Upstream Kernel Hardening: Recent Progress and Challenges

Gustavo A. R. Silva
gustavoars@kernel.org
https://embeddedor.com/blog/

Supported by
The Linux Foundation & Alpha-Omega

Open Source Summit Korea
August 11, 2026
Seoul, South Korea

Who am I?

By @shidokou

Who am I?

- **Upstream first** – 10th year.
- Upstream Linux Kernel Engineer.
    - Kernel hardening.
    - Proactive security.

By @shidokou

Who am I?

- **Upstream first** – 10th year.
- Upstream Linux Kernel Engineer.
    - Kernel hardening.
    - Proactive security.
- Kernel Self-Protection Project (**KSPP**).
- Google Open Source Security Team (**GOSST**).
    - Linux Kernel division.

By @shidokou

Linux Kernel Self-Protection Project (KSPP)

Our goal is to remove entire classes of bugs and methods of exploitation.

REMOVE BUG CLASSES

Express bounds, sizes, and remove ambiguity so mistakes become diagnosable.

BLOCK/MITIGATE EXPLOITATION

Reduce primitives and make common exploitation methods less reliable or impossible to occur.

DETECT ATTACKS

Turn violated invariants into visible failures instead of silent corruption.

The recent direction: turn intent into machine-checkable invariants

SOURCE INTENT	COMPILER MEANING	ENFORCEMENT
FAM transformations, counted_by annotations, struct_group(), TRAILING_OVERLAP(), memcpy() semantics, Write unambiguous code	object size bounds, counted_by info, -fstrict-flex-arrays=3	-Wvla, -Warray-bounds, UBSAN_BOUNDS, FORTIFY_SOURCE, KCFI

In one year, several long-running hardening threads crossed milestones

v6.17

Jul 2025

TRAILING_OVERLAP(), STACKLEAK renamed to KSTACK_ERASE (Clang native support)

v6.18–6.19

Fall 2025

KCFI naming/toolchain prep better GCC diagnostics for -Warray-bounds

v7.0–7.1

Feb–Apr 2026

__counted_by_ptr, randomize_kstack rework, kmalloc_obj() family

v7.2 cycle

Jun 2026

last strncpy() user gone, strncpy() removed from the kernel

Flexible-array revolution is approaching the point of enforcement

`-Wflex-array-member-not-at-end`

~60,000

warnings in the early campaign – late 2023

~650

unique issue initially

100

issues left in linux-next this year - 2026

One January patch eliminated 2,600 warnings at once—but every remaining case requires careful auditing.

The warning targets a memory layout the compiler cannot reason about safely

AMBIGUOUS LAYOUT

```
struct flex {
    size_t count;
    u8 data[];          /* FAM */
};

struct composite {
    ...
    struct flex x; /* not-at-end FAM */
    u32 trailer;
};
```

UNAMBIGUOUS INTENT

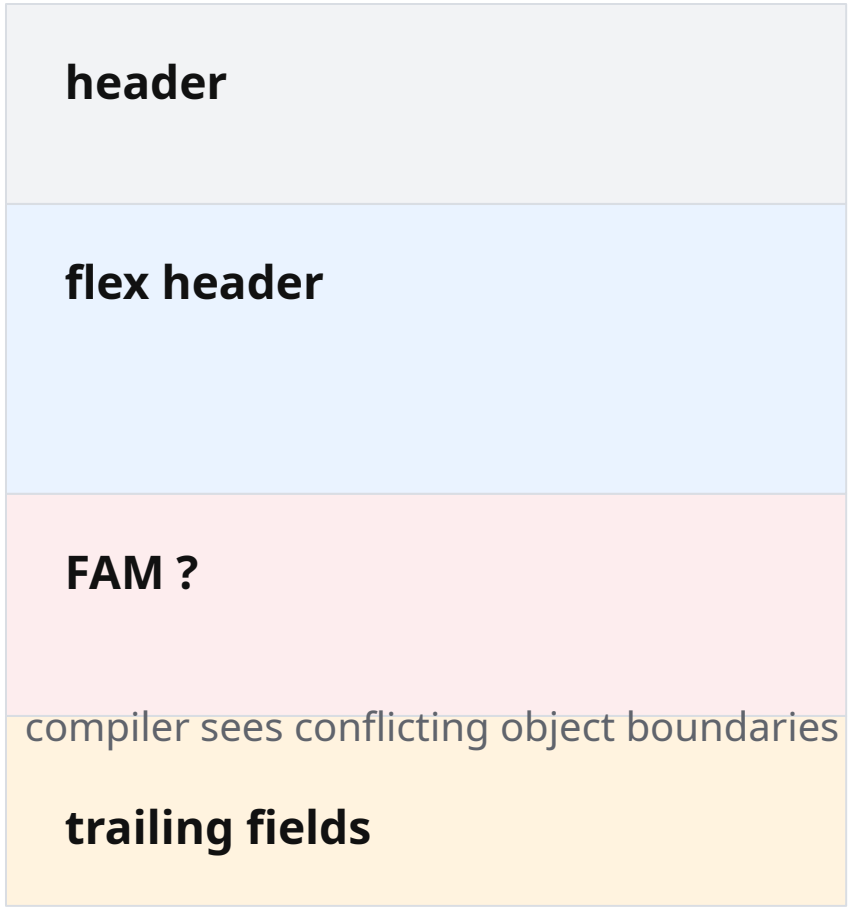
```
struct composite {
    ...
    /* FAM ends the object */
    struct flex x;
};

/* or express a real overlap
 * explicitly */
```

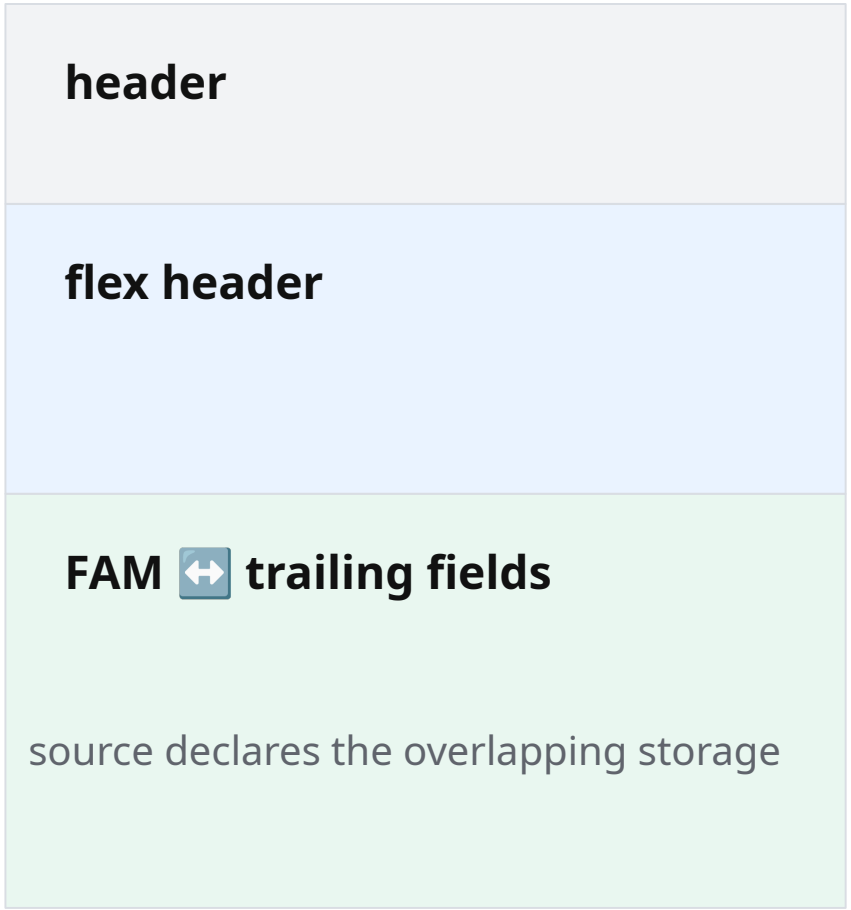
WHY IT MATTERS: object boundaries drive bounds checking

TRAILING_OVERLAP() makes an implicit union explicit

Before



After



-Wflex-array-member-not-at-end warnings find real bugs

virtio_net

A FAM-not-at-end warning exposed a misalignment bug in struct virtnet_info.

The fix used TRAILING_OVERLAP() to preserve the intended overlap while restoring correct alignment.

[PATCH] virtio_net: Fix misalignment bug in struct virtnet_info

LESSON

A warning cleanup can uncover a correctness defect that may also have security consequences.

CVE-2026-23143

counted_by aids the compiler to determine the size of a FAM at runtime

FLEXIBLE-ARRAY MEMBER BOUNDS

```
struct foo {
    size_t count;
    int fam[] __counted_by(count);
};
```

IN-STRUCT POINTER BOUNDS

```
struct bar {
    char *name __counted_by_ptr(length);
    size_t length;
};
```

~500 counted_by FAMs

In linux-next today

v7.0: __counted_by_ptr enters the kernel

Type-based slab allocations: kmalloc_obj() family

Old:

```
p = kmalloc(sizeof(*p), gfp);

p = kmalloc(sizeof(struct foo), gfp);

p = kzalloc(sizeof(*p), gfp);

p = kmalloc_array(count, sizeof(*p), gfp);

p = kcalloc(count, sizeof(*p), gfp);
```

No type info passed.
Returns void *

New:

```
p = kmalloc_obj(*p, gfp);

p = kmalloc_obj(struct foo, gfp);

p = kzalloc_obj(*p, gfp);

p = kmalloc_objs(*p, count, gfp);

p = kzalloc_objs(*p, count, gfp);
```

GFP flags are optional. GFP_KERNEL is used by default.
Returns TYPE *

This is the first phase. No allocation by type yet.

Type-based slab allocations: kmalloc_flex() family

```
struct flex {
    size_t count;
    struct foo fam[] __counted_by(count);
} *p;
```

Old:

```
p = kmalloc(struct_size(p, fam, ITEMS), gfp);
...
p->count = ITEMS;
```

New:

```
p = kmalloc_flex(p, fam, ITEMS, gfp);
```

Type-based slab allocations: kmalloc_flex() family

```
struct flex {
    size_t count;
    struct foo fam[] __counted_by(count);
} *p;
```

Old:

```
p = kmalloc(struct_size(p, fam, ITEMS), gfp);
...
p->count = ITEMS;
```

New:

```
p = kmalloc_flex(p, fam, ITEMS, gfp);
```

This will also initialize the counter member (if the FAM is annotated).

Required compiler support: __builtin_counted_by_ref

strncpy() is gone

6 years

sustained cleanup

362

commits

70

contributors

The 7.2 development cycle could finally remove the implementation itself.

strncpy() is gone

6 years

sustained cleanup

362

commits

70

contributors

One (very) ambiguous API → explicit operations

- strscpy()
- strscpy_pad()
- memtostr()
- memtostr_pad()
- strtomem()
- strtomem_pad()
- memcpy_and_pad()

The 7.2 development cycle could finally remove the implementation itself.

strncpy() is gone

6 years

sustained cleanup

362

commits

70

contributors

One (very) ambiguous API → explicit operations

- strscpy()
- strscpy_pad()
- memtostr()
- memtostr_pad()
- strtomem()
- strtomem_pad()
- memcpy_and_pad()
- memcpy()

The 7.2 development cycle could finally remove the implementation itself.

FORTIFY is only as good as the object size the compiler can prove

1 SOURCE

```
memcpy(dst, src, len)
```

2 COMPILER

```
__builtin_dynamic_object_size(dst, 0)
__builtin_dynamic_object_size(dst, 1)
```

3 FORTIFY

```
compile-time diagnostic
and runtime bounds
checking
```

FORTIFY is only as good as the object size the compiler can prove

1 SOURCE

```
memcpy(dst, src, len)
```

2 COMPILER

```
__builtin_dynamic_object_size(dst, 0)
__builtin_dynamic_object_size(dst, 1)
```

3 FORTIFY

```
compile-time diagnostic
and runtime bounds
checking
```

Hard part today: nested flexible structures, subobjects, and dynamic counts can still lose size information.

Clang recently fixed a couple of bugs in __bdos(), and fixes for two more __bdos() bugs in GCC are currently in progress.

The compiler is now part of the hardening architecture

OBJECT & BOUNDS SEMANTICS

`-fstrict-flex-arrays=3`

`__counted_by() / __counted_by_ptr()`

`__builtin_dynamic_object_size()`

`__nonstring`

ENFORCEMENT & DIAGNOSTICS

`-Wflex-array-member-not-at-end`

`-Warray-bounds / string diagnostics`

`value-tracking diagnostic context`

`CONFIG_FORTIFY_SOURCE=y`

`CONFIG_UBSAN_BOUNDS=y`

`-fsanitize=kcfi`

v6.18 renamed CONFIG_CFI_CLANG → CONFIG_CFI to stop encoding one compiler into the feature name.

The last year also moved stack hardening across architectures

KSTACK_ERASE

STACKLEAK refactored and renamed; native Clang stack-depth tracking support.

RANDOMIZE_KSTACK

v7.1 improved implementation across architectures (ARM64)

HARDENED CONFIG

The hardening config enables KSTACK_ERASE and init-on-free defaults.

What we actually build in KSPP falls into four reinforcing layers

SOURCE TRANSFORMATION

FAM transformations,
counted_by annotations,
remove unsafe APIs,
eradicate ambiguity

SAFE HELPERS

struct_size() / size_add(), saturation helpers,
struct_group(), TRAILING_OVERLAP(), etc...

LANGUAGE + TOOLCHAIN IMPROVEMENTS

counted_by attribute,
compiler warnings,
KCFI / object size,
language extensions

COMPILE TIME / RUNTIME ENFORCEMENT

FORTIFY_SOURCE, UBSAN_BOUNDS,
KSTACK_ERASE, KCFI,
RANDOMIZE_KSTACK_OFFSET

The present challenges are mostly at the boundaries between layers

LEGACY LAYOUTS

ABI, wire formats, packed structs, and intentional overlaps constrain “clean” rewrites.

COMPILER KNOWLEDGE

Nested FAMs and subobjects still expose object-size blind spots and GCC/Clang differences.

GLOBAL ENFORCEMENT

A warning is useful only when the tree is clean enough to keep it enabled without noise (false positives).

UPSTREAM SCALE

Hundreds of fixes cross subsystem ownership; review and long-tail maintenance dominate the final mile. There is always a social factor.

What comes next: finish enforcement, then make the model richer

1	ENABLE THE FAM WARNING	Drive remaining cases to zero and make -Wflex-array-member-not-at-end a global guardrail.
2	MODEL REAL FAM OVERLAP	Explore compiler support for layouts a source helper alone cannot fully describe.
3	TIE ALLOCATION TO TYPE	type-aware allocations like kmalloc_{obj,objs,flex}() still need more work.
4	IMPROVE OBJECT SIZE	Teach GCC/Clang to retain bounds through nested subobjects so FORTIFY_SOURCE can consume them.

TAKEAWAY

Hardening scales when intent becomes unambiguous

Remove ambiguity → teach the compiler → enforce behavior → keep the tree clean.

The best end state is to shift the burden of security to the toolchain.

Have fewer exceptions, and a compiler that refuses to let the old bug classes come back. :)

Thank you, Seoul! 🇰🇷

Gustavo A. R. Silva
gustavoars@kernel.org
fosstodon.org/@gustavoars
https://embeddedor.com/blog/

By @shidokou

Sponsors

Resources

- **Safer flexible arrays for the kernel (LWN article)**
- **How to use the new counted_by attribute in C (and Linux) (Blogpost)**
- **Enhancing spatial safety: Better array-bounds checking in C (and Linux) (Presentation)**
- **Linux Kernel Hardening: Ten Years Deep (YouTube video)**
- **GCC features to help harden the kernel (LWN article)**
- **https://best.openssf.org/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.html**
- **Open-coded kmalloc assignments for struct objects**
- **Upstream Kernel Hardening: Progress on enabling -Wflex-array-member-not-at-end**