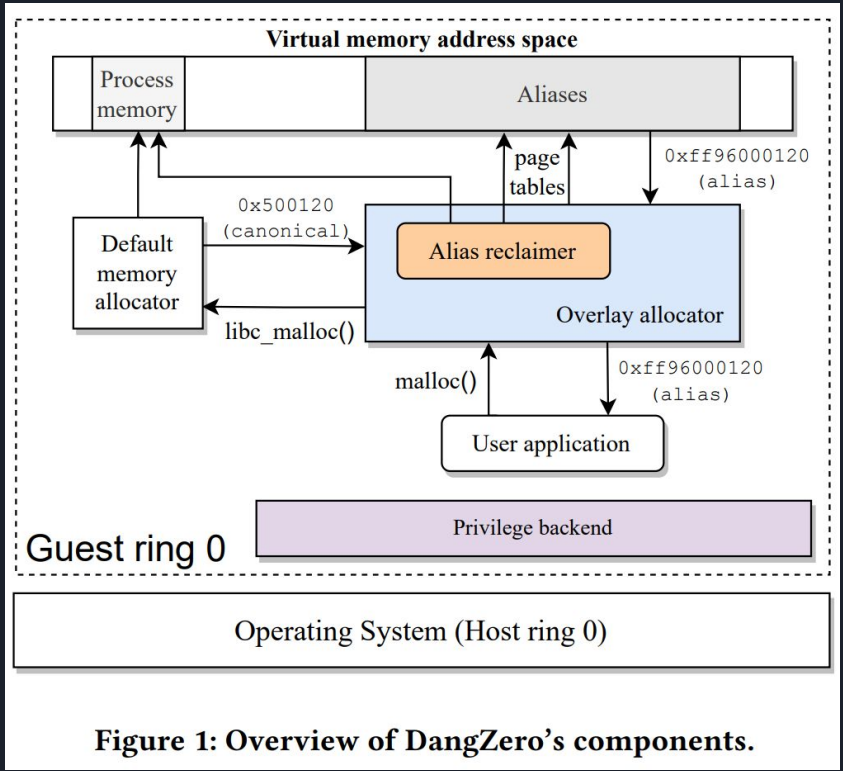

Figure 1: Overview of DangZero’s components.

UAF Detector Code Available At: https://github.com/vusec/dangzero

What We Will Cover

- The Cool Part
- Main Components
  - Overlay Allocator
  - Alias Reclaimer
- Implementation Nuance
- Performance
- Addl Resources in Appendices

Main Components

Main Components

- Overlay Allocator
- Alias Reclaimer

Proxy between User Space and Allocator

Main Components

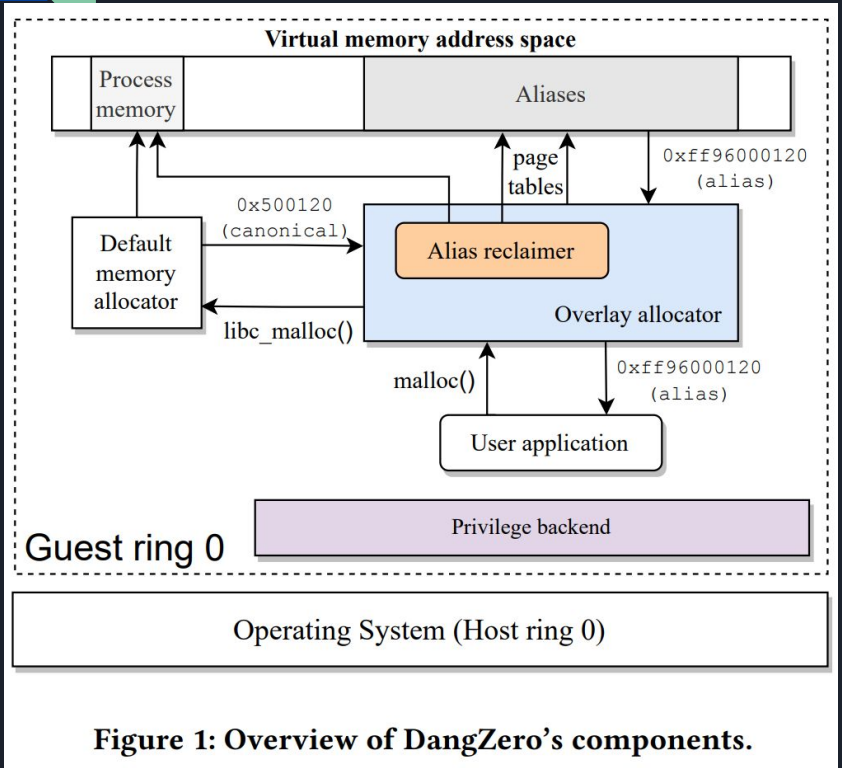

Figure 1: Overview of DangZero's components.

Proxy between User Space and Allocator

- Allocations Operates in 4 Steps:
  - (1) malloc() to overlay_alloc
  - (1) libc_malloc() to default allocator
  - (2) Canonical virtual address returned
  - (3) OA creates alias page(s)
  - (4) Alias Pages Returned
- Frees follow a similar pattern, but with additional metadata

Main Components

- Overlay Allocator
- Alias Reclaimer

The reclaimer does exactly what you would think that it would do at an interval by marking previously freed areas as reusable

- Two Main Phases
  - Marking Phase
  - Sweeping Phase
- Relevant Metadata
  - State of the page
  - Object boundaries

Main Components

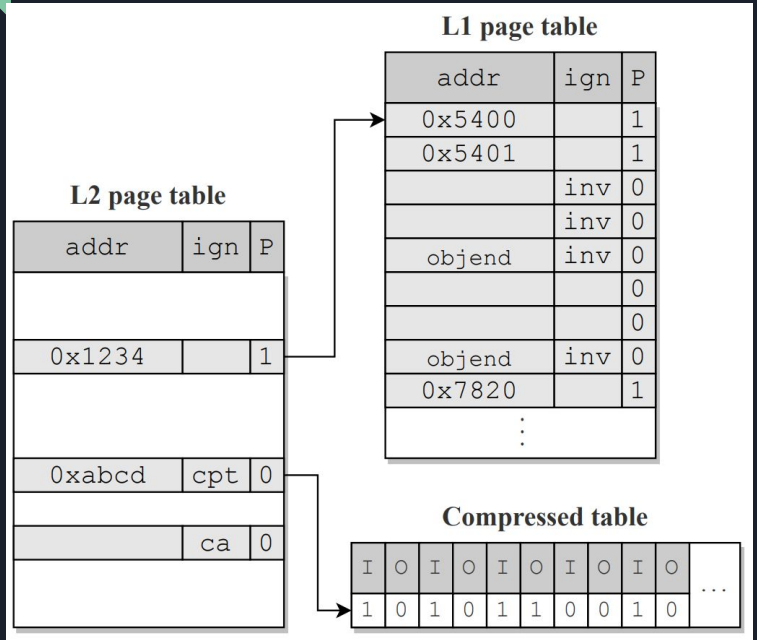

Figure 3: Alias reclaimer metadata in the page tables.

The reclaimer does exactly what you would think that it would do at an interval by marking previously freed areas as reusable

- Two Main Phases
  - Marking Phase
  - Sweeping Phase
- Relevant Metadata
  - State of the page
  - Object boundaries

Implementation Details

Implementation Details

- Privileged Backend
- Alias Page Tables
- Forking & Child Processes

- Built on top of Kernel Mode Linux
- “Potential Performance Benefits”
- Patch system calls into direct calls

Implementation Details

- Privileged Backend
- Alias Page Tables
- Forking & Child Processes

- Uses normally reserved Linux Kernel Address Space
- Kernel Captures more space than it uses in modern systems

Implementation Details

- Privileged Backend
- Alias Page Tables
- Forking & Child Processes
- Alias Pages Not Passed To Children
- Managed with an epilogue to Fork

Implementation Details

```
// construct physical -> canonical map (pre-CoW)
for canon_addr in heap:
    phys_addr = page_walk(canon_addr)
    map[phys_addr] = canon_addr

// reconstruct alias mappings in child
for alias_addr in alias_space:
    phys_addr = page_walk(alias_addr)
    canon_addr = map[phys_addr]
    *canon_addr; // trigger CoW to force new backing
    child_phys_addr = page_walk(canon_addr)
    pt_map(alias_addr, child_phys_addr)
```

Listing 2: Recreating alias mappings in the child after fork.

Walk page tables for canonical addrs in parent & read physical page, creating PAddr-Canonical mapping, then:

1) Get the physical address for each alias
2) Touch VAddr (trigger CoW)
3) Get the child-PAddr via page walk
4) Create PT Entries in child alias space

Implementation Details

- Privileged Backend
- Alias Page Tables
- Forking & Child Processes

- Alias Pages Not Passed To Children
- Managed with an epilogue to Fork
- Sync the parent to wait for child to finish to keep memory state consistent

Performance

Performance

System	Reported	Measured
DangZero-base	-	14.0%
DangZero-alias-reclaim	-	16.0%
Oscar	40.0%	40.0%
MineSweeper	5.4%	10.0%
MarkUs	10.0%	14.0%
FFmalloc	2.3%	1.2%

Table 1: SPEC CPU2006 runtime overhead compared.

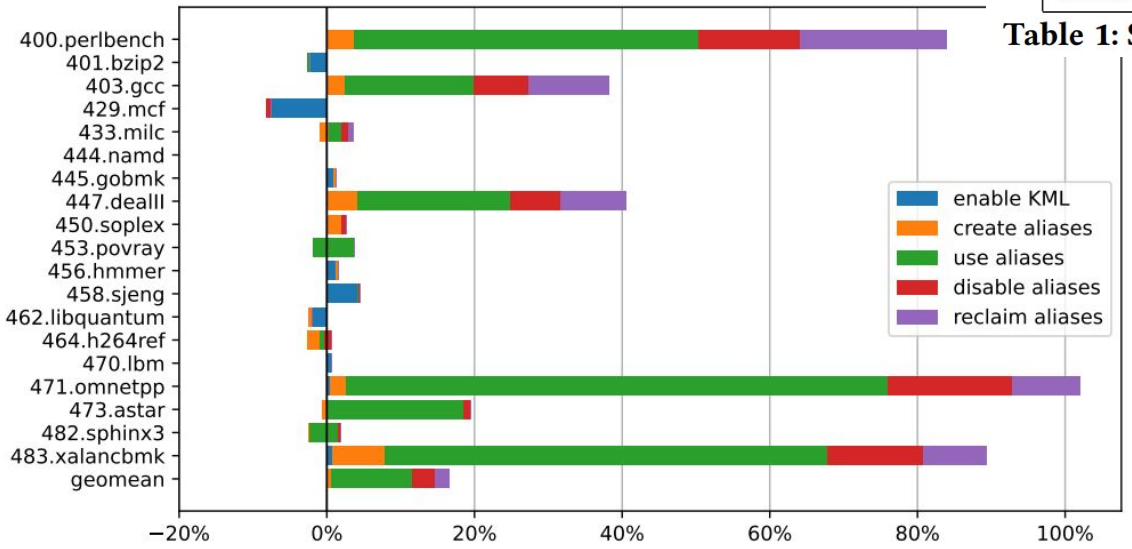

Figure 5: SPEC CPU2006 runtime overhead for DangZero.

Performance

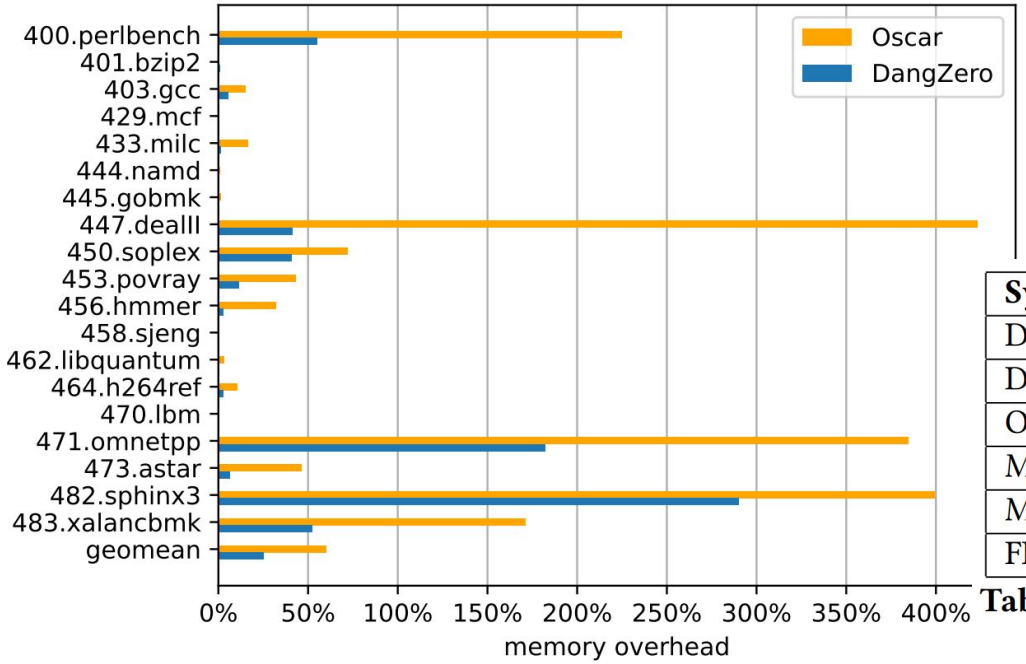

Figure 6: SPEC CPU2006 memory overhead for DangZero.

System	Reported	Measured
DangZero-no-compression	-	75.0%
DangZero-with-compression	-	25.0%
Oscar	60.0%	nontrivial
MineSweeper	11.1%	22.3%
MarkUs	16.0%	26.9%
FFmalloc	61.0%	115.8%

Table 2: SPEC CPU2006 memory overhead compared.