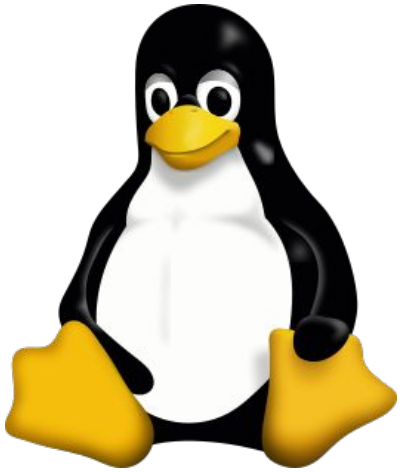
A Decade of Low-hanging Fruit in the Linux Kernel



Kees ("Case") Cook
https://fosstodon.org/@kees
kees@kernel.org

https://outflux.net/slides/2024/bsidespdx/decade.pdf

Hello Neighbors!

Fred Rogers

with apologies to

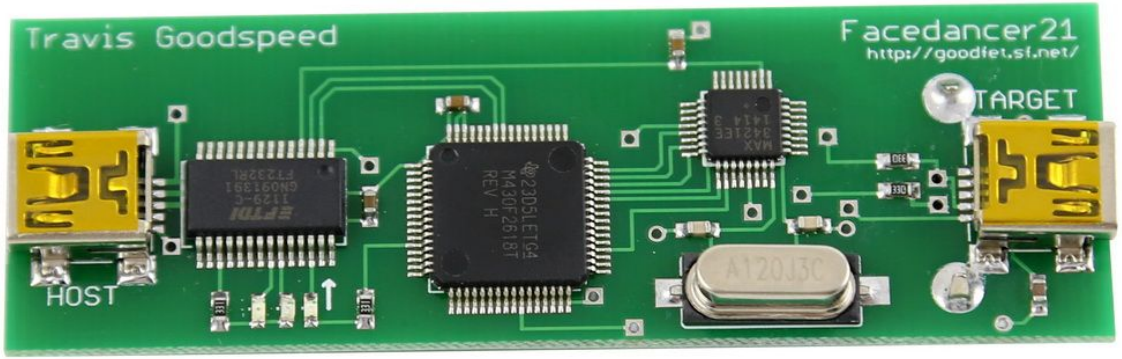
&

Travis Goodspeed

Hello Neighbors!

Fred Rogers

with apologies to



About me

Professionally:

2003 .. 2006: Open Source Development Lab (became the Linux Foundation)

2006 .. 2011: Canonical, Ubuntu Security Team Lead

2011+: Google, Upstream Linux Kernel Security Hardening Lead

chromeOS

Personally:

2002+: Portlander

∞: Free Software Hacker

2006, 2007: DefCon CTF Black Badge winner


BUT YOU DON'T
HAVE TO TAKE
MY WORD FOR IT...

"Seriously, Kees. You are just making security people look bad. Stop it."

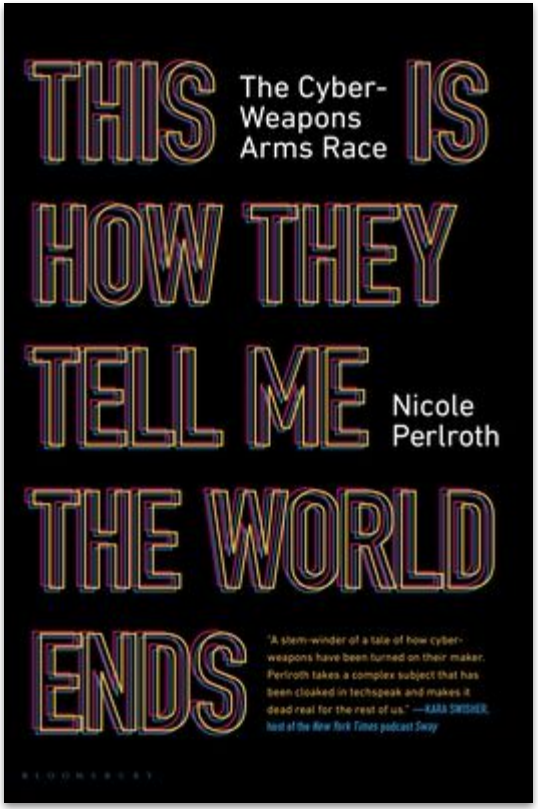
– Linus Torvalds, circa 2017

"Seriously, Kees. You are just making security people look bad. Stop it."

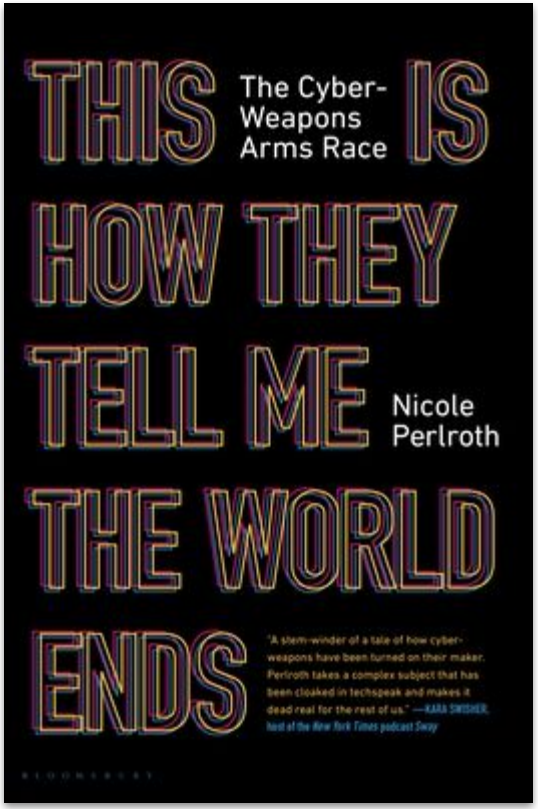
– Linus Torvalds, circa 2017

<Narrator> He did not, in fact, stop it </Narrator>

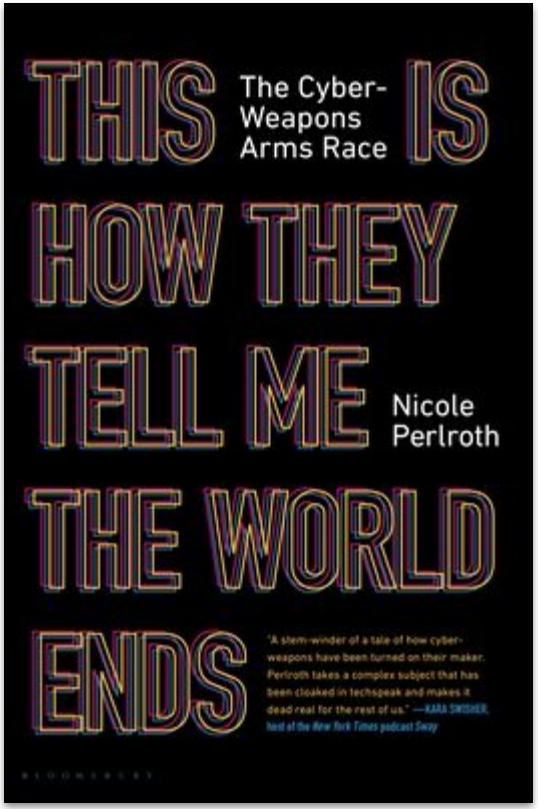
"The most likely way for the world to be destroyed, most experts agree, is by accident.



"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals.

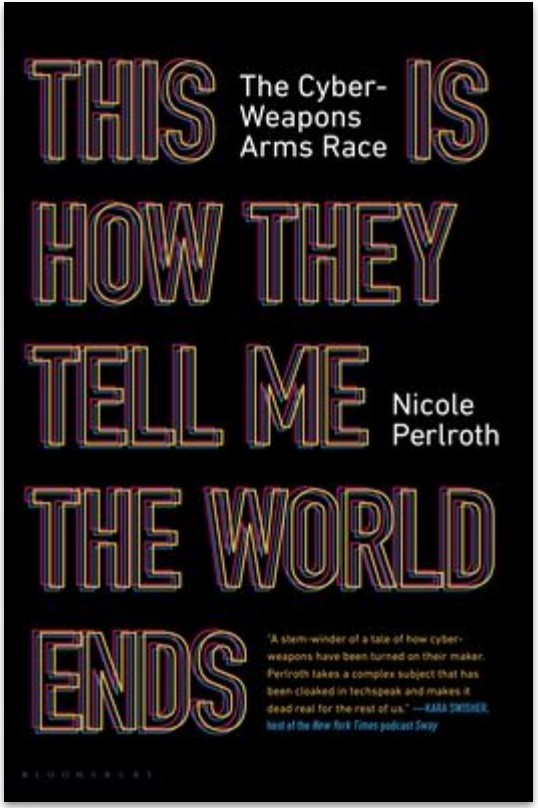


"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals.
We cause accidents."

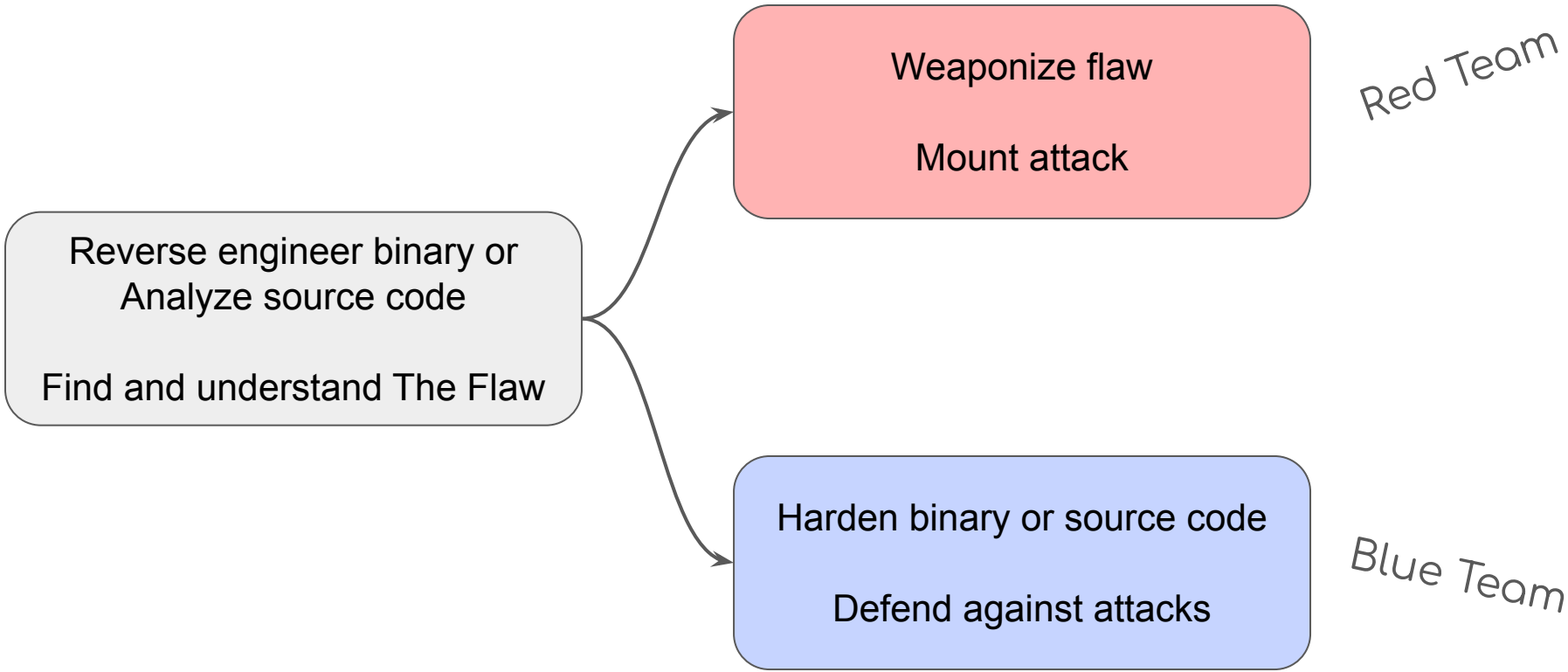


"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals.
We cause accidents."

– Nathaniel Borenstein, of MIME fame (as attributed by Nicole Perlroth)



Practicing Accidents: Capture the Flag



Practicing Accidents: Capture the Flag

Reverse engineer binary or
Analyze source code

Find and understand The Flaw

Weaponize flaw

Mount attack

Harden binary or source code

Defend against attacks

Body Armor: Linux Kernel Self-Protection Project

I announced the project in November 2015 (as an upstream Linux focus area)

Our two specific goals:

- Remove entire bug classes (stop the whack-a-mole of fixing individual bugs)
- Eliminate exploitation methods (don't make things easy for attackers)

It's been almost 10 years of cat herding!

Have things improved?

Let's look at vulnerability trends ...

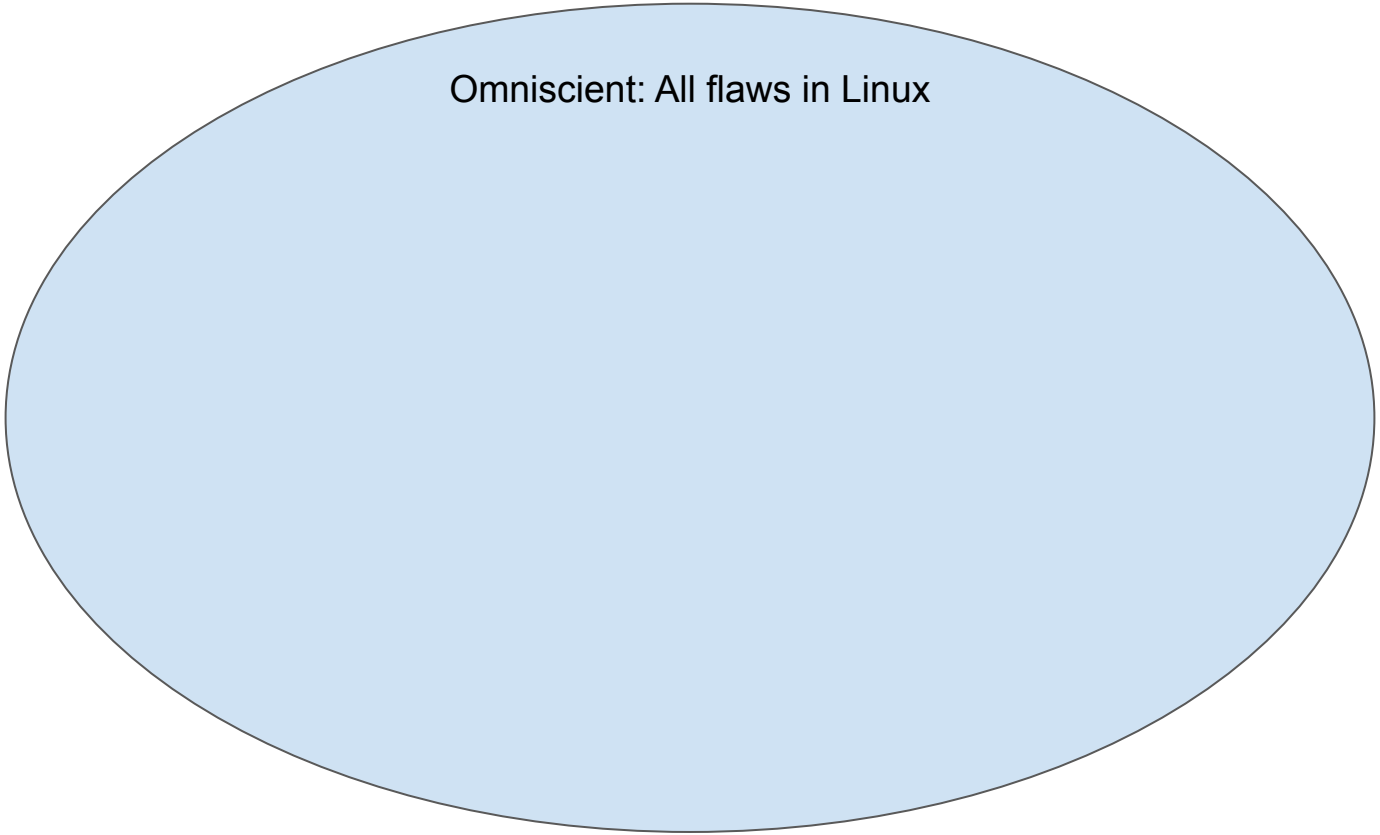
But first … Linux kernel flaws and CVEs

- Common Vulnerability Enumeration (maps vulnerabilities to CVE identifiers)

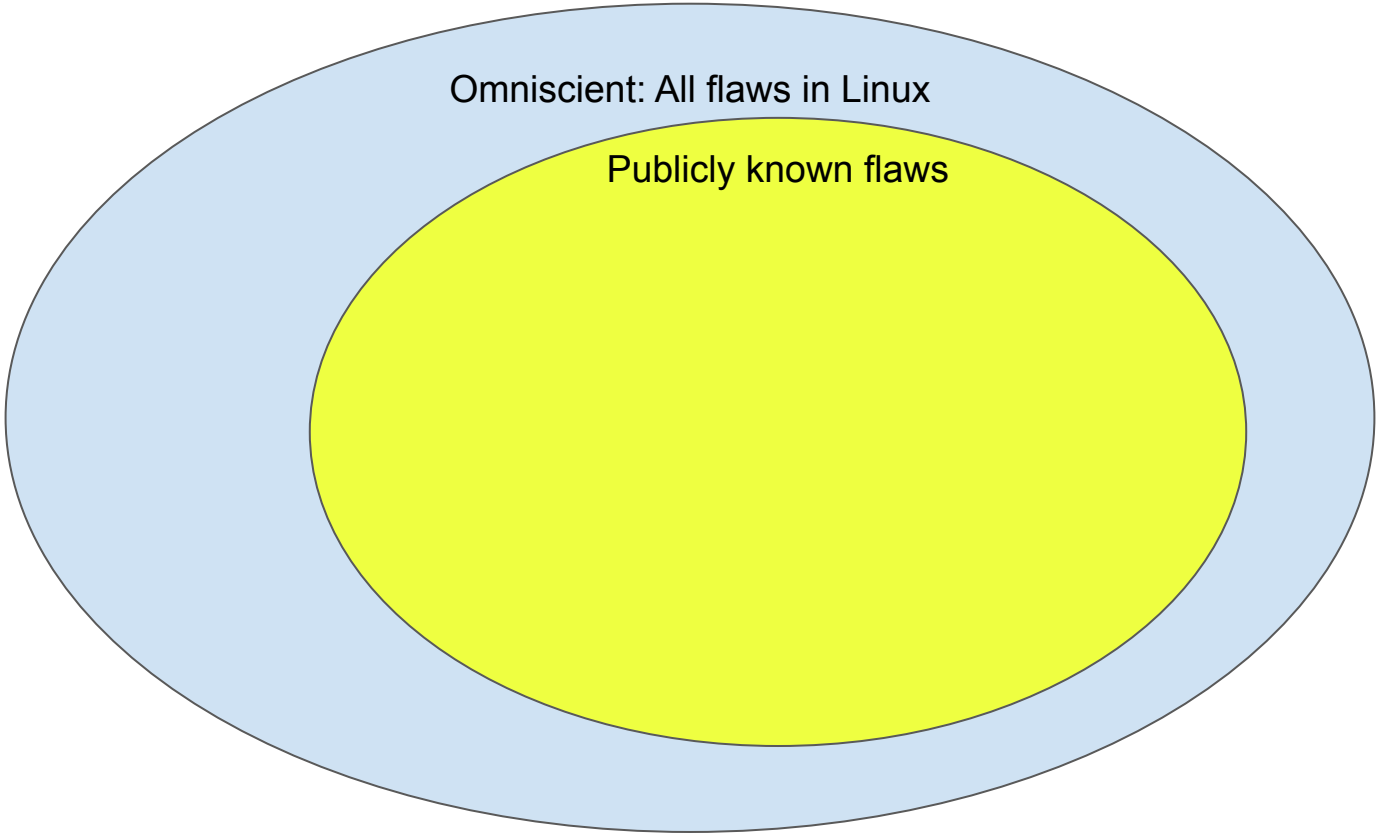
- Linux Kernel became its own CVE Naming Authority (CNA) in Feb 2024, which changed how CVEs got assigned.

- Prior to that, CVEs were most often assigned by general-purpose distros, and followed their threat models. (And dramatically under-counted flaws in the kernel.)

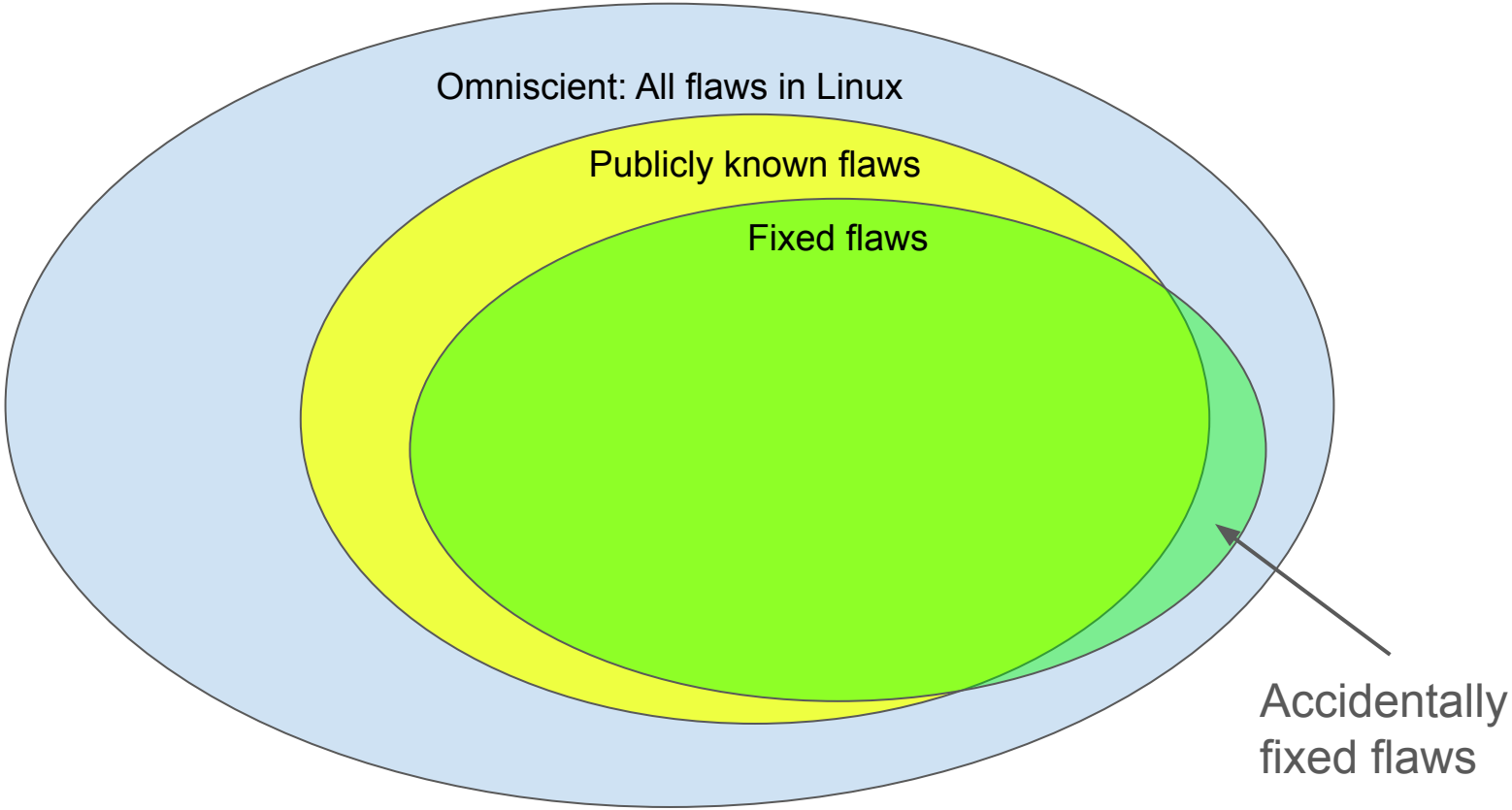
Linux Flaws Venn Diagram of Doom



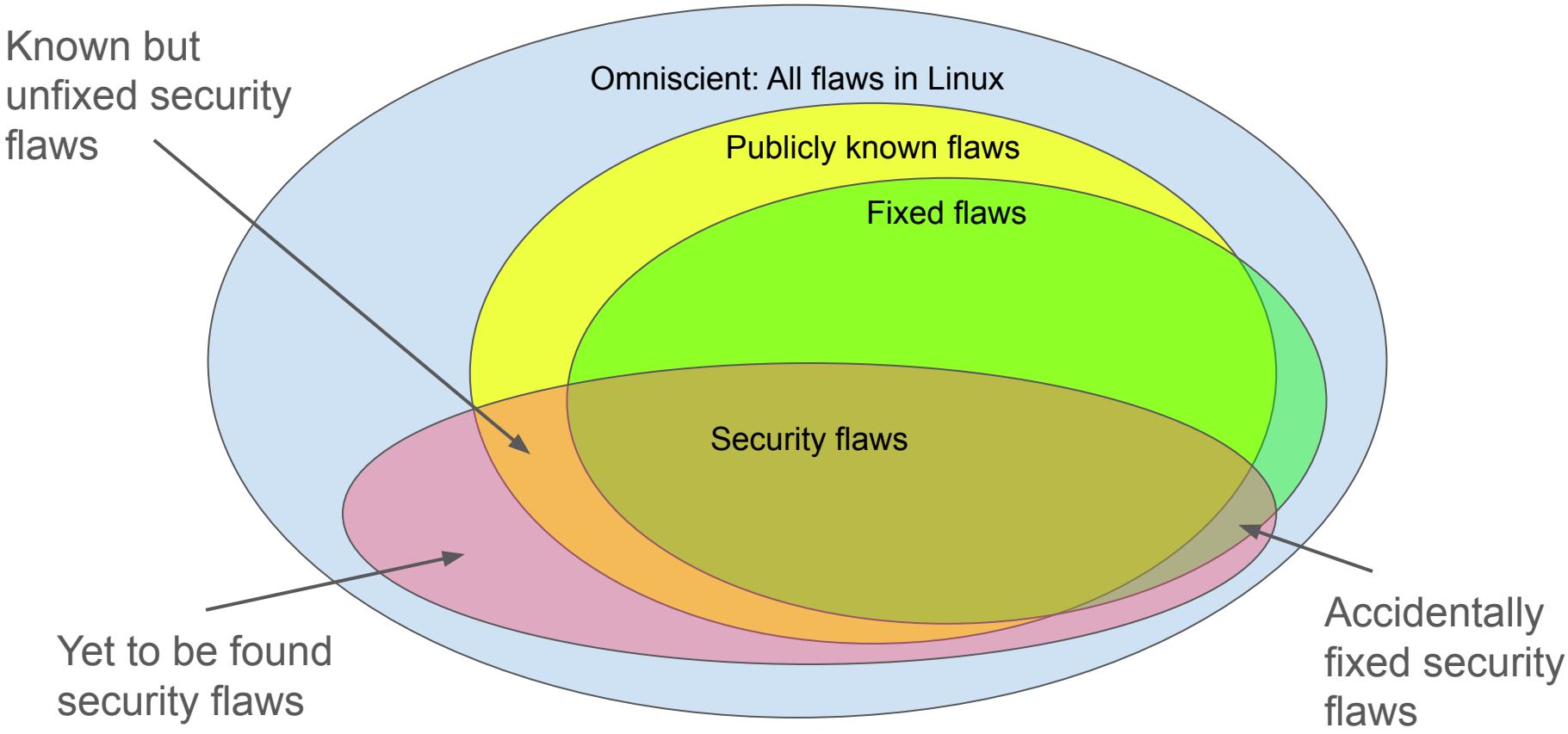
Linux Flaws Venn Diagram of Doom



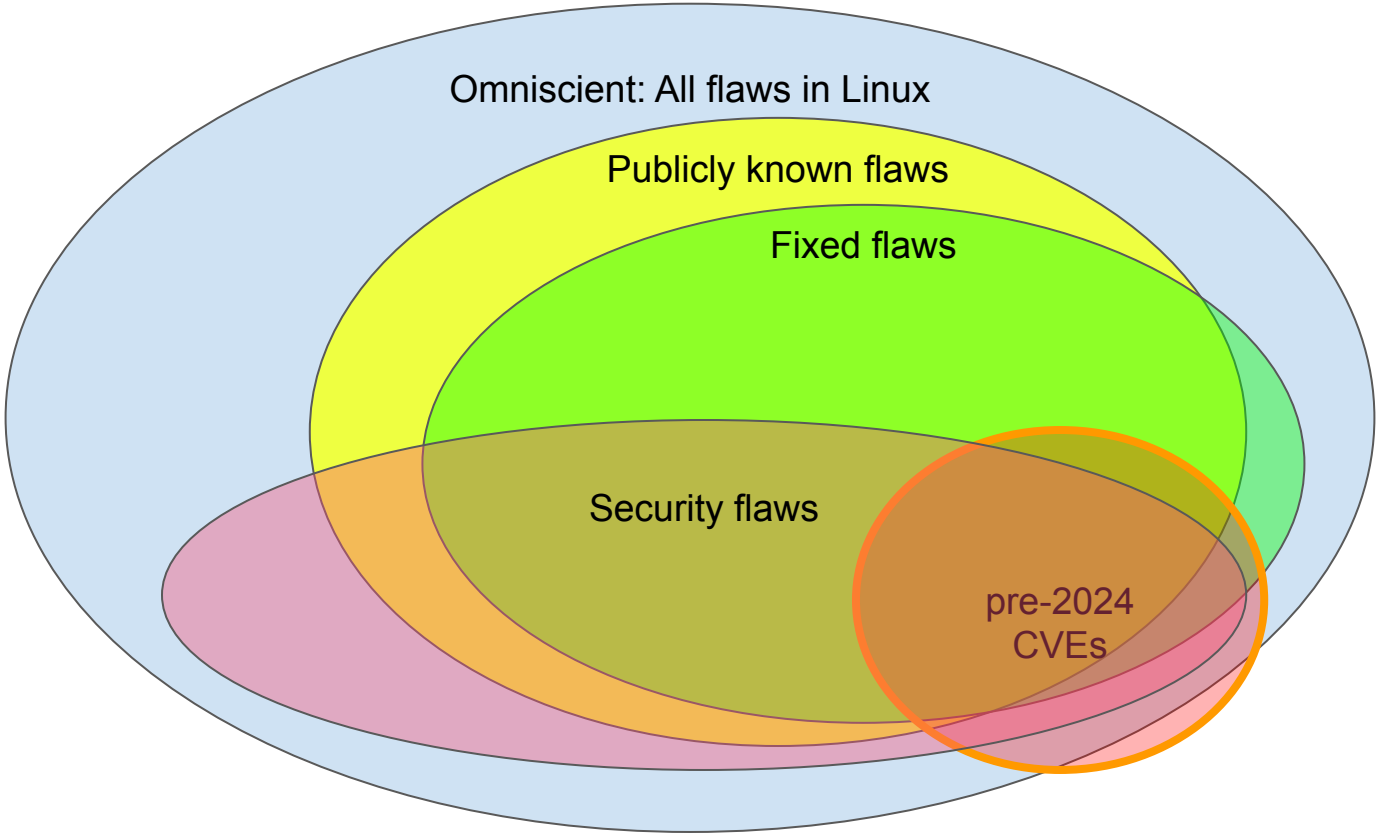
Linux Flaws Venn Diagram of Doom

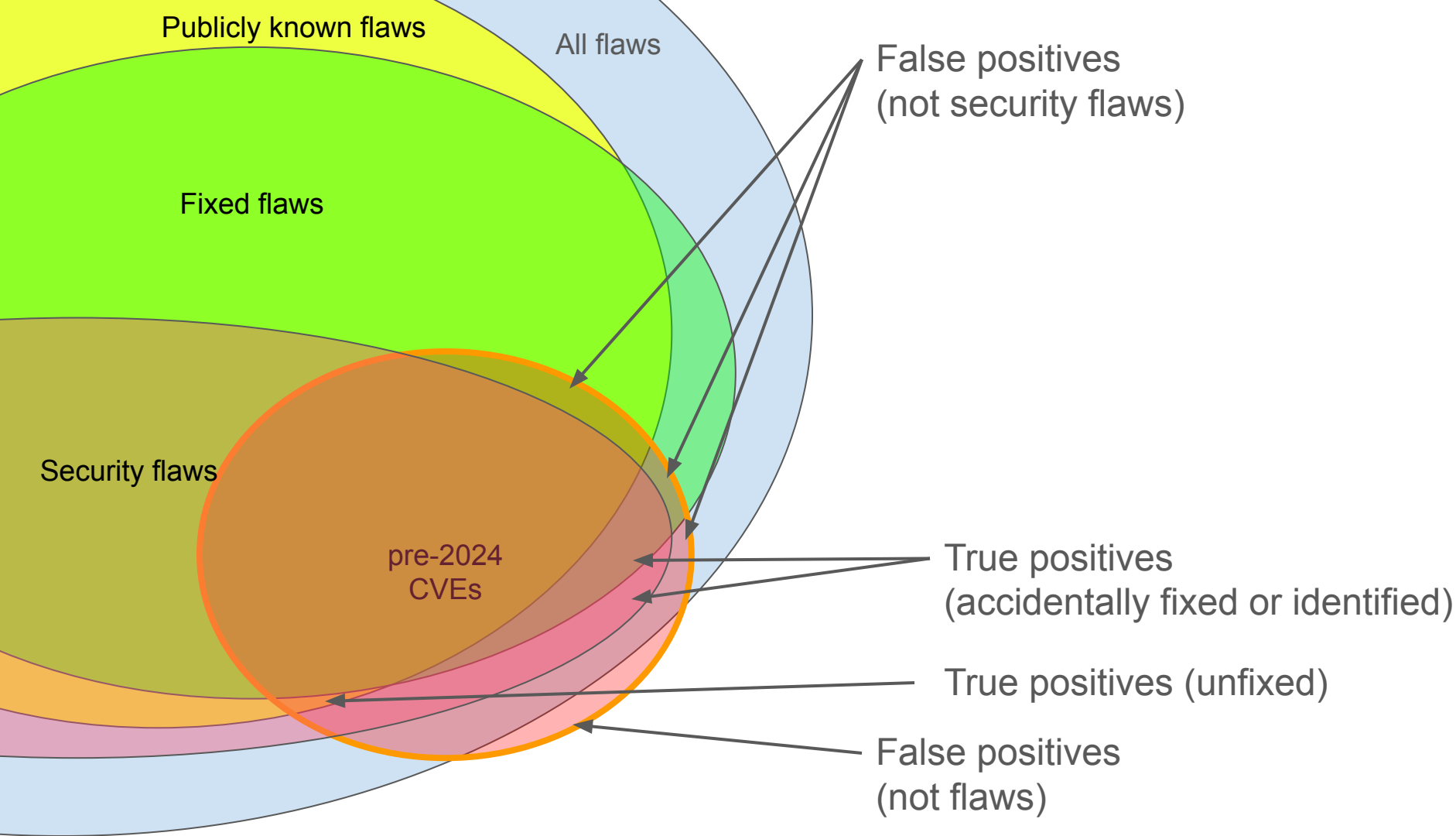


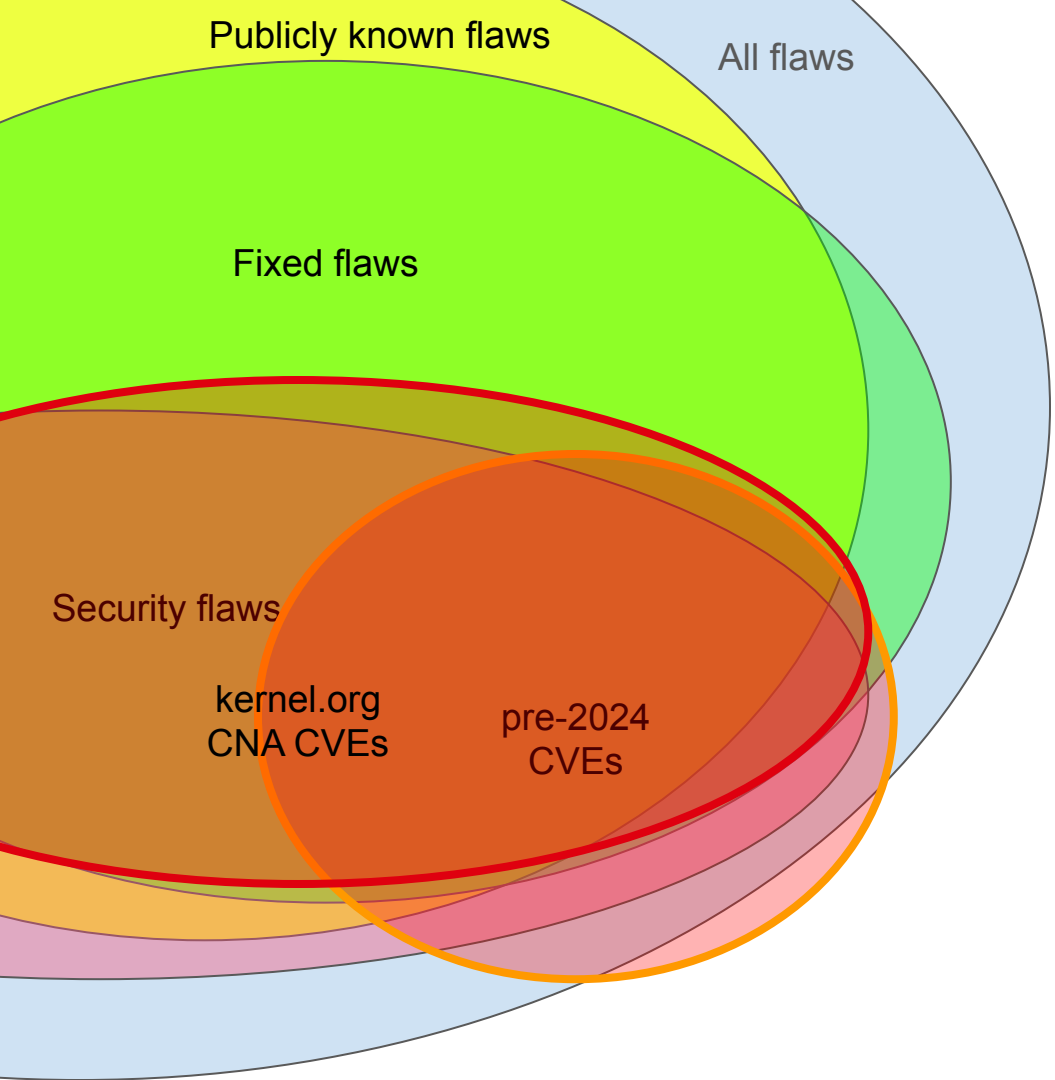
Linux Flaws Venn Diagram of Doom

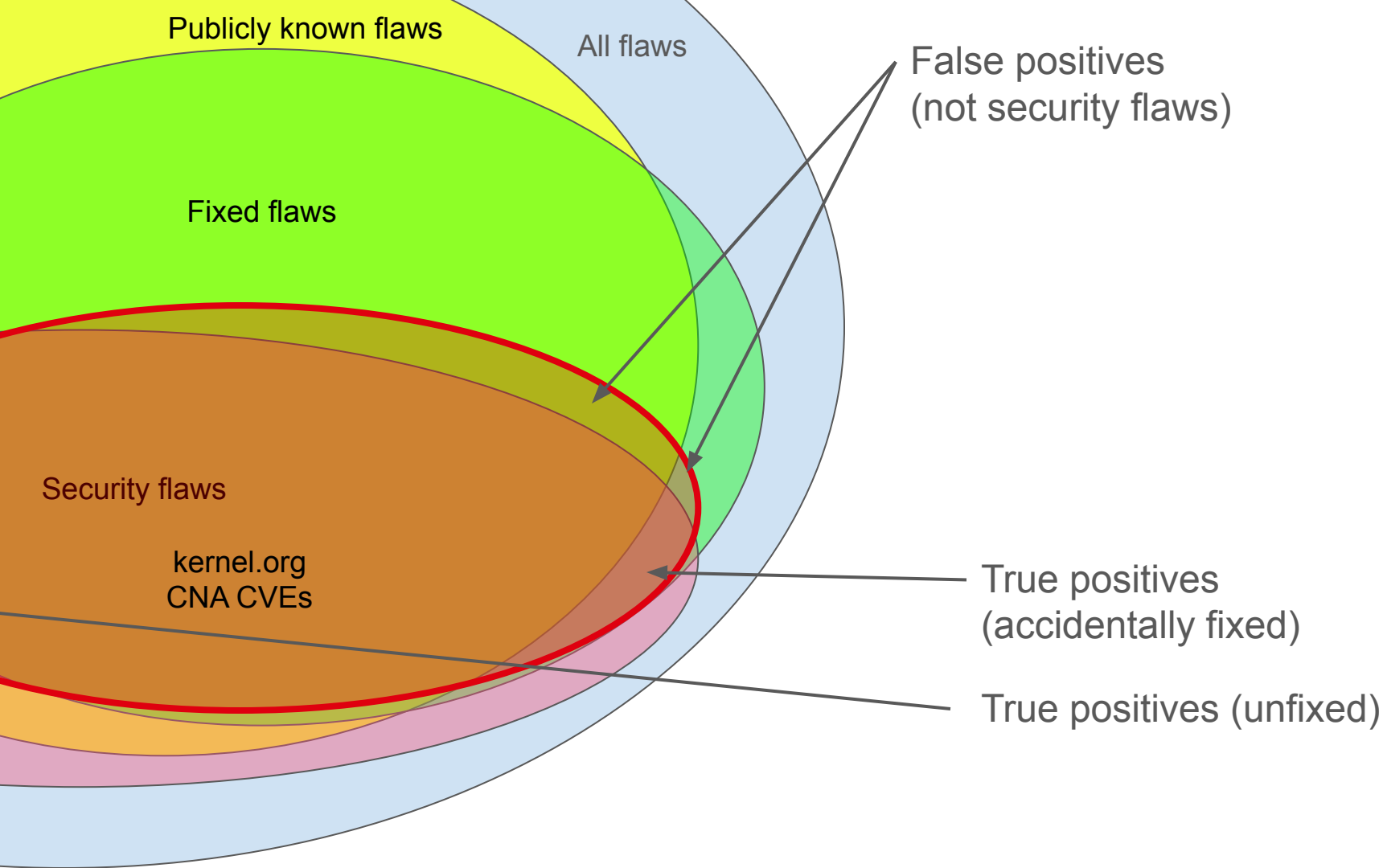


Linux Flaws Venn Diagram of Doom

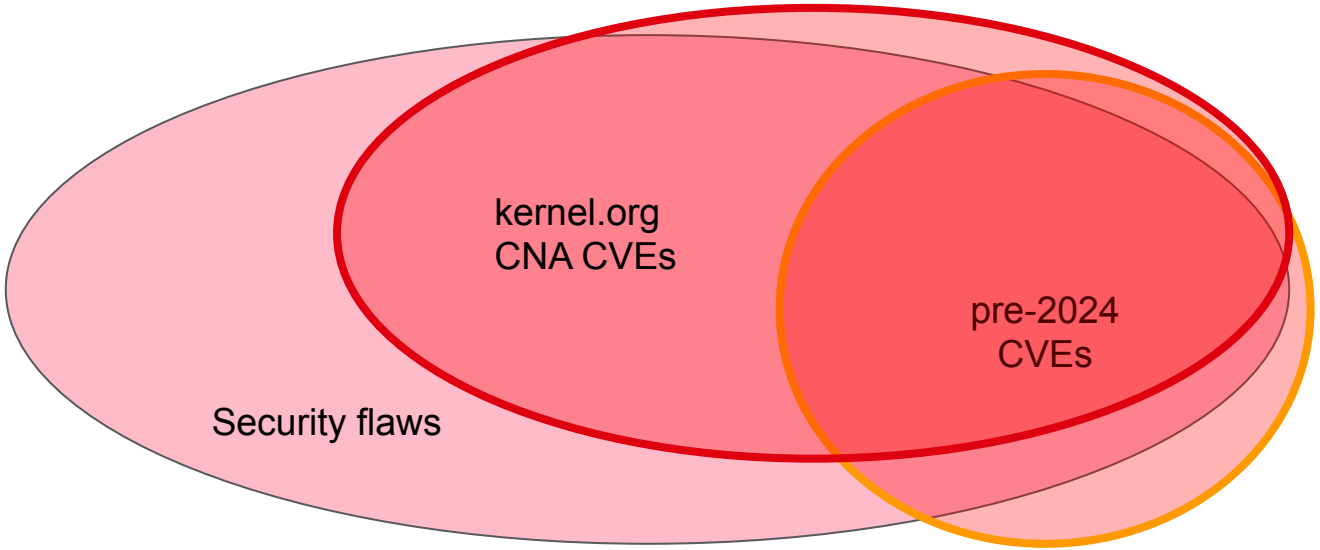



Publicly known flaws
All flaws
Fixed flaws
Security flaws
pre-2024
CVEs
False positives
(not security flaws)
True positives
(accidentally fixed or identified)
True positives (unfixed)
False positives
(not flaws)


Publicly known flaws
All flaws
Fixed flaws
Security flaws
kernel.org
CNA CVEs
pre-2024
CVEs


Publicly known flaws
All flaws
Fixed flaws
Security flaws
kernel.org
CNA CVEs
False positives
(not security flaws)
True positives
(accidentally fixed)
True positives (unfixed)

Reminder: the goal is to fix *security flaws*, not CVEs… (kernel.org CNA CVEs match reality much better)

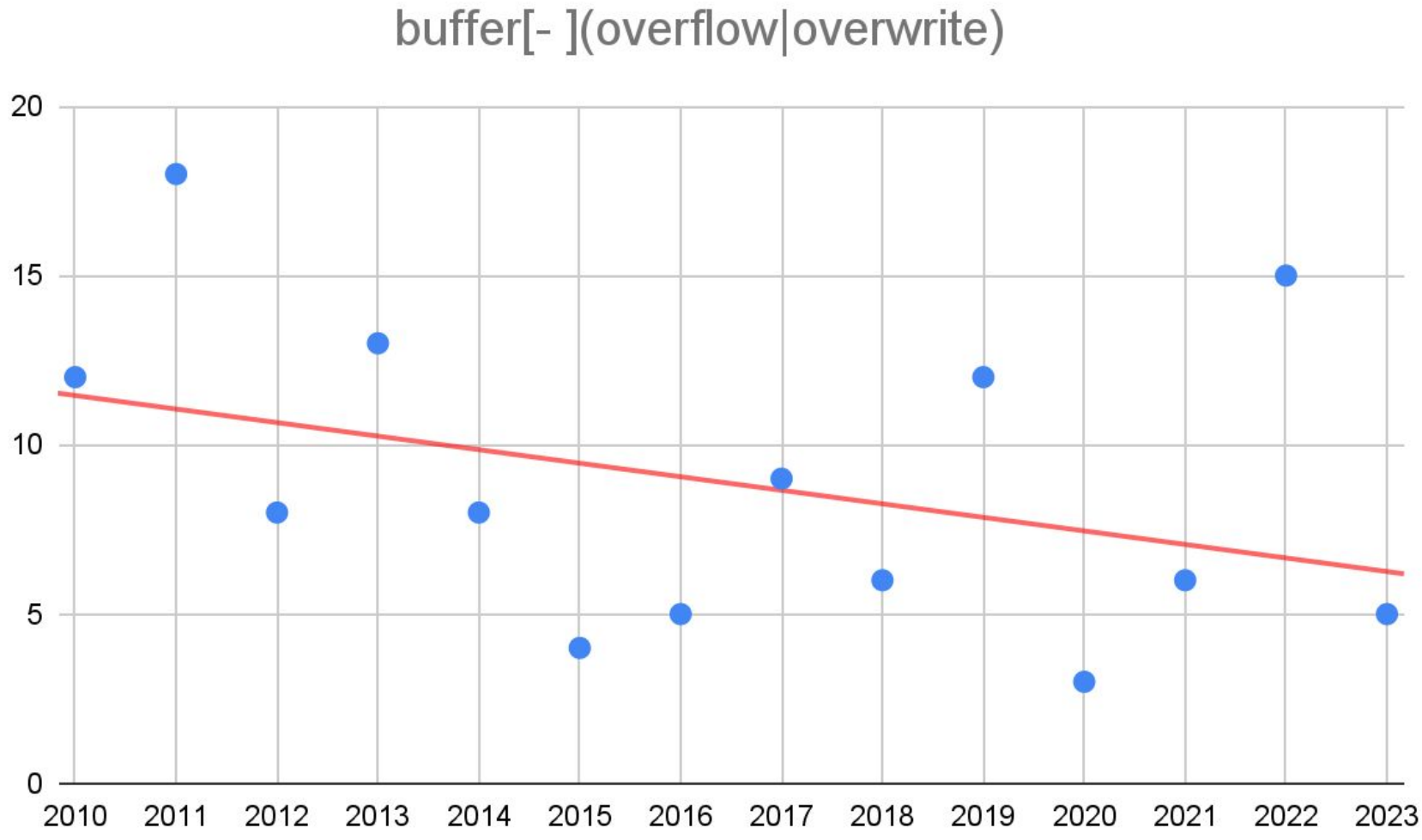


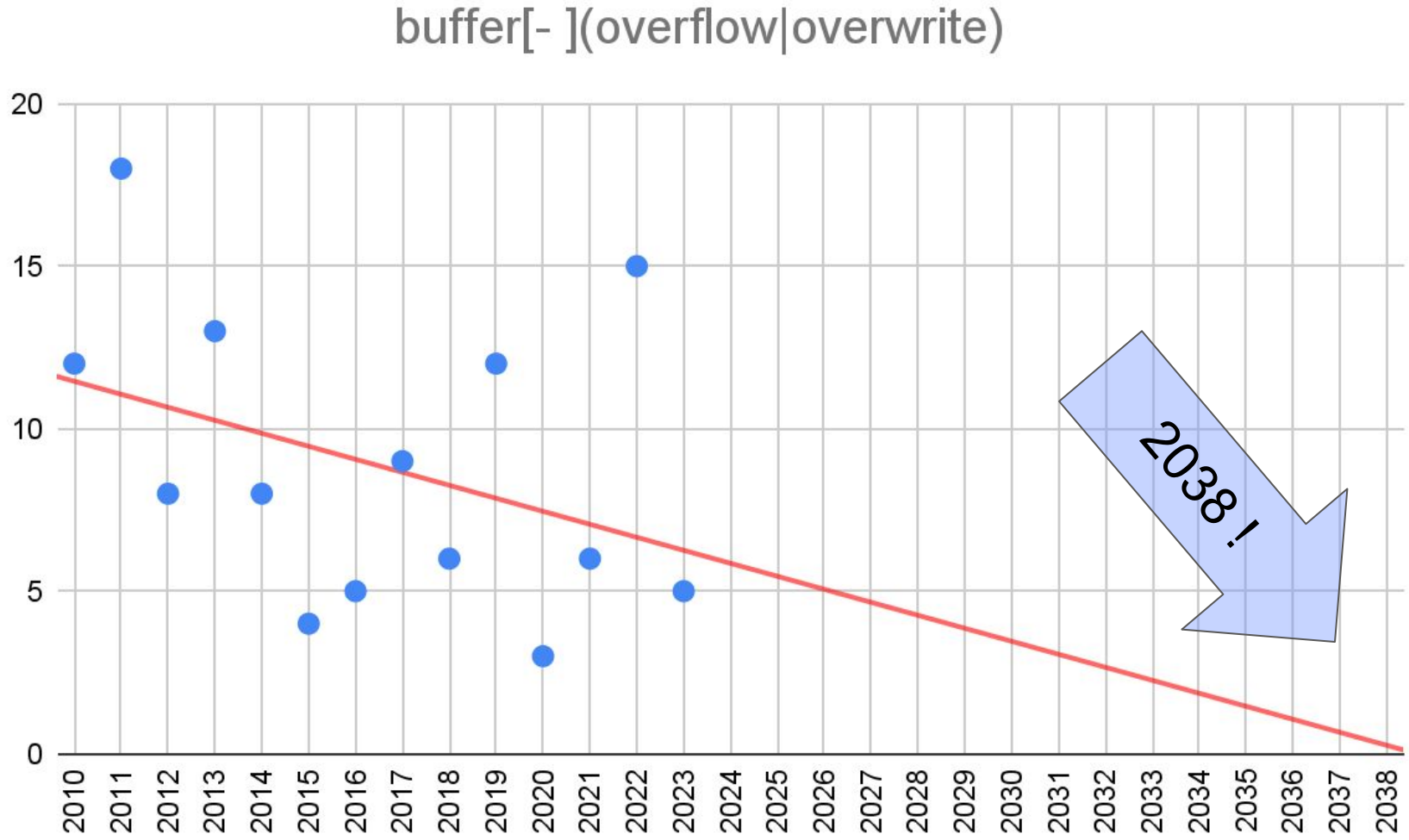
Lies, Damn Lies, and Statistics

- I use the Ubuntu CVE Tracker for my vulnerability statistics – they track the commits that introduced flaws as well as commits that fixed flaws, and they assign severity. This is everything I need to examine trends and lifetimes.

- Doing a retrospective examination of CVEs across the switch between CVE assignment methods isn't going to be easy. So I won't! To get a historical sense of vulnerability class trends, I only looked at pre-CNA CVEs.

- Now let's really look at some trends in bug classes!

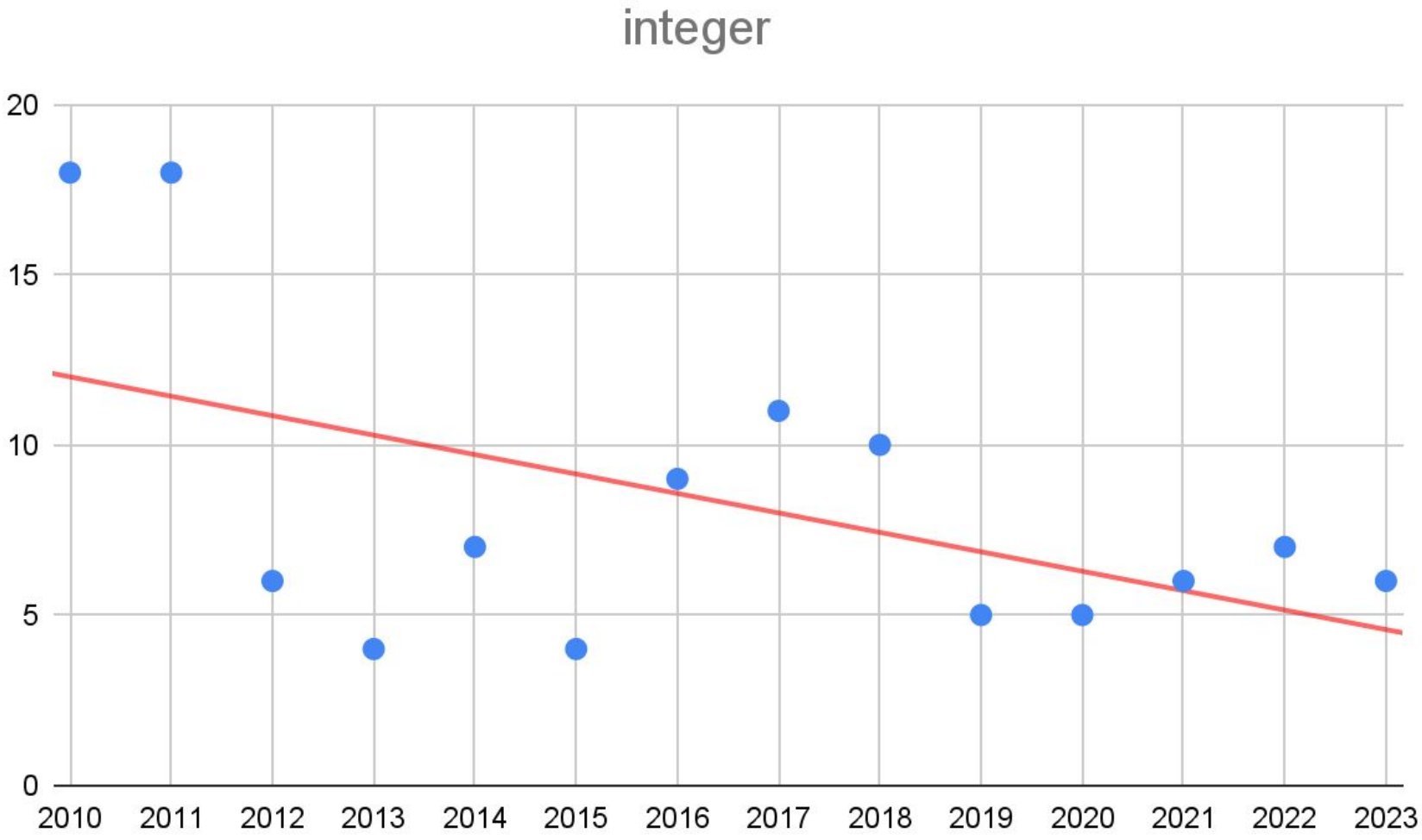

buffer[-](overflow|overwrite)
20
15
10
5
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023

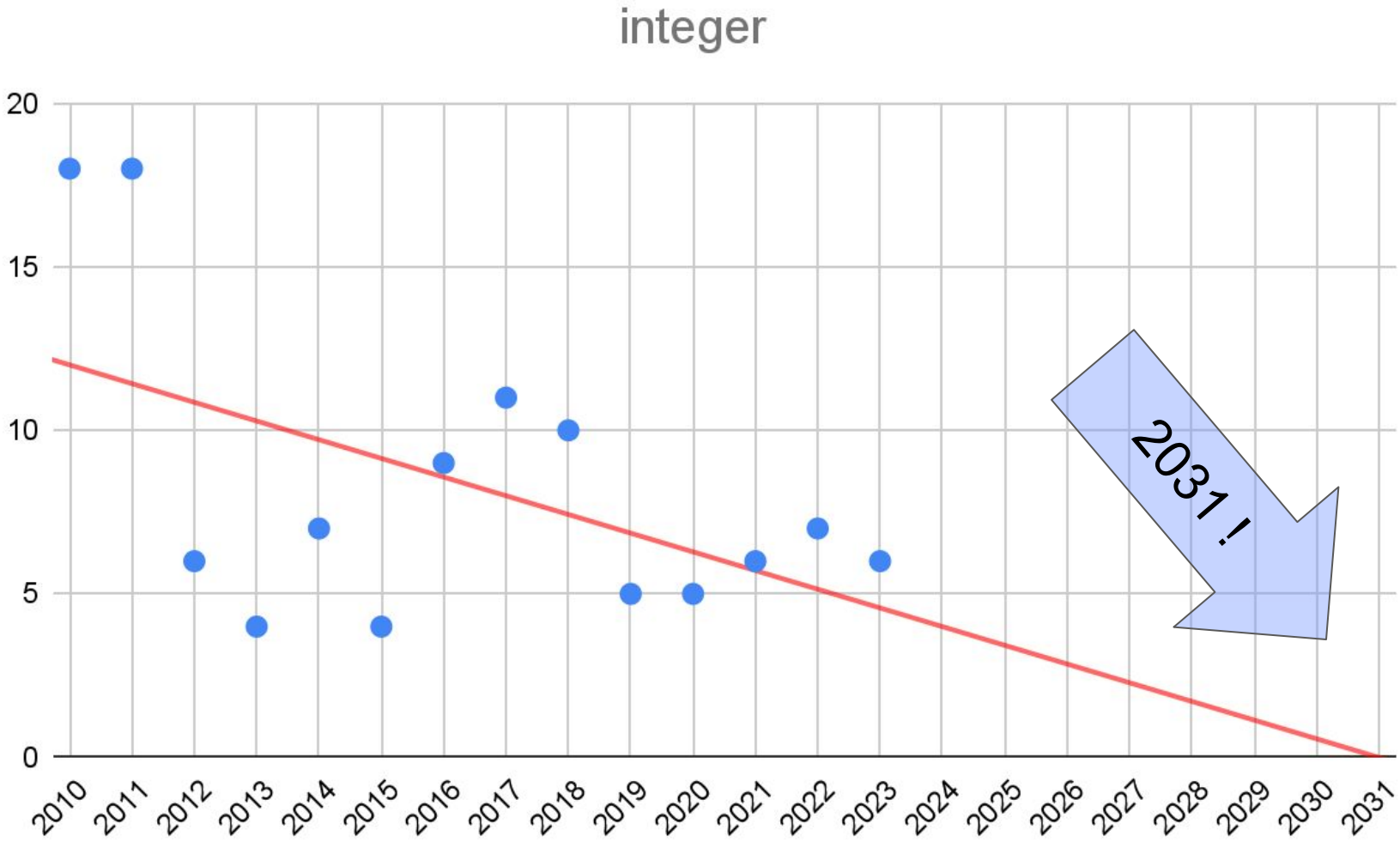

buffer[-](overflow|overwrite)
20
15
10
5
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038.!

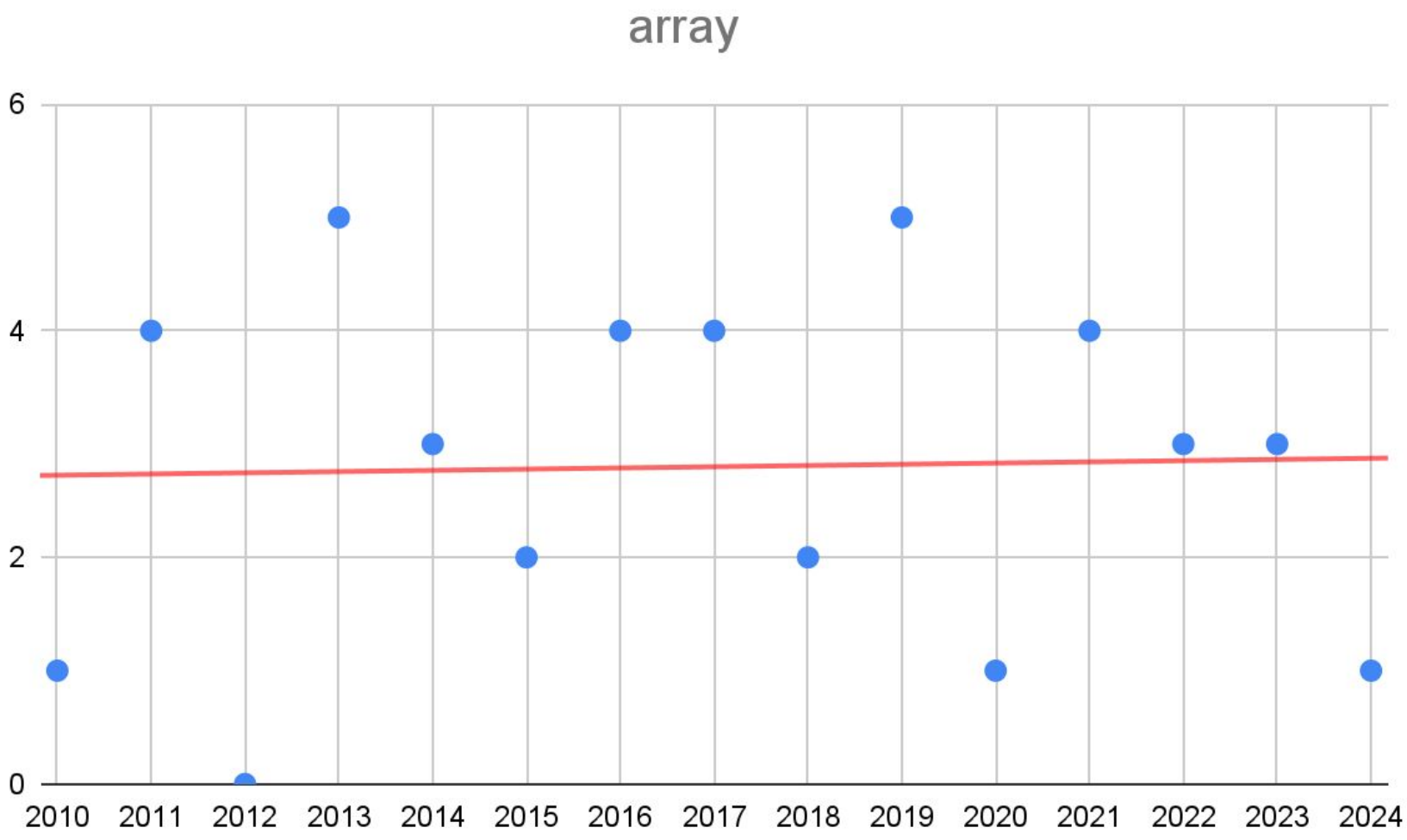
32-bit time_t Unix Epoch wrap!

```
11111111111111111111111111111111 03:14:07 19 Jan 2038 UTC
                                +1         *tick*
---------------------------------
00000000000000000000000000000000 00:00:00  1 Jan 1970 UTC
```

So … integer overflows …


integer
20
15
10
5
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023


integer
20
15
10
5
0
2031 !
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031

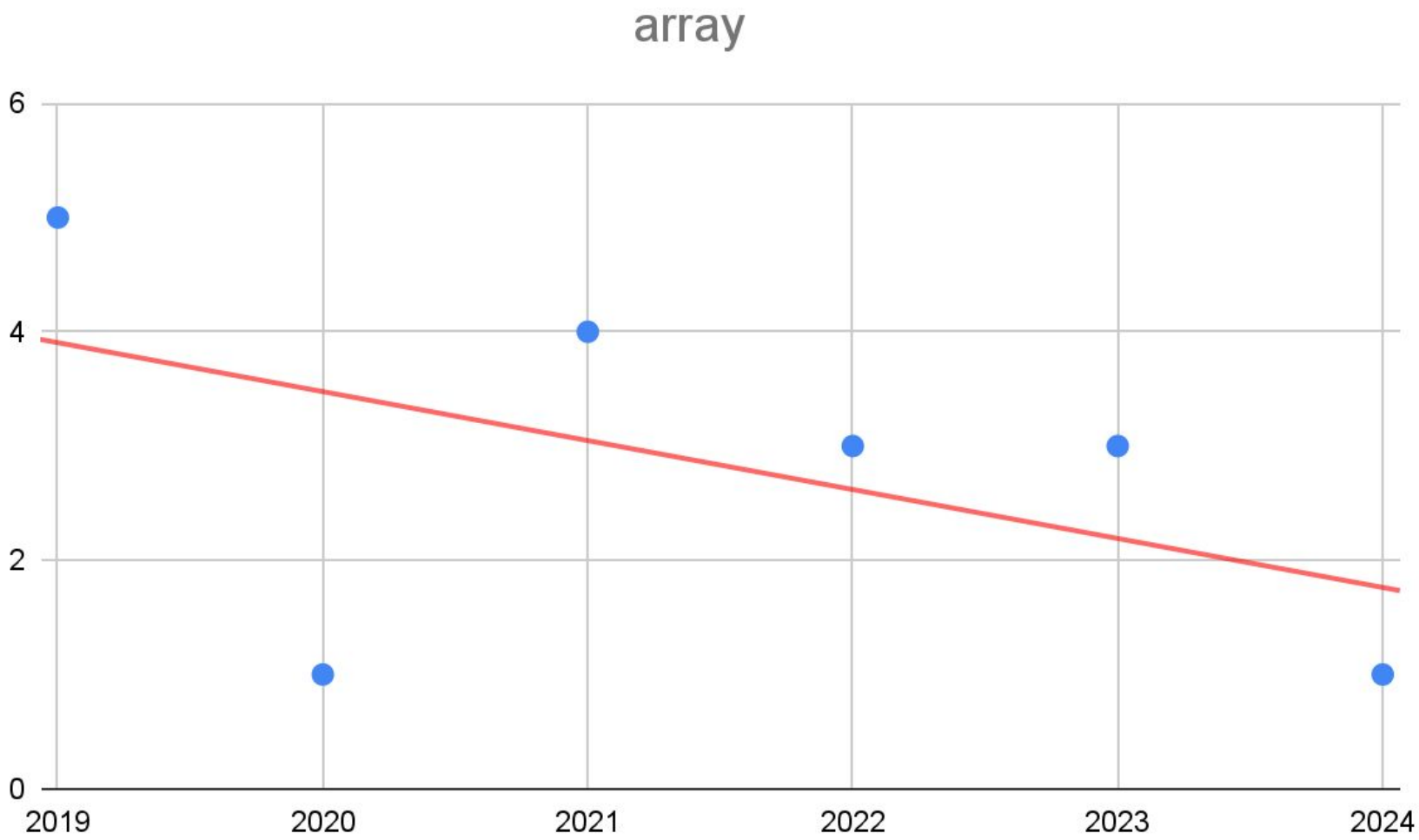

array
6
4
2
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024

2020: BleedingTooth

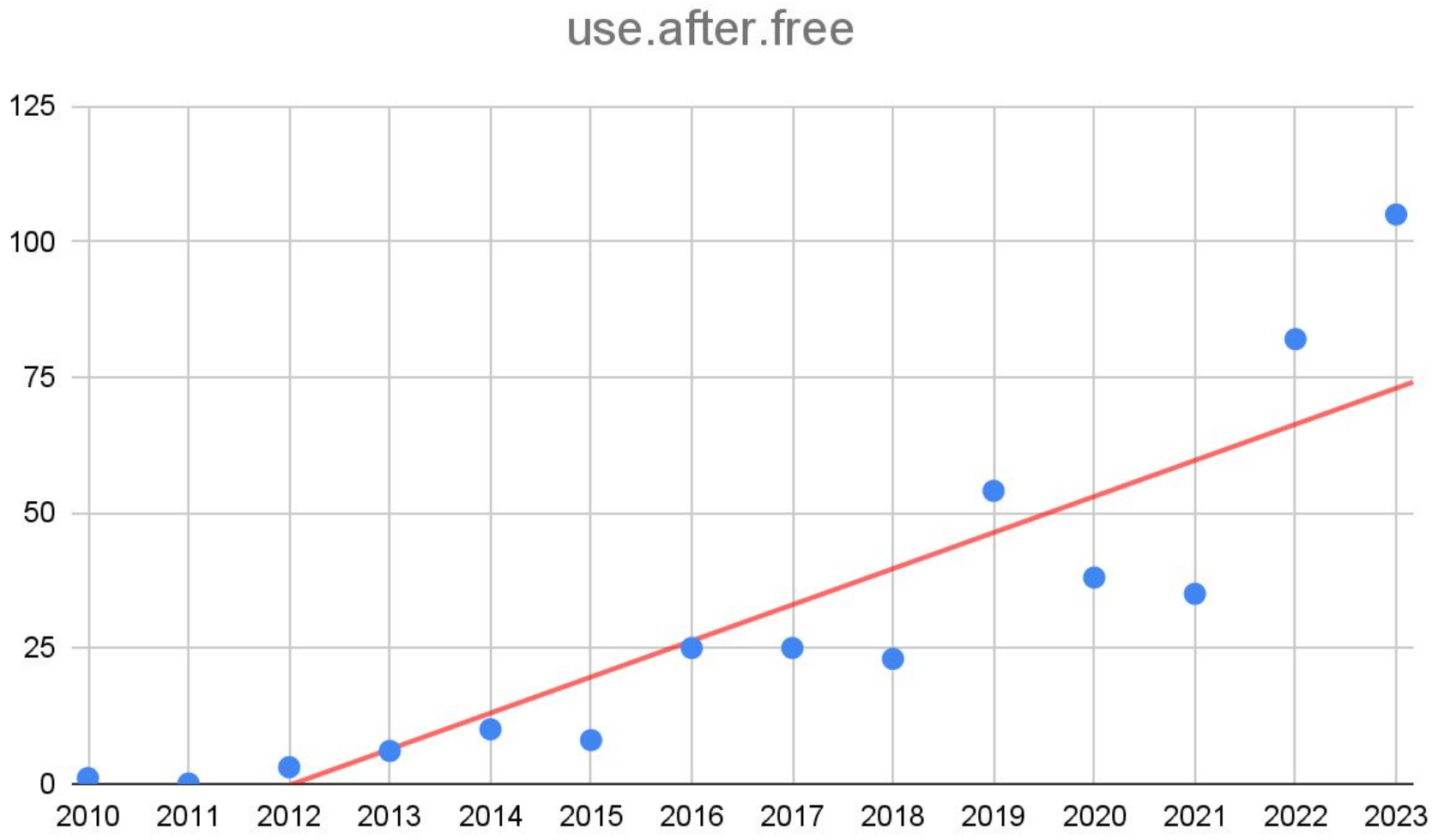
https://google.github.io/security-research/pocs/linux/bleedingtooth/writeup.html

```
struct hci_dev {
        ...
        struct discovery_state {
                ...
                u8 last_adv_data[HCI_MAX_AD_LENGTH];
                ...
        };
        ...
        struct list_head {
                struct list_head *next;
                struct list_head *prev;
        } mgmt_pending;
        ...
};

memcpy(d->last_adv_data, data, len);  /* len > HCI_MAX_AD_LENGTH ?! */
```


array
6
4
2
0
2019
2020
2021
2022
2023
2024

So where is the low hanging fruit now?


use.after.free
125
100
75
50
25
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023

Where are all the Use-After-Free flaws coming from?

```
30 net/netfilter
28 net/l2tp
17 drivers/android/binder.c
16 sound/core
15 fs/ext4
14 net/sched
14 fs/io_uring.c
11 net/bluetooth
10 net/ipv4
 9 kernel/futex.c
 8 net/ax25
 7 fs/btrfs
 6 net/nfc
 6 kernel/trace
 5 net/sctp
 5 net/packet
 5 net/ipv6
 5 fs/io-wq.h
 5 drivers/tty/vt
 5 drivers/net/hamradio
 5 drivers/gpu/drm
 4 net/unix
 4 net/socket.c
 4 fs/ntfs3
 4 fs/namei.c
 4 fs/eventpoll.c
 4 fs/cifs
 4 drivers/usb/misc
 4 drivers/media/dvb-core
 4 drivers/media/cec/core
 4 drivers/gpu/drm/vmwgfx
 4 drivers/block
 3 net/xfrm
   ...
```

Use-After-Free Research and Mitigation

- Google kernelCTF Vulnerability (and Patch) Reward Program
  https://google.github.io/security-research/kernelctf/rules
  - netfilter
    https://docs.google.com/spreadsheets/d/e/2PACX-1vS…wfvYC2oF/pubhtml
  - io_uring
    https://security.googleblog.com/2023/06/learnings-from-kctf-vrps-42-linux.html

- Android Binder being rewritten in Rust:
  https://rust-for-linux.com/android-binder-driver

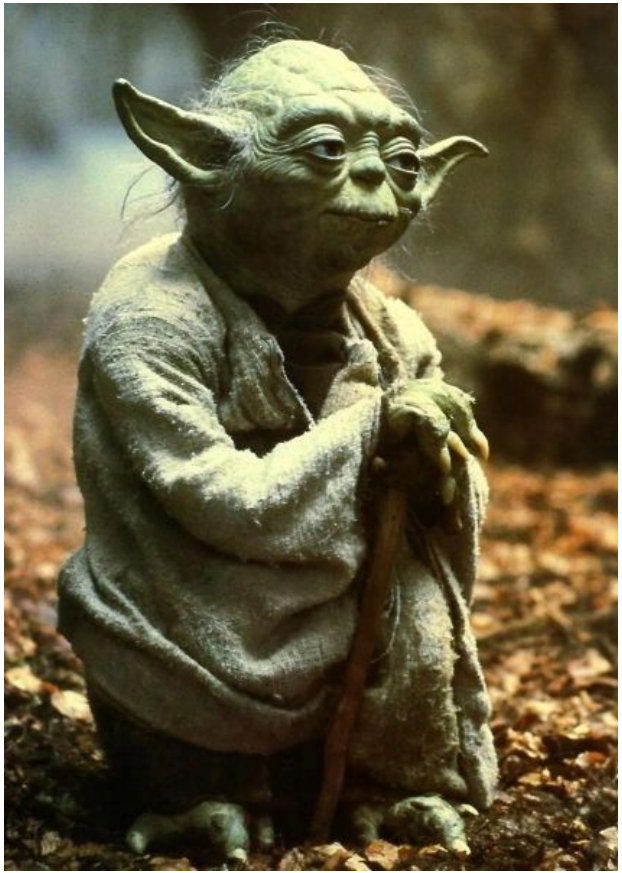
How did we drive down other bug classes?

- refactored to use trapping reference counters
- refactored to fault when accessing beyond the end of kernel stack
- removed Variable Length Arrays (VLAs) on the stack
- replaced open-coded allocation size arithmetic
- replaced set_fs() API to avoid user/kernel address space confusions
- improved compiler to reject implicit switch case fall-throughs
- improved compiler to zero-initialize stack variables
- improved compiler to actually check array sizes
- MOAR…

How did we drive down other bug classes?

- refactored to use trapping reference counters
- refactored to fault when accessing beyond the end of kernel stack
- removed Variable Length Arrays (VLAs) on the stack
- replaced open-coded allocation size arithmetic
- replaced set_fs() API to avoid user/kernel address space confusions
- *improved compiler* to reject implicit switch case fall-throughs
- *improved compiler* to zero-initialize stack variables
- *improved compiler* to actually check array sizes
- MOAR…

C supports ambiguity



"Ambiguity is the path to the Dark Side.

Ambiguity leads to confusion;

confusion leads to flaws;

flaws lead to suffering.

I sense much ambiguity in you."

– Yoda, about the C language

C supports ambiguity (but we can fix that)

- "Undefined Behavior" is the source of *so many* flaws, but is just one special case of "language ambiguity"
- and of course the lack of memory safety, no variable lifetime enforcement, no safe concurrency

What to do about it?

- Remove ambiguity in C
- Write new stuff in Rust

With Undefined Behavior

Anything is Possible

https://raphlinus.github.io/programming/rust/2018/08/17/undefined-behavior.html

Remove Ambiguity in C
"uninitialized" stack variables

There is no such thing as "uninitialized" !

```
int function(int input)
{
    int on_the_stack;  /* contains whatever was on stack */

    return input * on_the_stack; /* returns what??? */
}
```

Remove Ambiguity in C
"uninitialized" stack variables

Now we can build with `-ftrivial-auto-var-init=zero` ...

```
int function(int input)
{
    int on_the_stack;  /* contains 0 */

    return input * on_the_stack; /* returns 0 */
}
```

Some compiler folks worried "this will fork the language" … YES PLEASE

Remove Ambiguity in C
not all arrays can be bounds checked

```
struct foo {
    …
    …
    int fixed_size_array[16];
    int flexible_array[];
};
```

Can do bounds checking! (16 elements)

No dimension: no bounds checking :(

Remove Ambiguity in C
not all arrays can be bounds checked

Now we can use the `counted_by` attribute …

```
struct foo {
    …
    int items;
    int fixed_size_array[16];
    int flexible_array[] __attribute__((counted_by(items));
};
```

Can do bounds checking! (16 elements)

Can do bounds checking! ("items"-many elements)

Remove Ambiguity in C

The C Standard is strict, slow-moving, and prioritizes compatibility over robustness. The key to making any practical progress with GCC, Clang, and even MSVC is to use the magic phrase:

Remove Ambiguity in C*ompilers*

The C Standard is strict, slow-moving, and prioritizes compatibility over robustness. The key to making any practical progress with GCC, Clang, and even MSVC is to use the magic phrase:

I would like to add this **Language Extension** …

Then coordinate the extension between compilers, and the C Standard can catch up when they're ready.

Write New Stuff in Rust

It's a long road to in the Linux kernel, but the language bindings have been steadily landing. Entire graphics drivers have been written in Rust: Apple AGX, Nova. Also filesystems, block drivers, network PHY drivers… If the Linux kernel can get it done, so can your project!

You know it's time to ditch C/C++ when even governments have noticed the dumpster fire. National Security Agency (NSA), Cybersecurity and Infrastructure Security Agency (CISA), and Office of the National Cyber Director (ONCD):

The Case for Memory Safe Roadmap

Exploring Memory Safety in Critical Open Source Projects

https://rust-for-linux.com/

Those have been my shared struggles. How're you doing?

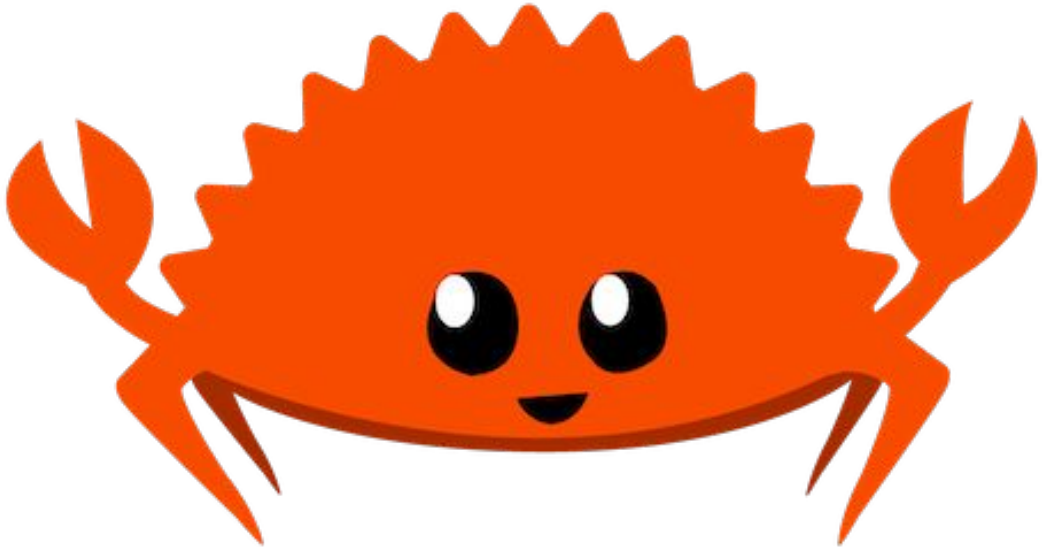
Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!



Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!

Defending the Cloud from evil? The job never ends!

Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!

Defending the Cloud from evil? The job never ends!

Keeping AI from consuming the planet? I don't want to be turned into paperclips!

Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!

Defending the Cloud from evil? The job never ends!

Keeping AI from consuming the planet? I don't want

Jail-breaking devices so I can fully use the hardware I own? Thank you!

Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!

Defending the Cloud from evil? The job never

Keeping AI from consuming the planet? I don't

Jail-breaking devices so I can fully use the hard

Doing other stuff in this industry? I love it!

Those have been my shared struggles. How're you doing?

Just getting started? Keep it up!

Already writing stuff in Rust? You're awesome!

Defending the Cloud from evil? The job never ends!

Keeping AI from consuming the planet? I don't want to be turned into paperclips!

Jail-breaking devices so I can fully use the hardware I own? Thank you!

Doing other stuff in this industry? I love it!

All of our work can be a struggle, but it makes a difference

I don't care if this is cheesy, it's still true…

Fred Rogers again:

"… what you're planning and doing are things that can be a real help to you and your neighbor.

I'm proud of you."

Thank you!

Enjoy the rest of the day :)

Kees ("Case") Cook
https://fosstodon.org/@kees
kees@kernel.org

https://outflux.net/slides/2024/bsidespdx/decade.pdf