

# Hopin Resources for Speakers\*

## Before your session:

- Check your [network connectivity](#) and [browser compatibility](#)
- Sign up for a [Hopin demo](#) or contact us at [blinkon@chromium.org](mailto:blinkon@chromium.org) with questions

## Additional resources:

- [Hopin Knowledge Center](#)
- [Quick Troubleshooting Reference Guide](#)
- [Sessions Tutorial - During the Event](#)
- [Tips for Speaking in and Moderating Sessions](#)
  - Your session is an Open Session (anyone can participate on screen)

*\* Please note that this slide is for speakers only. Please feel free to delete this slide before sharing.*

# Breakout Session Instructions\*

## To join your session:

- Head to the calendar invite or the Sessions tab in Hopin and find the Session you'll be hosting
- Click Share Audio and Video in the center of the Session screen
  - If you haven't allowed access to your camera or mic in the event, you'll be prompted to do so. Once you see yourself on screen, you're live to your audience.

## To share your screen:

- Click the Screen sharing icon at the bottom of the page
- Switch to Chrome Tab on the popup window and choose the required browser tab
- Check the Share audio box
- Click Share to start sharing

**Your session will be automatically recorded.**

*\* Please note that this slide is for speakers only. Please feel free to delete this slide before sharing.*

# MiraclePtr

the UaF slayer

Keishi Hattori, Bartek Nowierski

Google

May 18th, 2022

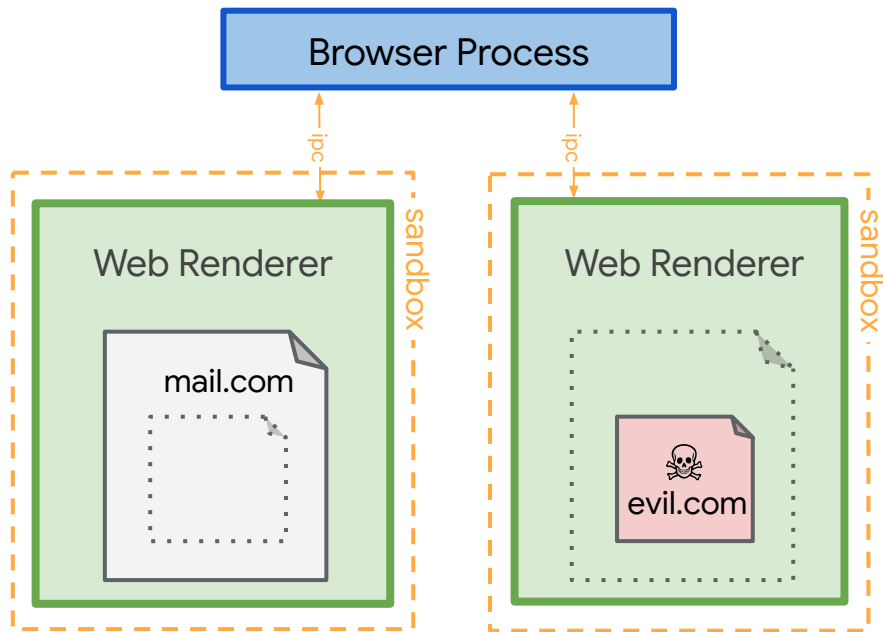


# Impact of UaF (Use-after-Free)

# Renderer sandbox

Sandbox + Site Isolation:

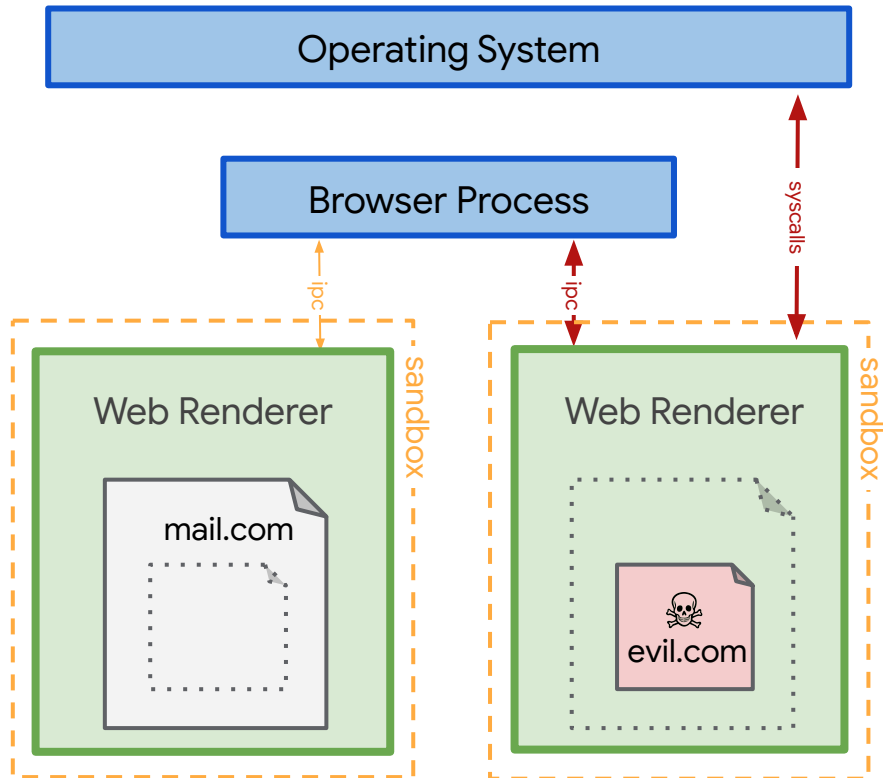
- Renderer has no access to cross-site data



# Sandbox escape routes

## Sandbox + Site Isolation:

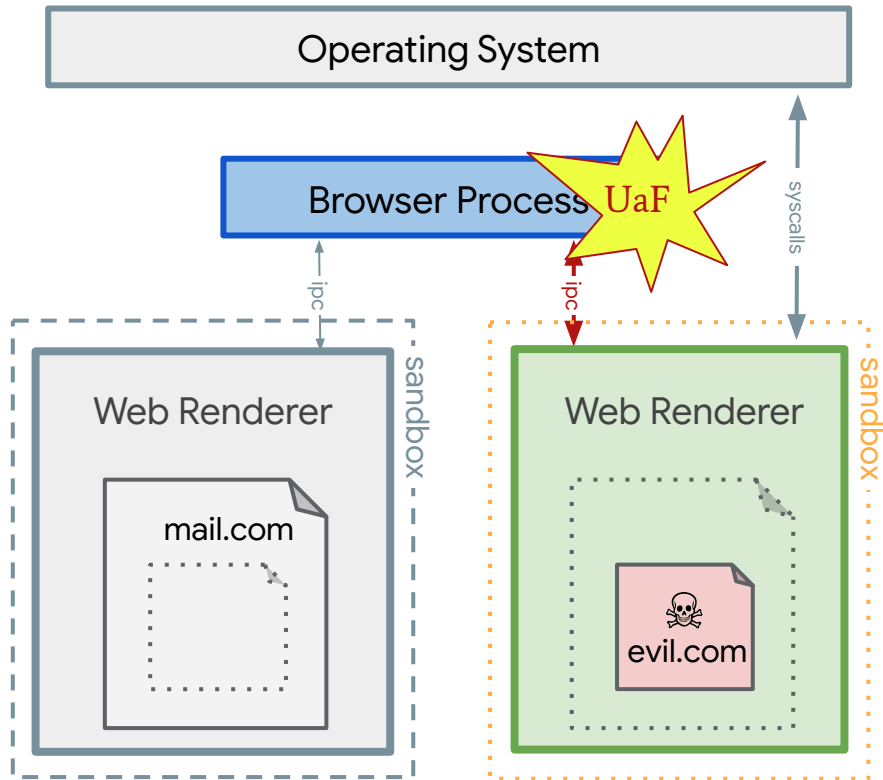
- Renderer has no access to cross-site data
- Unless it can escape the sandbox
  - By attacking the OS/kernel
  - Or the Browser Process



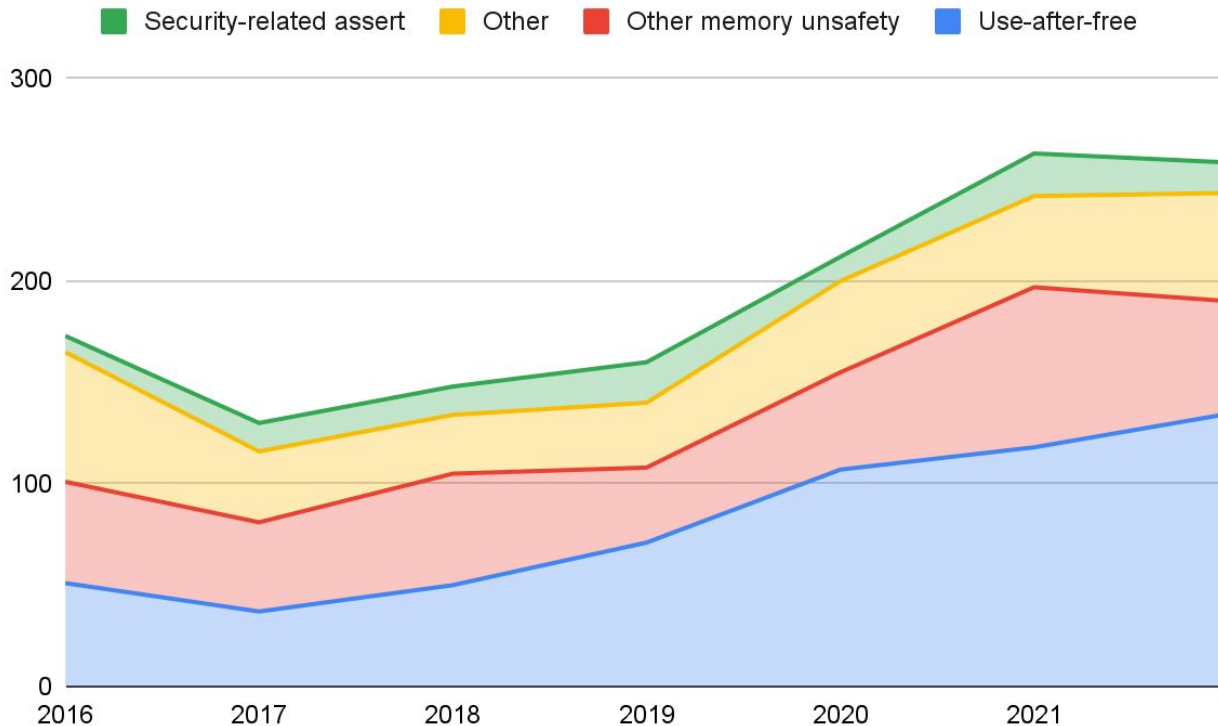
# UaF in the Browser Process = a sandbox escape

## Sandbox + Site Isolation:

- Renderer has no access to cross-site data
- Unless it can escape the sandbox
  - By attacking the OS/kernel
  - Or the Browser Process via **use-after-free (UaF)**
- UaF bugs = high security severity
  - Assuming they *can* be exploited
  - For more details see [an example](#) from Project Zero



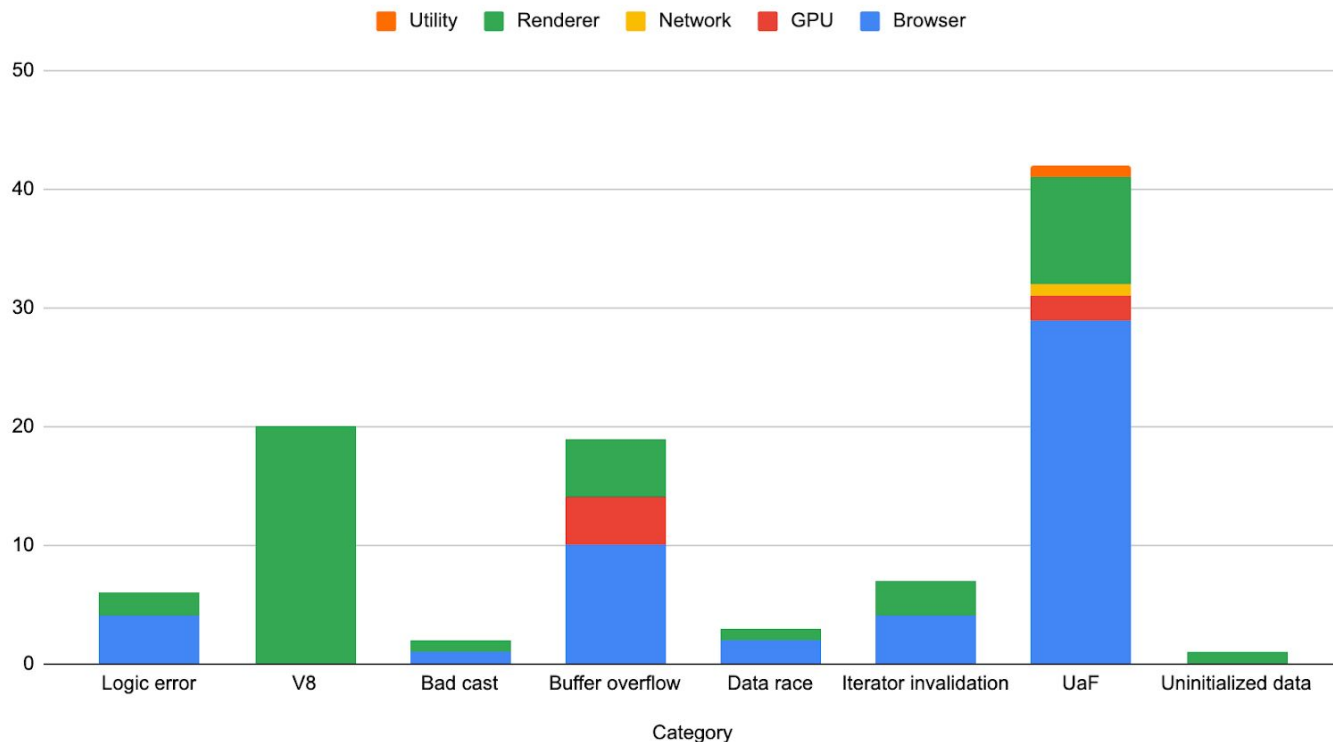
# High+ severity bugs impacting Stable



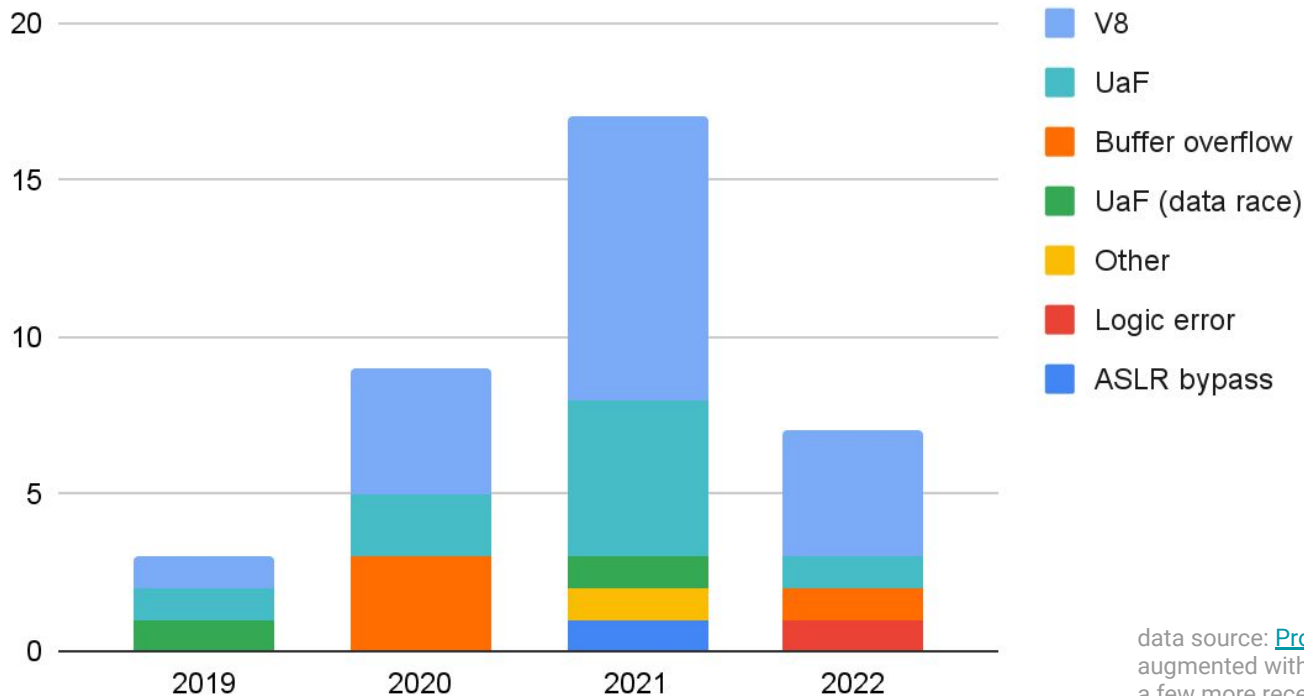
2022 figures are extrapolated from early November to the whole year



# High+ severity bugs impacting Stable



# In-the-wild 0-day exploited bugs



data source: [Project Zero](#),  
augmented with metadata and  
a few more recent bugs

MiraclePtr to the rescue!

# A flow of a UaF attack

Attack outline:


- Conjure a dangling pointer
- Inject attacker-controlled data into the same memory
- Dereference the dangling pointer

# A flow of a UaF attack

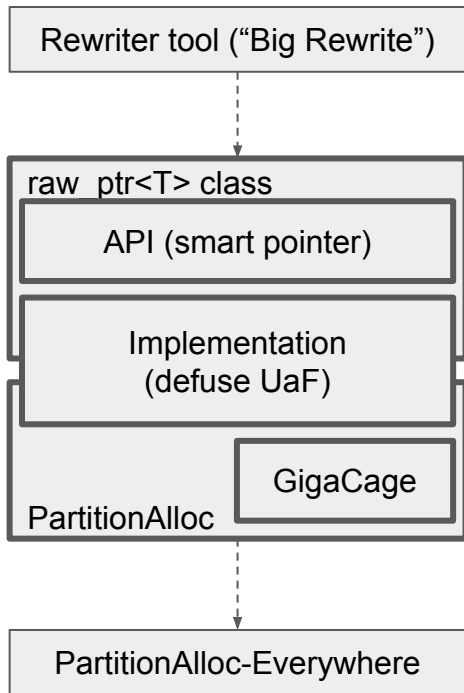
## Attack outline:

- Conjure a dangling pointer
- Inject attacker-controlled data into the same memory
  - **Disruption technique: Don't release the memory**  
(as long as dangling pointers exist)
    - “free” is postponed, but timing of destructors is not affected
- Dereference the dangling pointer

*This is where  
we currently  
focus our efforts*



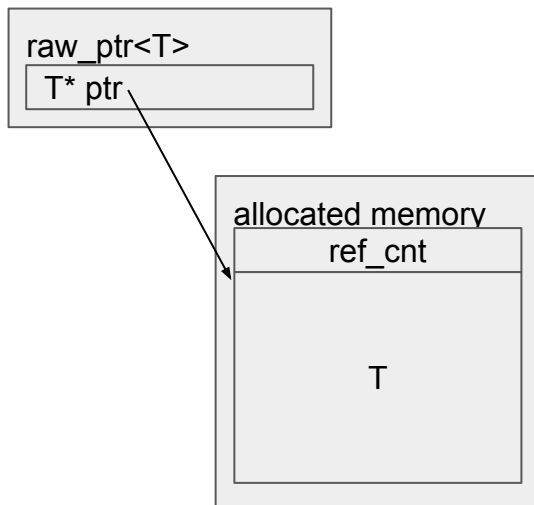
# How MiraclePtr works



- `raw_ptr<T>` API is similar to other smart pointers
  - current implementation in [base/memory/raw\\_ptr.h](#)
  - but `|.get()|` is usually not needed thanks to implicit casts
  - unlike e.g. `unique_ptr<T>`, doesn't manage lifetime
- `raw_ptr<T>` implementation
  - needs `PartitionAlloc` support
  - GigaCage tells if memory is allocated from `PartitionAlloc` (as opposed to Oilpan, stack, ...)
- Rewriter tool rewrites `T*` to `raw_ptr<T>`
  - only rewriting raw pointer fields (exclusions apply)
  - "Big Rewrite" rewrote 15+k pointer fields in Chromium
- Multiple other algorithms were considered  
[go/designdoc\\_PointerSafety](#)

# How MiraclePtr works

BackupRefPtr algorithm:



- `++ref_cnt` in `raw_ptr<T>` ctor or assignment
- `--ref_cnt` in `raw_ptr<T>` dtor or deassignment
- `Free()`: if `ref_cnt > 0`
  - quarantines memory
  - poisons memory
    - a future read may hopefully lead to a crash
  - the last `raw_ptr<T>` dtor will release memory

# Scope and focus of MiraclePtr

- Preventing exploitation
  - not detecting (though detection techniques are under development)
- Deterministic prevention
  - not just making it probabilistically harder
- Targeting all platforms
  - depends on PartitionAlloc-Everywhere (currently missing on iOS)
- Targeting mainly non-Renderer processes
  - perf impact on Renderer may be prohibitive, we have to narrow down scope significantly
- class/struct fields only
  - on-stack pointer protection extension is under consideration



# Performance

# Performance evaluation

We knew from the get go that MiraclePtr will cost some performance to achieve the extra security.

We have been running experiments for over an year to find out if this would be acceptable. Evaluating “Big Rewrite” required developing new ways of running performance experiments.

The full analysis is in [BackupRefPtr M98/M99 experiment results](#) (sorry Google only). I will share the rundown in percentages here.

# CPU Overhead

Most of the overhead comes from ref count manipulation.

`raw_ptr<T>` incurs additional runtime overhead for initialization, destruction, and assignment. There is no overhead when dereferencing or extracting a pointer.

On Windows, the only change to our guiding metrics is a regression to `JankyIntervalsPerThirtySeconds`.

On Android, in addition to `JankyIntervalsPerThirtySeconds`, `First Input Delay` regresses around 1%.

The decision was that the regressions would be acceptable and the impact to the user would be small, especially since `MiraclePtr` is not adding new long tasks.

# Memory Overhead

	Windows	Android
Browser process PMF	50P +6.6% ◆◆◆◆◆ 95P +5.7% ◆◆◆◆◆ 99P +4.4% ◆◆◇◇	50P +5.2% ◆◆◆◆◆ 95P +4.4% ◆◆◆◆◆ 99P +3.4% ◆◆◆◆◆
Total PMF	50P +1.0% ◆◆◇◇	50P +2.7% ◆◆◆◆◆ 95P +1.1% ◆◆◆◆◆ 99P +0.8% ◆◆◆◆◆

PMF=[Private Memory Footprint](#), ◆ is significance level determined by p-value

The regressions are not small but was decided the tradeoff to be worth it, for the security benefit. Also we did check that Android OOM crash rates did not change.

# Memory Overhead

There are three sources of memory overhead

- **Quarantined memory**

Memory referenced by a dangling pointer is “quarantined”, meaning it is reserved even after free()

- **Fragmentation caused by splitting the main PartitionAlloc partition**

Because we have to enable MiraclePtr for just the browser process, we had to prepare a separate partition for allocations before a bit of process initialization.

- **Ref count added to each allocation**

# Memory Overhead

## Quarantined memory

We had no idea how much this would be in the real world.

So we added a metric

`Memory.Experimental.Browser2.Small.Malloc.BRPQuarantined`

This was much smaller than expected.

It accounted for less than +0.01% (Browser PMF)

# Memory Overhead

## **Fragmentation caused by splitting the main PartitionAlloc partition**

We prepared a special experiment group that split the partitions but kept MiraclePtr disabled.

Less than +1% on Windows and around +1-2.5% on Android (Browser PMF)

# Memory Overhead

## Ref count added to each allocation

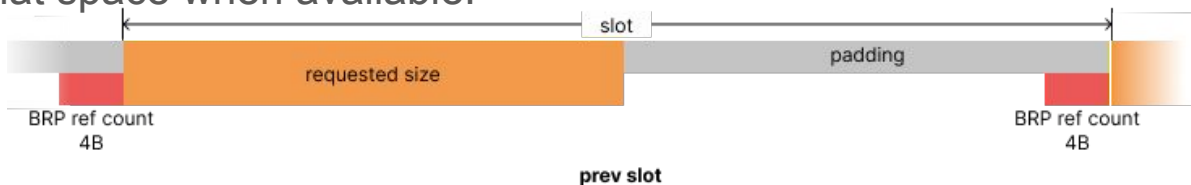
We tried two ways of storing the ref count.

“**before alloc**” is the naive solution of having the ref count at the beginning of the allocation. Ref count is 4 bytes, but due to alignment, we need to add 16 bytes to every allocation.



✓ “**prev slot**” places the ref count in the padding of the previous slot. Because PartitionAlloc is a bucket-based memory allocator, allocations often have wasted padding space at the end.

“Prev slot” uses that space when available.





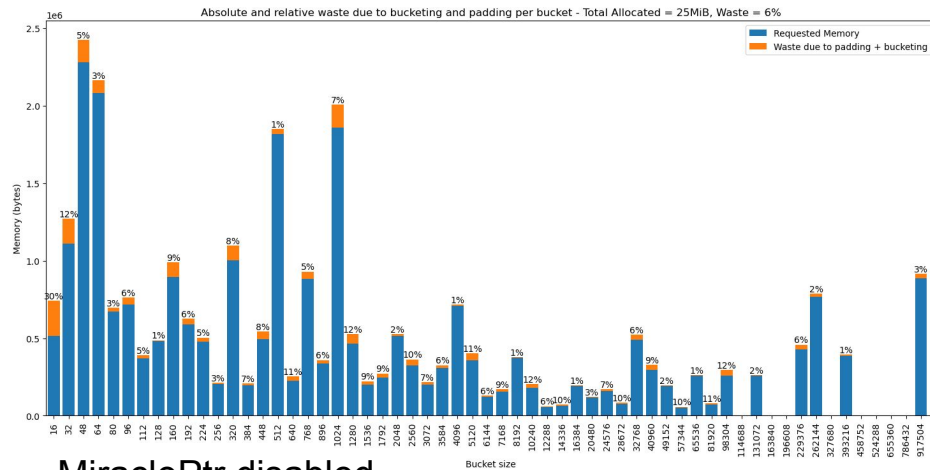
# Memory Overhead

## Ref count added to each allocation

Prev slot puts ref count in the wasted padding space so we didn't think this would be so big.

But it turned out this is the majority of overhead.

- There are a lot of small allocations.
- There are a lot of allocations that have a size of exactly power of two.



raw\_ptr<T> usage

# raw\_ptr<T>

Please use raw\_ptr<T> for class and struct fields in place of a raw C++ pointer T\* whenever possible, except in Renderer-only code.

Details are available in the [Usage Guideline](#).

# Clang plugin enforcement coming soon...

We are developing a clang plugin that will show compiler errors when `raw_ptr<T>` is not used.

## Example of error

```
../../../../base/trace_event/traced_value.h(103,18): error:  
[chromium-rawptr] Use raw_ptr<T> instead of a raw pointer.  
TracedValue* value_;
```

## Example of exclusion

```
struct _HEAP_32 {  
    // `Heap` is not a raw_ptr<...>, because reinterpret_cast of uninitialized  
    // memory to raw_ptr can cause ref-counting mismatch.  
    RAW_PTR_EXCLUSION struct _HEAP_32* Heap;  
};
```

# Roadmap

# Roadmap update (v1)

- ✓ BackupRefPtr implemented and thoroughly tested
- ✓ PartitionAlloc-Everywhere now almost everywhere
- ✓ Evaluate overhead that can't be tested via field trials (binary A/B experiments)
- ✓ Land the "Big Rewrite" ( $T^* \rightarrow \text{raw\_ptr}<T>$ )
- ✓ Evaluate overhead that can be tested using field trials
- ↻ Enable MiraclePtr protection
- ↻ Integrate with ASan
- ↻ Enable clang plugin to enforce  $\text{raw\_ptr}<T>$  usage  
(per-pointer opt-out option available)



done







in progress





not started (but planned)

# Roadmap update (beyond v1)

Increase coverage:

-  Reference field support (`raw_ref<T>`)
-  On-stack pointer support (local variable/param protection)
-  Pointers inside templated collections
-  `raw_ptr<const char>`

Increase debuggability:

-  MTE-like algorithm (likely debug builds only)
-  Dangling pointer detector



done



in progress



not started (but planned)

Thank you



# Thank you to everyone on the team!

- MiraclePtr implementation & testing: bartekn@, keishi@, glazunov@, tasak@, lukasza@, yukiy@, lizeb@, arthursonzogni@
- Security advisors: glazunov@, markbrand@, adetaylor@, lukasza@, tsepez@, danakj@, palmer@
- Release: sebmachand@, srinivassista@, govind@, benmason@, pbommana@
- PM: dharani@
- Shepherding the project: haraken@

Questions?