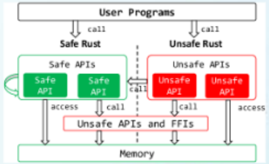
STOP DOING MEMORY SAFETY RESEARCH

- BOUNDS WERE NEVER MEANT TO BE CHECKED
- YEARS OF **PAPERS** yet NO REAL-WORLD **ADOPTION** for **BOUNDS CHECKERS**
- Wanted to **PREVENT DANGLING POINTERS** ? We had a tool for that: It was called **DON'T FREE YOUR MEMORY**
- "Yes please give me **PERLBENCH COMPATIBILITY** Please give me **"CAPABILITIES"** - Statements dreamed up by evil wizards

LOOK at what **PHD STUDENTS** have been demanding your Respect for all this time, with all the **CPU BENCHMARKS** we built for them

(This is REAL RESEARCH done by REAL ACADEMICS):



?????

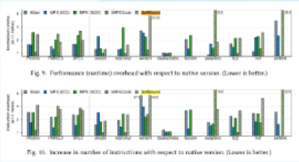
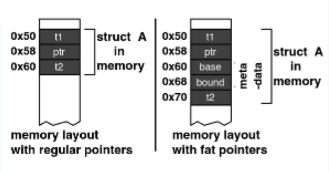


Fig. 9. Performance (runtime) overhead with respect to native version. (Lower is better.)

Fig. 10. Increase in number of instructions with respect to native version. (Lower is better.)

???????



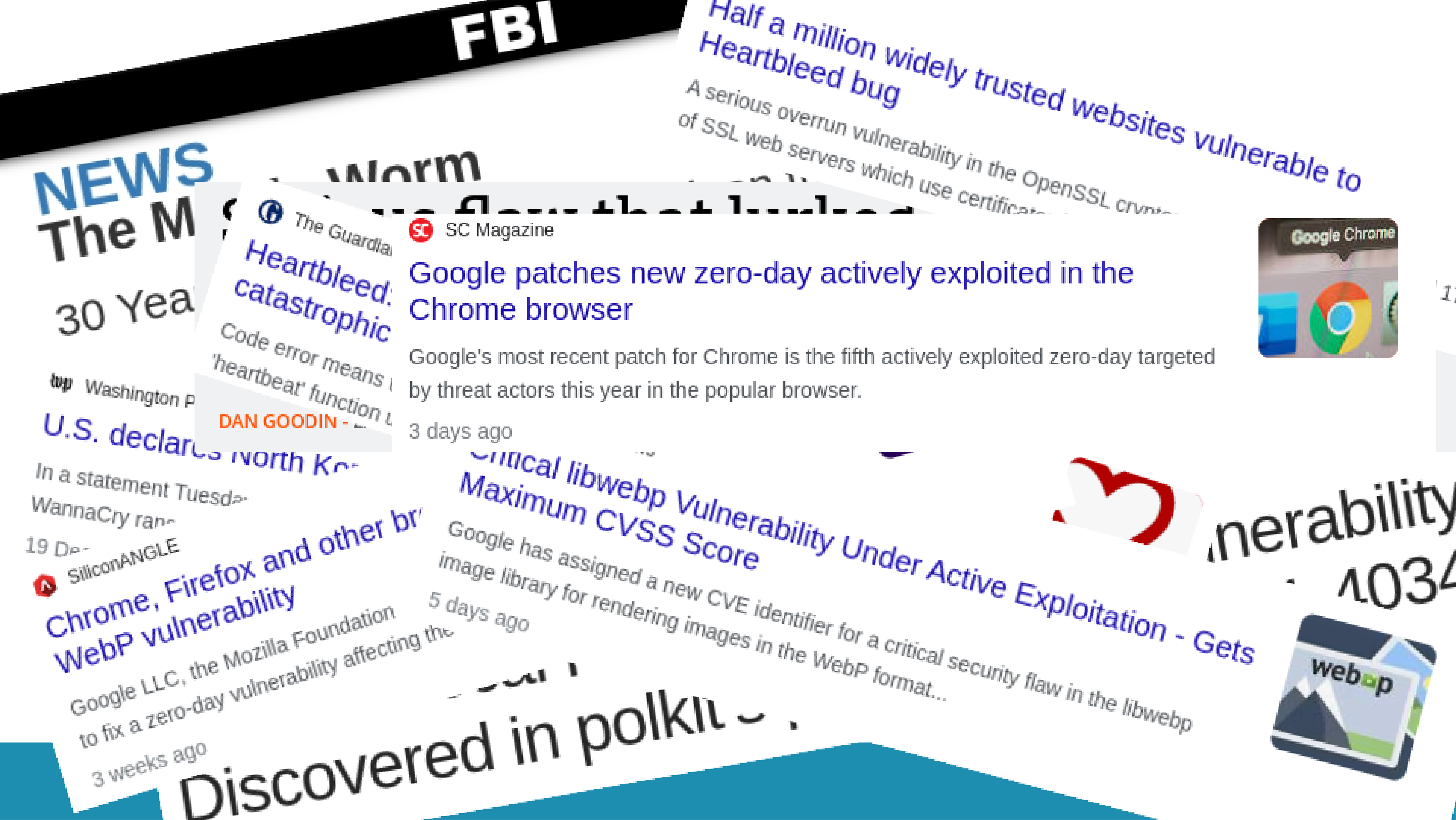
?????????????????

"Hello I would like `int *__counted_by(count) p` please"

They have played us for absolute fools

Not Quite Write: On the Effectiveness of Store-Only Bounds Checking

Adriaan Jacobs, Stijn Volckaert



FBI

NEWS

The M... Worm

30 Yea...

Half a million widely trusted websites vulnerable to Heartbleed bug

A serious overrun vulnerability in the OpenSSL crypto... of SSL web servers which use certificat...

The Guardian

Heartbleed: ... catastrophic

Code error means ... 'heartbeat' function ...

DAN GOODIN -

SC Magazine

Google patches new zero-day actively exploited in the Chrome browser

Google's most recent patch for Chrome is the fifth actively exploited zero-day targeted by threat actors this year in the popular browser.

3 days ago

Washington P...

U.S. declares North Ko...

In a statement Tuesda... WannaCry ran...

19 De...

SiliconANGLE

Chrome, Firefox and other br... WebP vulnerability

Google LLC, the Mozilla Foundation ... to fix a zero-day vulnerability affecting the...

3 weeks ago

Critical libwebp Vulnerability Under Active Exploitation - Gets Maximum CVSS Score

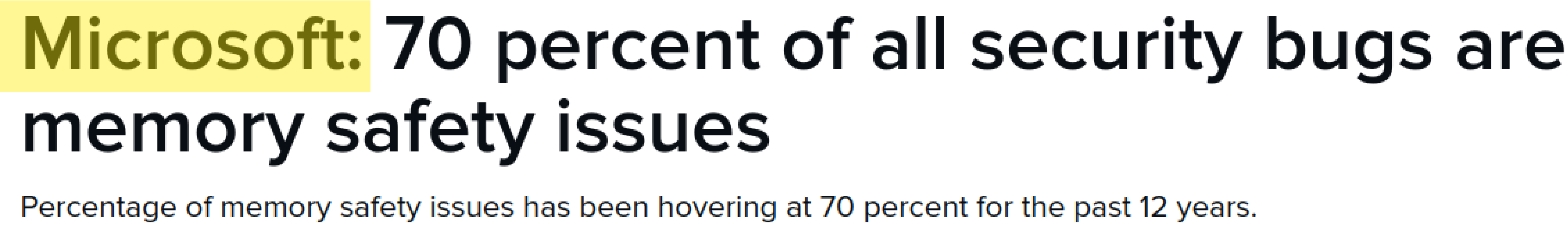
Google has assigned a new CVE identifier for a critical security flaw in the libwebp image library for rendering images in the WebP format...

5 days ago

...nerability

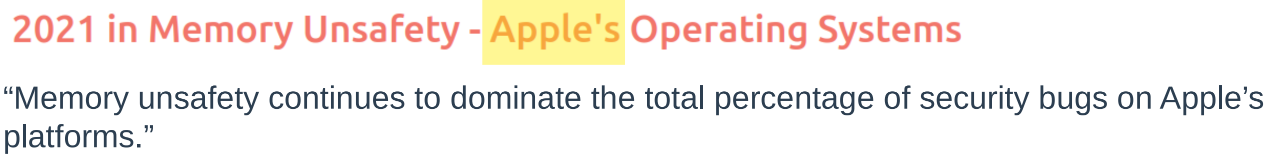
...4034

Discovered in polkit



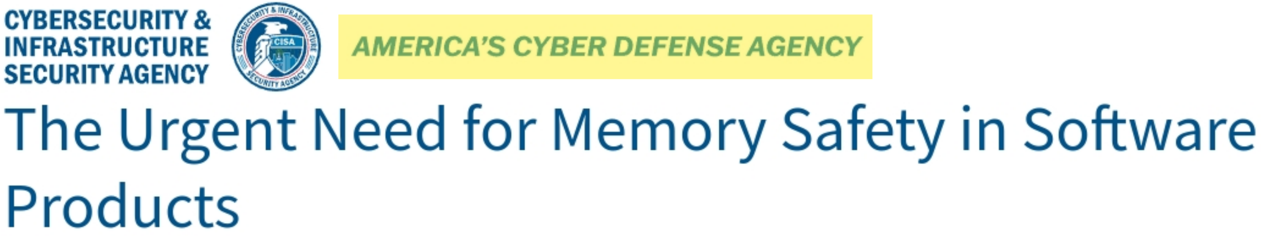
Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.



2021 in Memory Unsafety - Apple's Operating Systems

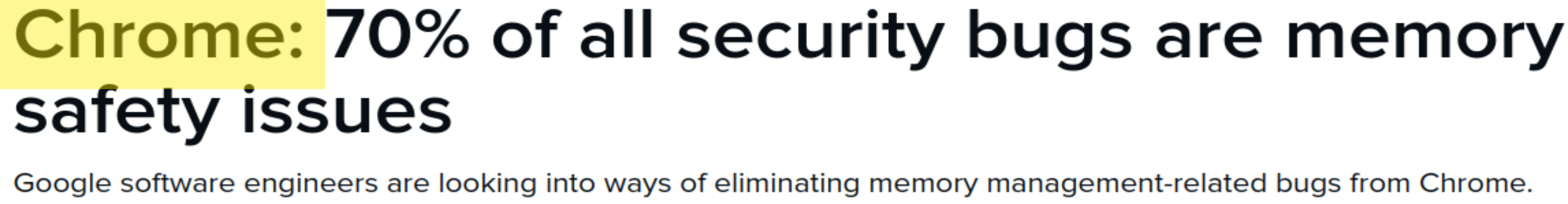
“Memory unsafety continues to dominate the total percentage of security bugs on Apple’s platforms.”



CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

AMERICA’S CYBER DEFENSE AGENCY

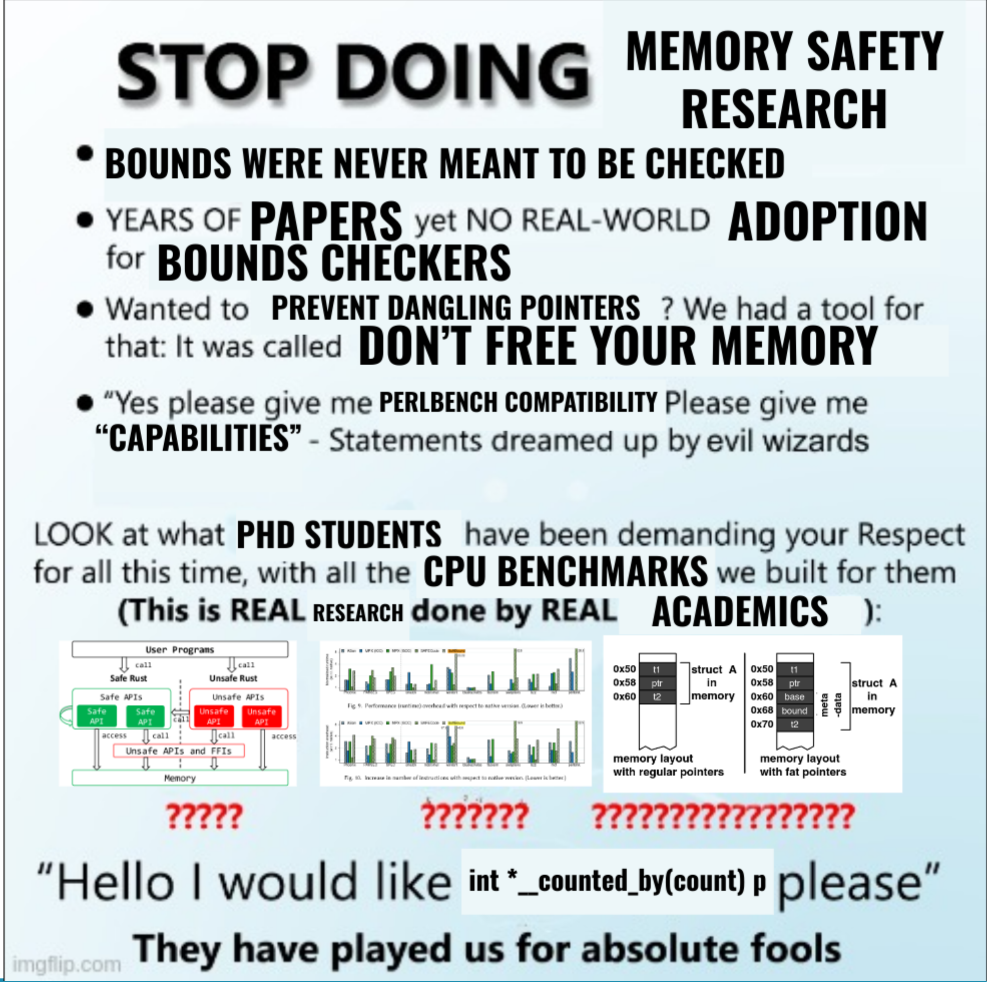
The Urgent Need for Memory Safety in Software Products



Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

Why Haven’t We Solved This Problem Yet?



STOP DOING MEMORY SAFETY RESEARCH

- BOUNDS WERE NEVER MEANT TO BE CHECKED
- YEARS OF PAPERS yet NO REAL-WORLD ADOPTION for BOUNDS CHECKERS
- Wanted to PREVENT DANGLING POINTERS ? We had a tool for that: It was called DON’T FREE YOUR MEMORY
- “Yes please give me PERLBENCH COMPATIBILITY Please give me “CAPABILITIES” - Statements dreamed up by evil wizards

LOOK at what PHD STUDENTS have been demanding your Respect for all this time, with all the CPU BENCHMARKS we built for them (This is REAL RESEARCH done by REAL ACADEMICS):

?????   ???????   ?????????????????

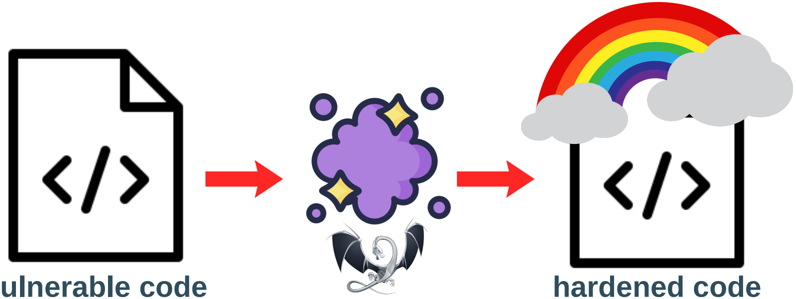
“Hello I would like `int *__counted_by(count) p` please”

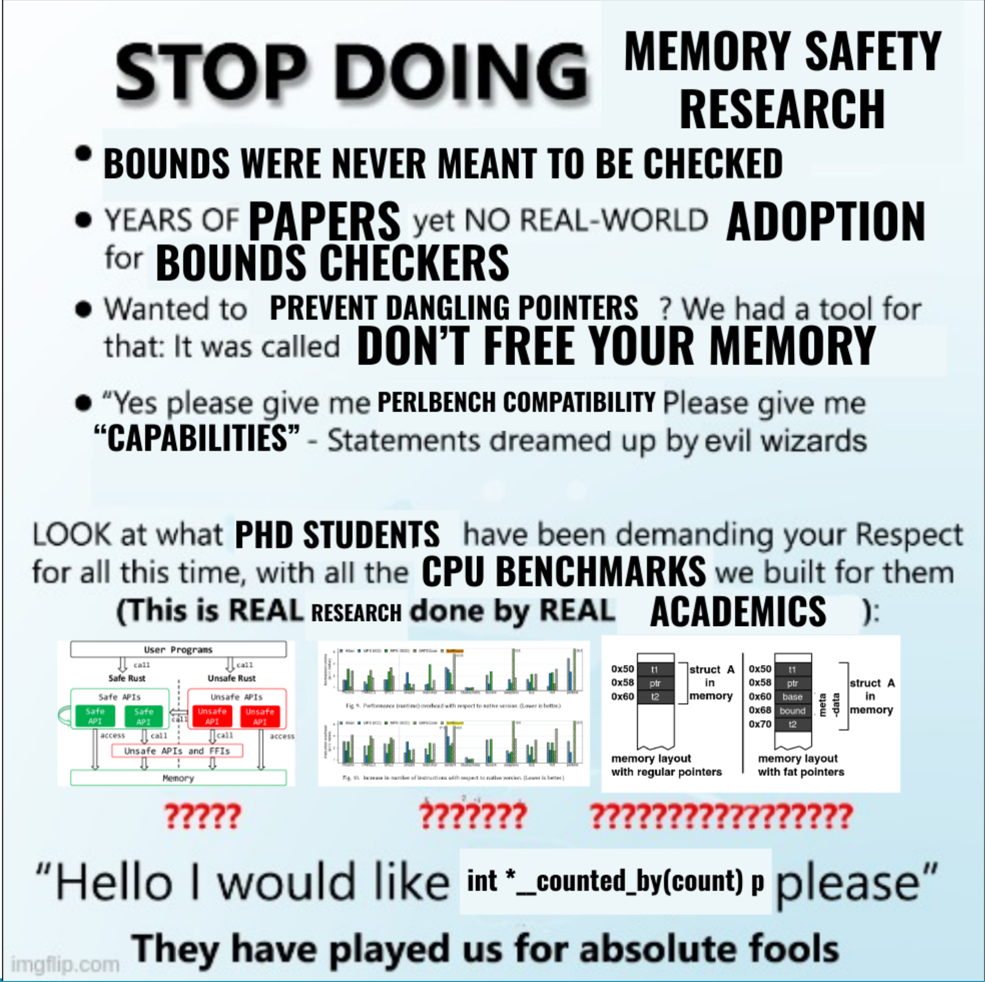
They have played us for absolute fools

imgflip.com

Why Haven’t We Solved This Problem Yet?

- *Very* frequent checks
- Intrusive instrumentation
- Hard-to-generalize hardware acceleration
- Compatibility with arcane programming practices





STOP DOING MEMORY SAFETY RESEARCH

- BOUNDS WERE NEVER MEANT TO BE CHECKED
- YEARS OF PAPERS yet NO REAL-WORLD ADOPTION for BOUNDS CHECKERS
- Wanted to PREVENT DANGLING POINTERS ? We had a tool for that: It was called DON’T FREE YOUR MEMORY
- “Yes please give me PERLBENCH COMPATIBILITY Please give me “CAPABILITIES” - Statements dreamed up by evil wizards

LOOK at what PHD STUDENTS have been demanding your Respect for all this time, with all the CPU BENCHMARKS we built for them (This is REAL RESEARCH done by REAL ACADEMICS):

User Programs
call
call
Safe Rust
Unsafe Rust
Safe APIs
Safe API
Safe API
Unsafe APIs
Unsafe API
Unsafe API
call
access
call
call
access
Unsafe APIs and FFIs
Memory
Fig. 9. Performance (runtime) overhead with respect to native version. (Lower is better.)
Fig. 10. Increase in number of instructions with respect to native version. (Lower is better.)
0x50
0x58
0x60
t1
ptr
t2
struct A in memory
memory layout with regular pointers
0x50
0x58
0x60
0x68
0x70
t1
ptr
base
bound
t2
meta -data
struct A in memory
memory layout with fat pointers

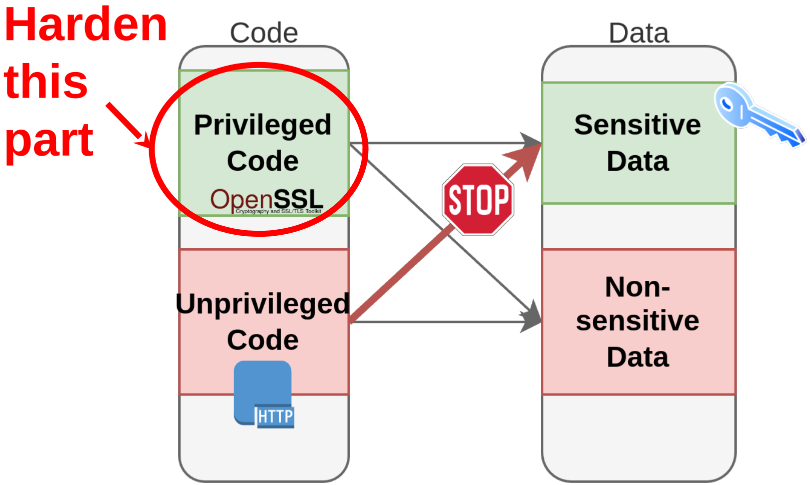
????? ??????? ??????????????????

“Hello I would like int *_counted_by(count) p please”

They have played us for absolute fools

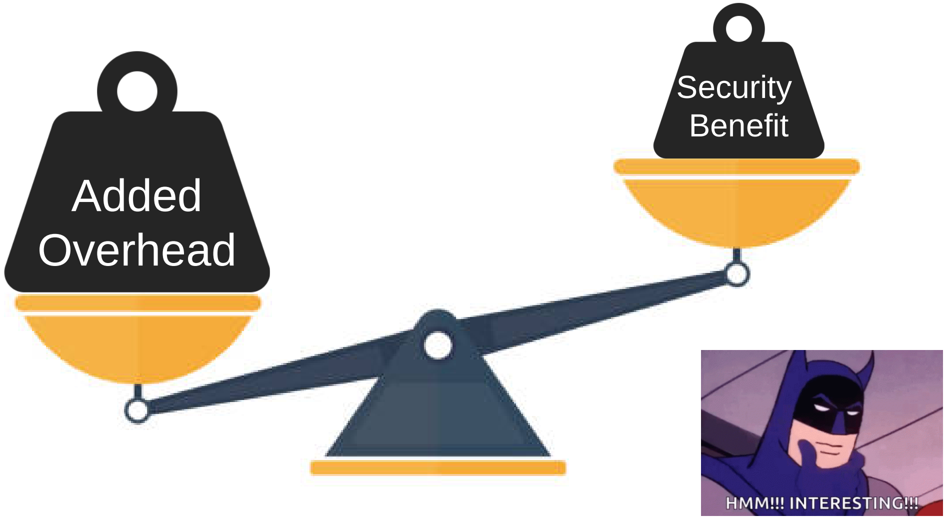
Partial Bounds Checking

Prioritize Security-Critical Code/Data



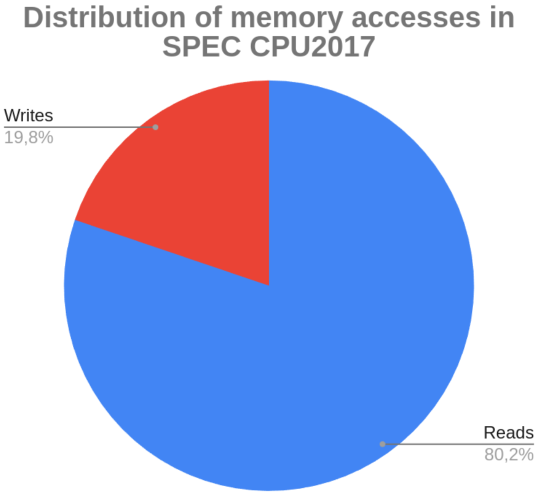
E.g., DataShield (AsiaCCS'17), OAT (S&P'20)

De-prioritize Costly Checks



E.g., ASAP (S&P'15), store-only bounds checking

Store-Only Bounds Checking

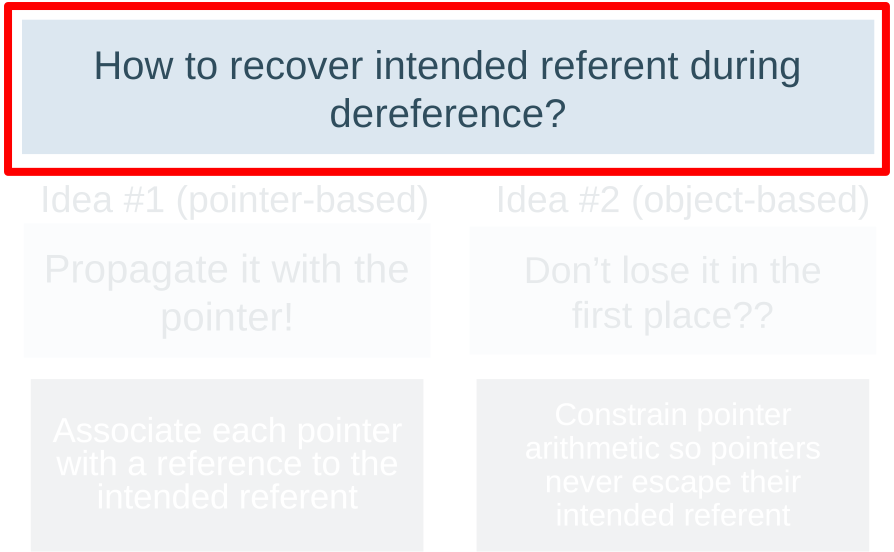


- Invalid **writes** are necessary for many attacks
  - Except pure information disclosure, e.g., Heartbleed
- Memory **writes** occur far **less frequently** than **reads**

> **“Store-only checking [...] is sufficient to prevent all memory corruption-based security vulnerabilities.”**

- Nagarakatte et al.

Bounds Checkers Demystified



How to recover intended referent during dereference?

Idea #1 (pointer-based)

Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent

```
void* ptr = malloc(...);
// ...
*ptr = ...;
```

intended referent

Bounds Checkers Demystified

How to recover intended referent during dereference?

Idea #1 (pointer-based)

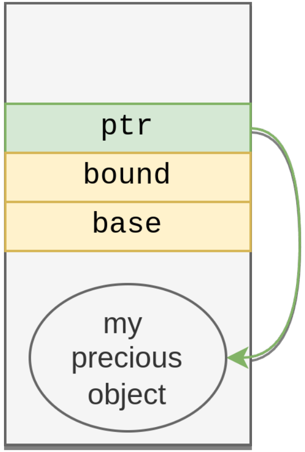
Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent



```
if (ptr < base || ptr > bound)
        exit();
*ptr = ...;
```

Bounds Checkers Demystified

How to recover intended referent during dereference?

Idea #1 (pointer-based)

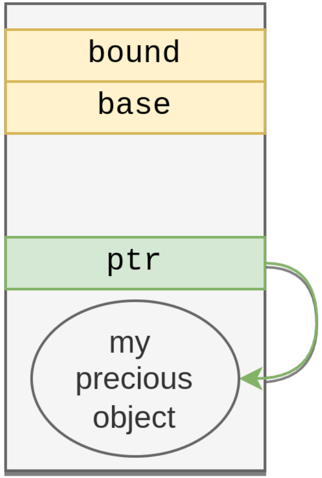
Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent



```
if (ptr < base || ptr > bound)
       exit();
*ptr = ...;
```

Bounds Checkers Demystified

How to recover intended referent during dereference?

Idea #1 (pointer-based)

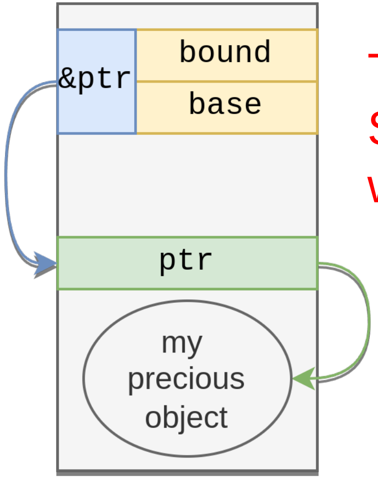
Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent



This is how SoftBound works

```
if (ptr < base || ptr > bound)
       exit();
*ptr = ...;
```

Bounds Checkers Demystified

How to recover intended referent during dereference?

Idea #1 (pointer-based)

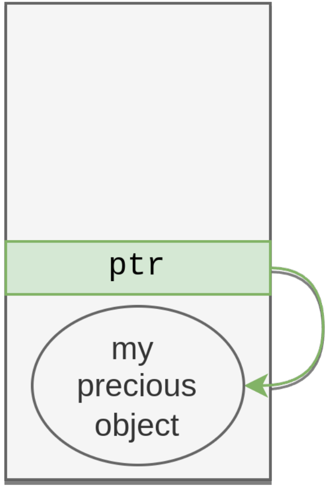
Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent



```
ptr += offset;
if (ptr < base || ptr > bound)
       exit();
```

Bounds Checkers Demystified

How to recover intended referent during dereference?

Idea #1 (pointer-based)

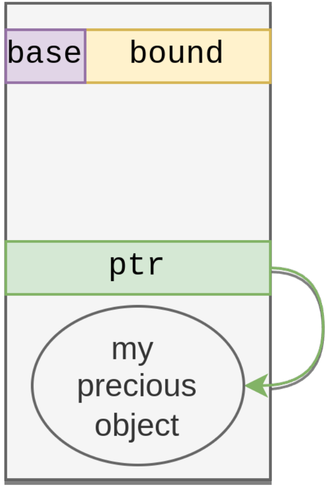
Propagate it with the pointer!

Associate each pointer with a reference to the intended referent

Idea #2 (object-based)

Don’t lose it in the first place??

Constrain pointer arithmetic so pointers never escape their intended referent

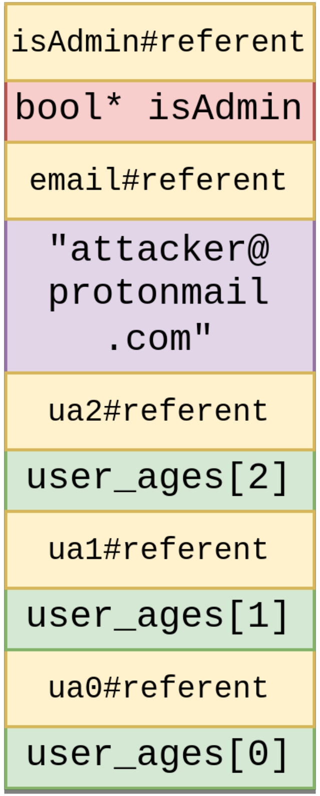


```
ptr += offset;
if (ptr < base || ptr > bound)
        exit();
```

Store-Only Bounds Checking

```
+ referent = *lookup_for(&user_ages[i]);
  int* user_age = user_ages[i];

+ assert_valid(user_age, referent);
  *user_age = input();
```



isAdmin#referent

bool* isAdmin

email#referent

"attacker@protonmail.com"

ua2#referent

user_ages[2]

ua1#referent

user_ages[1]

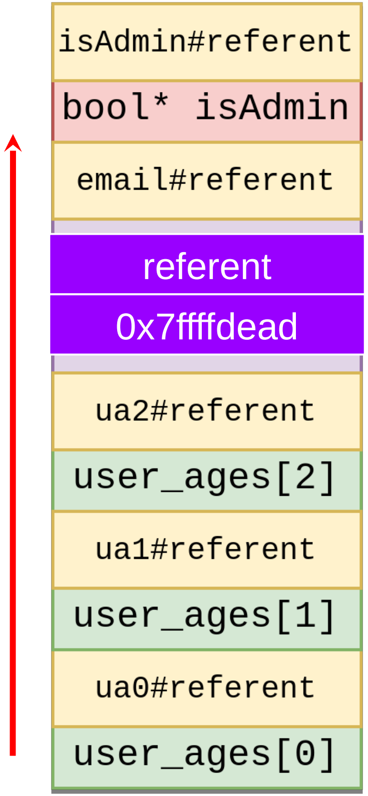
ua0#referent

user_ages[0]

Store-Only Bounds Checking

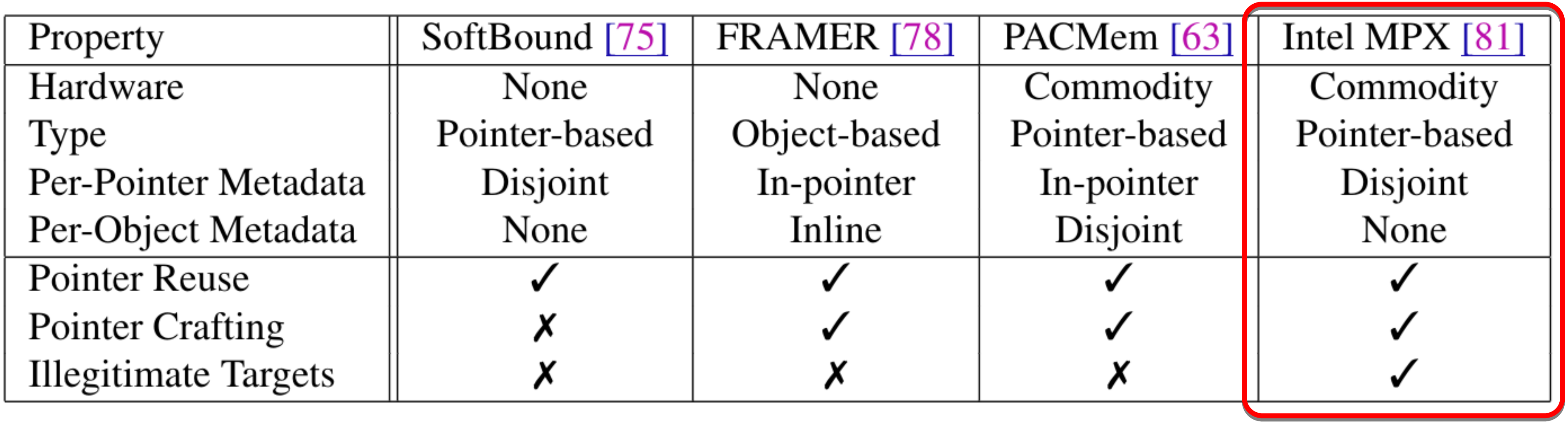
```
- assert_valid(&user_ages[i], user_ages_referent);
+ referent = *lookup_for(&user_ages[i]);
  int* user_age = user_ages[i];

+ assert_valid(user_age, referent);
  *user_age = input();
```



return address?
bounds table?

This Is Not a Design or Implementation Issue



Property	SoftBound [75]	FRAMER [78]	PACMem [63]	Intel MPX [81]
Hardware	None	None	Commodity	Commodity
Type	Pointer-based	Object-based	Pointer-based	Pointer-based
Per-Pointer Metadata	Disjoint	In-pointer	In-pointer	Disjoint
Per-Object Metadata	None	Inline	Disjoint	None
Pointer Reuse	✓	✓	✓	✓
Pointer Crafting	✗	✓	✓	✓
Illegitimate Targets	✗	✗	✗	✓

Who Needs Invalid Writes?

Arbitrary Code Execution

```
func = array[i];
func(args);
```

> “Store-only checking provides much better safety than control-flow integrity with similar performance overheads.”

- Nagarakatte et al.

Who Needs Invalid Writes?

Arbitrary Code Execution

```
func = array[i];
func(args);
```

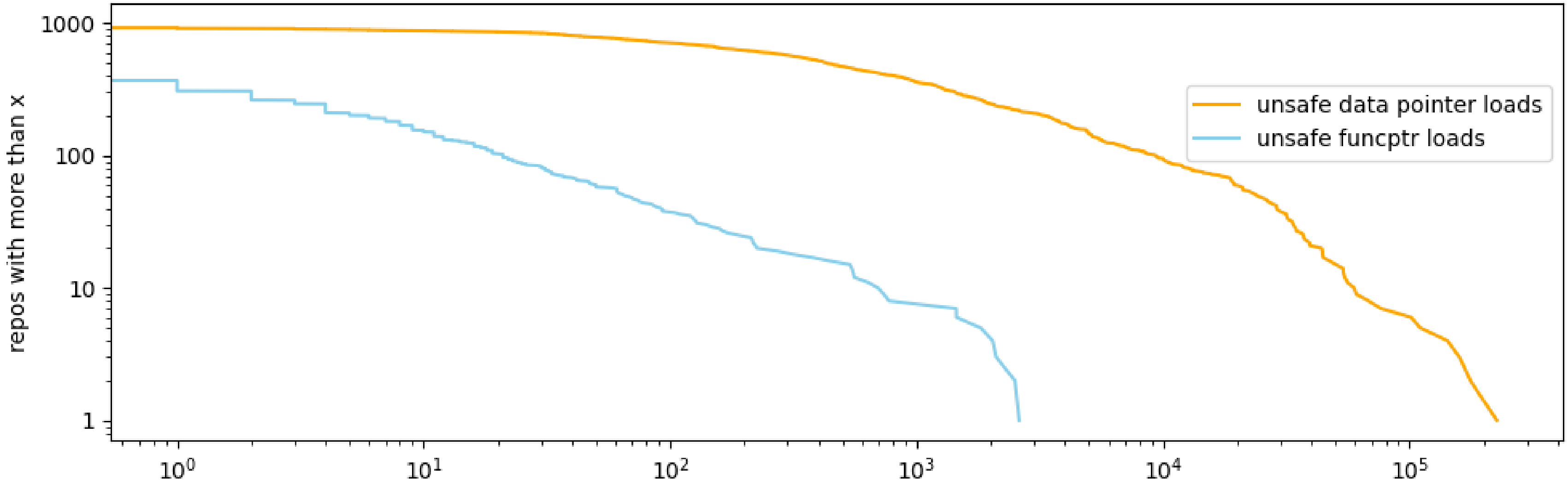
Memory “Corruption”

```
int adminLvl = dangling_ptr->lvl;
if (adminLvl > 2)
    system("/bin/bash");
globalAdminLvl = adminLvl;
```

✓ **Discovery** through invalid reads

✓ **Crafting** in accessible locations

Real-World Feasibility Study on 1,000 GitHub repos



Unsafe data pointer load

```
ptr = array[i];
// ...
*ptr = ...;
```

Unsafe funcptr load

```
ptr = array[i];
// ...
ptr(...);
```

Recap: Why Store-Only Bounds Checking **Fails**

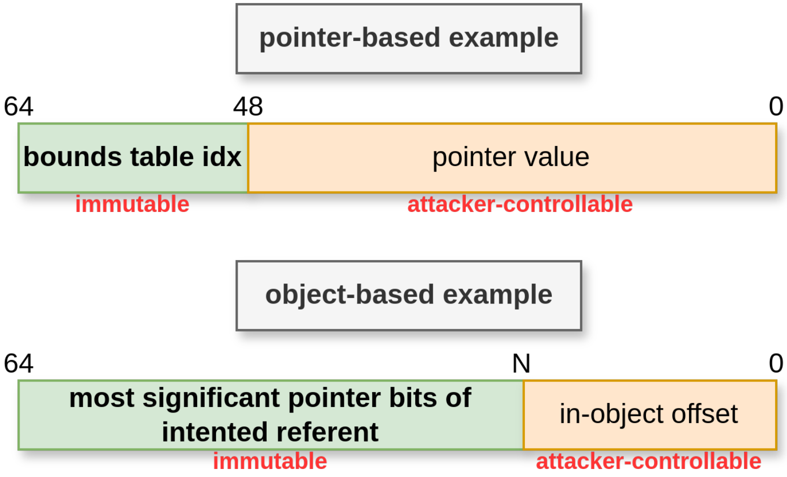
~~Invalid writes are necessary for expressive/severe exploitation~~

~~Store-only bounds checking protects against invalid writes~~

Looking Ahead: Promising Bounds Checking Trend

- Some pointer bits must typically be **immutable** to prevent bypass
  - **"Relative" overwrites via pointer arithmetic:** $ptr_A = ptr_B + (ptr_A - ptr_B)$
- OGs: constrain pointer arithmetic

```
offset &= MASK;
ptr += offset;
```



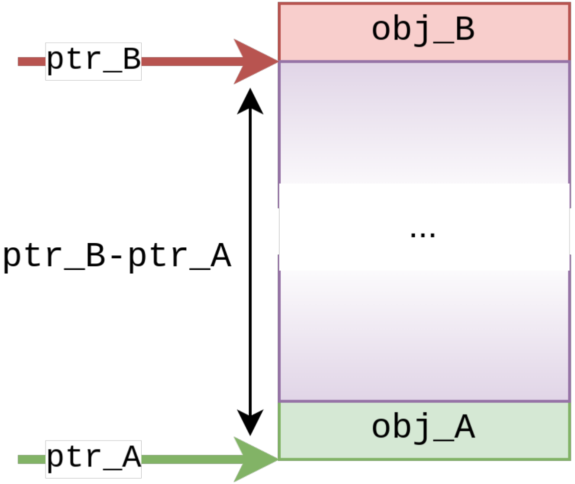
pointer-based example

object-based example

Looking Ahead: Promising Bounds Checking Trend

- Some pointer bits must typically be **immutable** to prevent bypass
  - **“Relative” overwrites via pointer arithmetic:** $ptr_A = ptr_B + (ptr_A - ptr_B)$
- New Age: cryptographic immutability

```
- offset &= MASK;
ptr += offset;
```

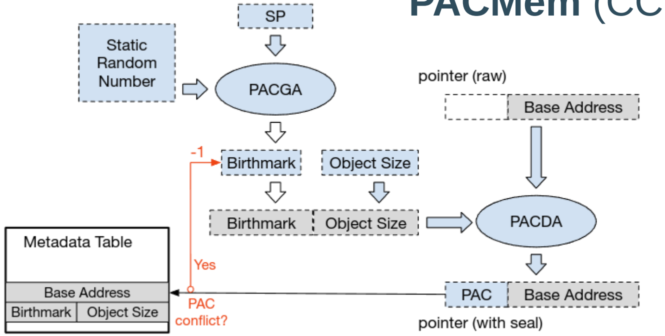


Breaching Pointer Confidentiality

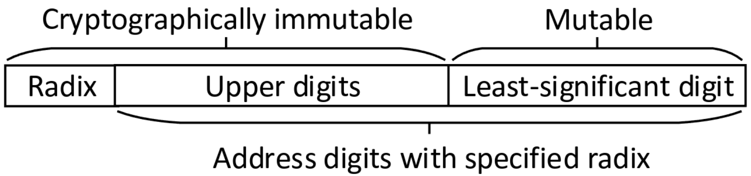
- Lack of pointer arithmetic constraints introduces *implicit* ***pointer secrecy requirement***
- Breached by store-only bounds checkers

GETTING TOTALLY SCREWED BY INFORMATION DISCLOSURE
NEW AND PROMISING BOUNDS CHECKING DESIGNS
STORE-ONLY BOUNDS CHECKING
imgflip.com

PACMem (CCS’22)



C³ (MICRO’21)

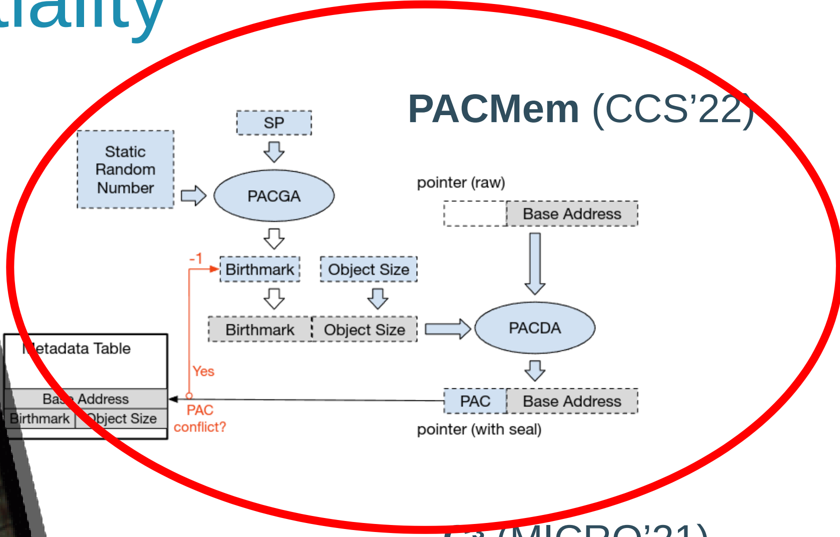


Cryptographically immutable
Mutable
Radix
Upper digits
Least-significant digit
Address digits with specified radix

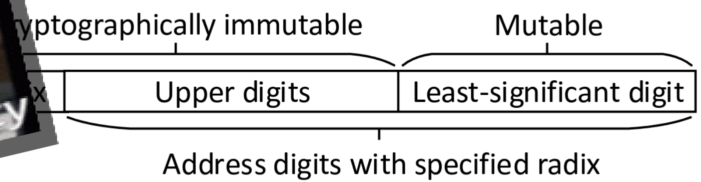
Breaching Pointer Confidentiality

- Lack of pointer arithmetic constraints introduces *implicit **pointer secrecy requirement***
- Breached by sto
  checkers

PACMem (CCS'22)



C^3 (MICRO'21)


yptographically immutable
Mutable
x
Upper digits
Least-significant digit
Address digits with specified radix

But I Still Want Store-Only Protection!

- WIT (S&P'08) computes intended referents ***statically***
- Store-only **testing/fuzzing** is still fine!
- Watch out for bounds checking **optimizations**, **selective** bounds checking, …

Not Quite Write:
On the Effectiveness of Store-Only Bounds Checking

Adriaan Jacobs
DistriNet, KU Leuven

Stijn Volckaert
DistriNet, KU Leuven

Check out the experiments!

Questions?

Read the paper!