Challenges and innovations towards safer flexible arrays in the Linux Kernel

Gustavo A. R. Silva
gustavoars@kernel.org
fosstodon.org/@gustavoars

Supported by
The Linux Foundation & Alpha-Omega

Lund LinuxCon 10th edition
May 24, 2024
Lund, Sweden

Who am I?

Who am I?

- **Upstream first** – 8 years.
- Upstream Linux Kernel Engineer.
    - Focused on security.

Who am I?

- **Upstream first** – 8 years.
- Upstream Linux Kernel Engineer.
    - Focused on security.

- Kernel Self-Protection Project (**KSPP**).
- Google Open Source Security Team (**GOSST**).
    - Linux Kernel division.

Agenda

- **Introduction**
  - Trailing arrays, flexible arrays, and flexible structures.
  - Flexible-array transformations (FATs)
- **Challenges and innovations**
  - memcpy() and -fstrict-flex-arrays=3
  - __counted_by() and bounds-checking.
  - __builtin_dynamic_object_size()
  - Bleeding-edge kernel hardening.

- **Conclusions**

Trailing arrays

Trailing arrays in the kernel

- Arrays declared at the end of a structure.
- Size determined at **compile-time**.

```
struct trailing {
        ...
        some members;
        ...

        int trailing_array[10];
};
```

Flexible arrays & flexible structures

Flexible arrays & flexible structures

- Flexible array
  - **Trailing** array as **Variable-Length Object** (**VLO**).
  - Size is determined at **run-time**.
- Flexible structure
  - Structure that contains a **flexible array**.

```
struct flex_struct {
        ...
        size_t count;
        struct foo flex_array[];
};
```

Flexible Arrays

Flexible-array members (FAMs)

- Introduced in C99.
- A proper way to declare a flexible array in a struct.
- The last member of an otherwise non-empty struct.

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
};
```

Flexible Arrays

Flexible-array members (FAMs)

- Introduced in C99.
- A proper way to declare a flexible array in a struct.
- The last member of an otherwise non-empty struct.
- The flex struct usually contains a ***counter*** member.

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
};
```

Flexible Arrays

Flexible-array members (FAMs)

- Introduced in C99.
- A proper way to declare a flexible array in a struct.
- The last member of an otherwise non-empty struct.
- The flex struct usually contains a ***counter*** member.
- Should be annotated with __counted_by()

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
};
```

Flexible Arrays

Flexible-array members (FAMs)

- Introduced in C99.
- A proper way to declare a flexible array in a struct.
- The last member of an otherwise non-empty struct.
- The flex struct usually contains a ***counter*** member.
- Should be annotated with __counted_by() (our new baby^.^)

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
};
```

Flexible Arrays

Example

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
} *p;

total_size = sizeof(*p) + sizeof(struct foo) * items;
p = kzalloc(total_size, GFP_KERNEL);
if (!p)
    return;

p->count = items;
```

Flexible Arrays

Example

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
} *p;

total_size = sizeof(*p) + sizeof(struct foo) * items;
p = kzalloc(struct_size(p, flex_array, items), GFP_KERNEL);
if (!p)
    return;

p->count = items;
```

Flexible Arrays

A bit of history (before C99 FAMs).

Flexible Arrays

A bit of history (before C99 FAMs).

```
struct ancient {
   ...
   size_t count;
   struct foo array[1];
} *p1;
```

```
struct old {
   ...
   size_t count;
   struct foo array[0];
} *p0;
```

Flexible Arrays

A bit of history (before C99 FAMs).

- One-element arrays (buggy hack).

```
ancient_size = sizeof(*p1) + sizeof(struct foo) * (items - 1);
```

```
struct ancient {
   ...
   size_t count;
   struct foo array[1];
} *p1;
```

```
struct old {
   ...
   size_t count;
   struct foo array[0];
} *p0;
```

Flexible Arrays

A bit of history (before C99 FAMs).

- One-element arrays (buggy hack).

```
ancient_size = sizeof(*p1) + sizeof(struct foo) * (items - 1);
```

- Zero-length arrays (C90 GNU extension)

```
old_size = sizeof(*p0) + sizeof(struct foo) * items;
```

```
struct ancient {
   ...
   size_t count;
   struct foo array[1];
} *p1;
```

```
struct old {
   ...
   size_t count;
   struct foo array[0];
} *p0;
```

Flexible-array transformations (FATs)

Kick-off of FATs in the Kernel Self-Protection Project

Flexible-array transformations (FATs)

Kick-off of FATs in the Kernel Self-Protection Project

- Undefined Behavior – **The bug**.
- e48f129c2f20 ("[SCSI] cxgb3i: convert cdev->l2opt to ...")

```
struct l2t_data {
        unsigned int nentries;
        struct l2t_entry *rover;
        atomic_t nfree;
        rwlock_t lock;
        struct l2t_entry l2tab[0];
+       struct rcu_head rcu_head;
};
```

Flexible-array transformations (FATs)

Kick-off of FATs in the Kernel Self-Protection Project

- Undefined Behavior – **The bug**.
- e48f129c2f20 ("[SCSI] cxgb3i: convert cdev->l2opt to ...")
- *rcu_head* is overrun at run-time.

```
struct l2t_data {
        unsigned int nentries;
        struct l2t_entry *rover;
        atomic_t nfree;
        rwlock_t lock;
        struct l2t_entry l2tab[0];
+       struct rcu_head rcu_head;
};
```

Flexible-array transformations (FATs)

Kick-off of FATs in the Kernel Self-Protection Project

- Undefined Behavior – **The bugfix**.
- 76497732932f ("cxgb3/l2t: Fix undefined behavior")

```
struct l2t_data {
        unsigned int nentries;
        struct l2t_entry *rover;
        atomic_t nfree;
        rwlock_t lock;
-       struct l2t_entry l2tab[0];
        struct rcu_head rcu_head;
+       struct l2t_entry l2tab[];
};
```

Flexible-array transformations (FATs)

Kick-off of FATs in the Kernel Self-Protection Project

- Undefined Behavior – **The bugfix**.
- 76497732932f ("cxgb3/l2t: Fix undefined behavior")
- **8-year-old bug** introduced in **2011**, and fixed in **2019**.

```
struct l2t_data {
        unsigned int nentries;
        struct l2t_entry *rover;
        atomic_t nfree;
        rwlock_t lock;
-       struct l2t_entry l2tab[0];
        struct rcu_head rcu_head;
+       struct l2t_entry l2tab[];
};
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

memcpy() and -fstrict-flex-arrays=3

- Hardening **memcpy()** and FATs
  - Under CONFIG_FORTIFY_SOURCE=**y**

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {     /* Compile-time */
                if (dst_size < size)
                        __write_overflow();
                if (src_size < size)
                        __read_overflow2();
        }
        ...
}
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

- Under CONFIG_FORTIFY_SOURCE=**y**
- **__builtin_object_size() was** used to determine the size of both **source** and **destination**.

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {     /* Compile-time */
                if (dst_size < size)
                        __write_overflow();
                if (src_size < size)
                        __read_overflow2();
        }
        ...
}
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)
                     __write_overflow();
                if (src_size < size)
                     __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
     ...
     size_t count;
     struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);   == -1 /* __bos() returns -1 */
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)
                     __write_overflow();
                if (src_size < size)
                     __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
     ...
     size_t count;
     struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);   == -1 /* __bos() returns -1 */
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)
                     __write_overflow();
                if (src_size < size)
                     __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
     ...
     size_t count;
     struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);   == -1 /* __bos() returns -1 */
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)  /* in this case, the condition is always false */
                        __write_overflow();
                if (src_size < size)
                        __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);  == -1 /* __bos() returns -1 */
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)  /* in this case, the condition is always false */
                        __write_overflow();
                if (src_size < size)
                        __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.
- Bounds-checking is not possible in this case.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1);  == -1 /* __bos() returns -1 */
        size_t src_size = __builtin_object_size(src, 1);

        if (__builtin_constant_p(size)) {      /* Compile-time */
                if (dst_size < size)  /* in this case, the condition is always false */
                        __write_overflow();
                if (src_size < size)
                        __read_overflow2();
        }
        ...
}
```

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
} *p;

memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.
- Bounds-checking is not possible in this case.
- This is expected behavior.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** returning -1 for any trailing array.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** returning -1 for any trailing array.

```
__builtin_object_size(any_struct->any_trailing_array, 1) == -1
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** returning -1 for any trailing array.

```
struct trailing {
        ...
        some members;
        ...

        int trailing_array[10];
};
```

```
__builtin_object_size(trailing->trailing_array, 1) == -1
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** returning -1 for any trailing array.

```
__builtin_object_size(any_struct->any_trailing_array, 1) == -1
```

memcpy() and -fstrict-flex-arrays=3

- Hardening **memcpy()** and FATs (long story short)
  - **__builtin_object_size()** returning -1 for any trailing array.

```
__builtin_object_size(any_struct->any_trailing_array, 1) == -1
```

Under this scenario *memcpy*() is not able to sanity-check trailing arrays at all.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** returning -1 for any trailing array.

```
__builtin_object_size(any_struct->any_trailing_array, 1) == -1
```

Under this scenario *memcpy*() is not able to sanity-check trailing arrays at all.

But why, exactly?

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** and trailing arrays.
- BSD **sockaddr** (sys/socket.h)
  - char sa_data[**14**]
  - #define SOCK_MAXADDRLEN **255**

```
struct sockaddr {
    unsigned char   sa_len;         /* total length */
    sa_family_t     sa_family;      /* address family */
    char            sa_data[14];    /* actually longer; */
};

#define  SOCK_MAXADDRLEN   255 /* longest possible addresses */
```

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs (long story short)

- **__builtin_object_size()** and trailing arrays.
- https://reviews.llvm.org/D126864

“Some code consider that trailing arrays are flexible, whatever their size. Support for these **legacy** code has been introduced in f8f632498307d22e10fab0704548b270b15f1e1e but **it prevents evaluation of builtin_object_size and builtin_dynamic_object_size in some legit cases**.”

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

- **__builtin_object_size()** and trailing arrays.

So, what do we do about it?

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

- **Compiler side**: Fix it and make it enforce FAMs.

- **Kernel side**: Make flex-array declarations **unambiguous**.

memcpy() and -fstrict-flex-arrays=3

Hardening **memcpy()** and FATs

- **Compiler side**: Fix it and make it enforce FAMs.
  - Fix **__builtin_object_size()**
  - Add new option **-fstrict-flex-arrays**[=n]
  - Enforcing FAMs as the only way to declare flex arrays.
- **Kernel side**: Make flex-array declarations **unambiguous**.
  - Get rid of **fake** flexible arrays ([1] and [0]).
  - Only C99 **flexible-array members** should be used as flexible arrays.

memcpy() and -fstrict-flex-arrays=3

-fstrict-flex-arrays[=n]

memcpy() and -fstrict-flex-arrays=3

-fstrict-flex-arrays[=n] – Released in **GCC-13** and **Clang-16.**

memcpy() and -fstrict-flex-arrays=3

- **-fstrict-flex-arrays[=n]** – Released in **GCC-13** and **Clang-16.**
  - -fstrict-flex-arrays**=3**
    - Only C99 flexible-array members (**[ ]**) are treated VLOs.

```
__builtin_object_size(flex_struct->flex_array_member, 1) == -1
```

memcpy() and -fstrict-flex-arrays=3

-fstrict-flex-arrays[=n] – Released in **GCC-13** and **Clang-16.**

- -fstrict-flex-arrays**=3**
  - Only C99 flexible-array members (**[ ]**) are treated VLOs.

```
__builtin_object_size(flex_struct->flex_array_member, 1) == -1

__bos(any_struct->one_element_array,  1) == sizeof(one_element_array)
__bos(any_struct->zero_length_array,  1) == sizeof(zero_length_array) == 0
__bos(any_struct->any_non_flex_array, 1) == sizeof(any_non_flex_array)
```

With this **ALL trailing arrays of fixed-size gain** bounds-checking. Including [1] & [0], of course.

memcpy() and -fstrict-flex-arrays=3

Fortified **memcpy()** and **-fstrict-flex-arrays=3**

- Globally enabled in **Linux 6.5**. Yeeiii!!

memcpy() and -fstrict-flex-arrays=3

Fortified **memcpy()** and **-fstrict-flex-arrays=3**

- Globally enabled in **Linux 6.5**. Yeeiii!!
- Only C99 flexible-array members are considered to be dynamically sized.
- **The trailing array ambiguity is gone.**

memcpy() and -fstrict-flex-arrays=3

Fortified **memcpy()** and **-fstrict-flex-arrays=3**

- Globally enabled in **Linux 6.5**. Yeeiii!!
- Only C99 flexible-array members are considered to be dynamically sized.
- **The trailing array ambiguity is gone.**

Therefore, we've gained bounds-checking on trailing arrays of **fixed-size.**

Great, but what about bounds-checking on **flexible-array members**?

__counted_by() and bounds-checking on FAMs

- We need a new attribute

__counted_by() and bounds-checking on FAMs

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?

```
struct bounded_flex_struct {
    ...
    size_t elements;
    struct foo array[] __attribute__((counted_by(elements)));
};
```

__counted_by() and bounds-checking on FAMs

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-15** (bugzilla id=108896)

```
#if __has_attribute(__counted_by__)
# define __counted_by(member) __attribute__((__counted_by__(member)))
#else
# define __counted_by(member)
#endif
```

__counted_by() and bounds-checking on FAMs

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-15** (bugzilla id=108896)
- Just released in **Clang-18 !!!** (LLVM id=76348)

```
#if __has_attribute(__counted_by__)
# define __counted_by(member) __attribute__((__counted_by__(member)))
#else
# define __counted_by(member)
#endif
```

__counted_by() and bounds-checking on FAMs

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-15** (bugzilla id=108896)
- Just released in **Clang-18 !!!** (LLVM id=76348)

“Clang now supports the C-only attribute counted_by. When applied to a struct’s flexible array member, **it points to the struct field that holds the number of elements in the flexible array** member. This information can **improve the results of the array bound sanitizer and the __builtin_dynamic_object_size** builtin.”

https://releases.llvm.org/18.1.0/tools/clang/docs/ReleaseNotes.html

__counted_by() and bounds-checking on FAMs

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-15** (bugzilla id=108896)
- Just released in **Clang-18 !!!** (LLVM id=76348)

```
struct bounded_flex_struct {
   ...
   size_t count;
   struct foo array[] __counted_by(count);
};
```

Before we continue - A quick recap

- **-fstrict-flex-arrays=3** prevents any trailing array ambiguity.
- __counted_by() gives accurate context to **-fsanitize=bounds**
- __counted_by() gives accurate context to __**bdos()**
- **CONFIG_UBSAN_BOUNDS** benefits from this.

__builtin_dynamic_object_size()

Fortified **memcpy()** and **__builtin_dynamic_object_size**()

__builtin_dynamic_object_size()

Fortified **memcpy()** and **__builtin_dynamic_object_size**()

- __bdos() **replaced** __builtin_object_size()
- __bdos() adds **run-time coverage** whereas __bos() only covers **compile-time**.
- It **gets hints from** __alloc_size__ and from __counted_by()
- Greater fortification for memcpy().
- **CONFIG_FORTIFY_SOURCE** benefits from this.

Bleeding-edge upstream kernel hardening

Bleeding-edge upstream kernel hardening

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

Bleeding-edge upstream kernel hardening

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

```
struct flex_struct {
    ...
    size_t count;
    struct something flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle; /* suspicious ¬.¬ */

    ...
};
```

Bleeding-edge upstream kernel hardening

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

- We had **~60,000 warnings** in total.

```
struct flex_struct {
    ...
    size_t count;
    struct something flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle; /* suspicious ¬.¬ */

    ...
};
```

Bleeding-edge upstream kernel hardening

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

- We had **~60,000 warnings** in total. Only **650** unique.

```
struct flex_struct {
    ...
    size_t count;
    struct something flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle; /* suspicious ¬.¬ */

    ...
};
```

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

Case 1: Some **FAMs not used at all.**

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

Case 1: Some **FAMs not used at all**.

- f4b09b29f8b4 (“wifi: ti: Avoid a hundred -Wflex-array-member-not-at-end warnings”)

```
struct wl1251_cmd_header {
   u16 id;
   u16 status;
-  /* payload */
-  u8 data[];
 } __packed;
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Two separate structs: original struct & header struct

```
struct flex_struct { /* original struct */
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
};
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Two separate structs: original struct & header struct
- New header struct named after original flex struct.

```
struct flex_struct { /* original struct */
    ...
    size_t count;
    struct foo flex_array[] __counted_by(count);
};

struct flex_struct_hdr { /* NEW header struct */
    ...
    size_t count;
};
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Two separate structs: original struct & header struct

```
struct composite_struct { /* BEFORE */
    ...
    struct flex_struct middle_object; /* FAM in the middle ¬.¬ */
    ...
};

struct composite_struct { /* AFTER */
    ...
    struct flex_struct_hdr middle_object; /* FAM is gone! ^.^ */
    ...
};
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Use struct_group_tagged()/__struct_group()

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Use struct_group_tagged()/__struct_group()

```
struct flex_struct { /* BEFORE */


    ...
    size_t  count;

    struct foo flex_array[] __counted_by(count);
};

struct composite_struct { /* BEFORE */
    ...
    struct flex_struct middle_object; /* FAM in the middle ¬.¬ */
    ...
};
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- Use struct_group_tagged()/__struct_group()

```
struct flex_struct { /* AFTER */
    /* New members must be added within the struct_group() macro below. */
    struct_group_tagged(flex_struct_hdr, hdr,
        ...
        size_t  count;
    );
    struct foo flex_array[] __counted_by(count);
};

struct composite_struct { /* AFTER */
    ...
    struct flex_struct_hdr middle_object; /* FAM is gone! ^.^ */
    ...
};
```

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

Case 2: **FAMs never accessed**. Only the rest of the members in the flex struct are used.

- 5c4250092fad (“wifi: mwl8k: Avoid -Wflex-array-…”)

```
 struct mwl8k_cmd_pkt {
-    __le16  code;
-    __le16  length;
-    __u8    seq_num;
-    __u8    macid;
-    __le16  result;
+    __struct_group(mwl8k_cmd_pkt_hdr, hdr, __packed,
+        __le16  code;
+        __le16  length;
+        __u8    seq_num;
+        __u8    macid;
+        __le16  result;
+    );
     char payload[];
 } __packed;
```

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

- 5c4250092fad (“wifi: mwl8k: Avoid -Wflex-array-…”)
- Replace *mwl8k_cmd_pkt* with *mwl8k_cmd_pkt_hdr*

```
 struct mwl8k_cmd_rf_antenna {
-  struct mwl8k_cmd_pkt header;
+  struct mwl8k_cmd_pkt_hdr header;
    __le16 antenna;
    __le16 mode;
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
struct flex_struct {
    ...
    size_t  count;
    struct foo flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle;
    struct foo fixed_array[MAX_LENGTH];
    ...
} __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
struct flex_struct {
    ...
    size_t  count;
    struct foo flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle;
    struct foo fixed_array[MAX_LENGTH];
    ...
} __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
struct flex_struct {
    ...
    size_t  count;
    struct foo flex_array[] __counted_by(count);
};

struct composite_struct {
    ...

    struct flex_struct flex_in_the_middle;
    struct foo fixed_array[MAX_LENGTH];
    ...
} __packed;
```

- `flex_array` and `fixed_array` share the same address in memory.
- Both form an implicit union.

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
 struct ima_digest_data { /* flexible struct */
+   /* New members must be added within the __struct_group() macro below. */
+   __struct_group(ima_digest_data_hdr, hdr, __packed,
        u8 algo;
        u8 length;
        ...
+   );
    u8 digest[];
 } __packed;
```

```
/* implicit union: FAM & fixed-size array*/
 struct ima_max_digest_data {
-  struct ima_digest_data hdr;
+  struct ima_digest_data_hdr hdr;
   u8 digest[HASH_MAX_DIGESTSIZE];
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
 struct ima_digest_data { /* flexible struct */
+   /* New members must be added within the __struct_group() macro below. */
+   __struct_group(ima_digest_data_hdr, hdr, __packed,
        u8 algo;
        u8 length;
        ...
+   );
    u8 digest[];
 } __packed;
```

```
/* implicit union: FAM & fixed-size array*/
 struct ima_max_digest_data {
-   struct ima_digest_data hdr;
+   struct ima_digest_data_hdr hdr;
    u8 digest[HASH_MAX_DIGESTSIZE];
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

```
 struct ima_digest_data { /* flexible struct */
+   /* New members must be added within the __struct_group() macro below. */
+   __struct_group(ima_digest_data_hdr, hdr, __packed,
        u8 algo;
        u8 length;
        ...
+   );
    u8 digest[];
 } __packed;
```

```
/* implicit union: FAM & fixed-size array*/
 struct ima_max_digest_data {
-   struct ima_digest_data hdr;
+   struct ima_digest_data_hdr hdr;
    u8 digest[HASH_MAX_DIGESTSIZE];
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- However, **FAM digest** is the one accessed at run-time.

```
 struct ima_digest_data { /* flexible struct */
+   /* New members must be added within the __struct_group() macro below. */
+   __struct_group(ima_digest_data_hdr, hdr, __packed,
        u8 algo;
        u8 length;
        ...
+   );
    u8 digest[];
 } __packed;
```

```
/* implicit union: FAM & fixed-size array*/
 struct ima_max_digest_data {
-   struct ima_digest_data hdr;
+   struct ima_digest_data_hdr hdr;
    u8 digest[HASH_MAX_DIGESTSIZE];
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- However, **FAM digest** is the one accessed at run-time.

```
 struct ima_digest_data { /* flexible struct */
+   /* New members must be added within the __struct_group() macro below. */
+   __struct_group(ima_digest_data_hdr, hdr, __packed,
        u8 algo;
        u8 length;
        ...
+   );
    u8 digest[];
 } __packed;
```

```
/* implicit union: FAM & fixed-size array*/
 struct ima_max_digest_data {
-   struct ima_digest_data hdr;
+   struct ima_digest_data_hdr hdr;
    u8 digest[HASH_MAX_DIGESTSIZE];
 } __packed;
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.

```
    struct ima_max_digest_data hash; /* struct with implicit union */
+   struct ima_digest_data *hash_hdr = container_of(&hash.hdr,
+                       struct ima_digest_data, hdr);
```

... **hash_hdr** is now a pointer to flex struct **ima_digest_data**

```
        /* read data from the FAM digest */
-       memcpy(digest_hash, hash.hdr.digest, digest_hash_len);
+       memcpy(digest_hash, hash_hdr->digest, digest_hash_len);
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.

```
    struct ima_max_digest_data hash; /* struct with implicit union */
+   struct ima_digest_data *hash_hdr = container_of(&hash.hdr,
+                       struct ima_digest_data, hdr);
```

... **hash_hdr** is now a pointer to flex struct **ima_digest_data**

```
        /* read data from the FAM digest */
-       memcpy(digest_hash, hash.hdr.digest, digest_hash_len);
+       memcpy(digest_hash, hash_hdr->digest, digest_hash_len);
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.

```
    struct ima_max_digest_data hash; /* struct with implicit union */
+   struct ima_digest_data *hash_hdr = container_of(&hash.hdr,
+                           struct ima_digest_data, hdr);

... hash_hdr is now a pointer to flex struct ima_digest_data

        /* read data from the FAM digest */
-       memcpy(digest_hash, hash.hdr.digest, digest_hash_len);
+       memcpy(digest_hash, hash_hdr->digest, digest_hash_len);
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.

```
    struct ima_max_digest_data hash; /* struct with implicit union */
+   struct ima_digest_data *hash_hdr = container_of(&hash.hdr,
+                           struct ima_digest_data, hdr);

... hash_hdr is now a pointer to flex struct ima_digest_data

        /* read data from the FAM digest */
-       memcpy(digest_hash, hash.hdr.digest, digest_hash_len);
+       memcpy(digest_hash, hash_hdr->digest, digest_hash_len);
```

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

Case 3: **Implicit unions** between FAMs and fixed-size arrays of the same element type.

- Use **container_of()** to get a pointer to the flex struct.
- Access FAM through that pointer.
- 38aa3f5ac6d2 (“integrity: Avoid -Wflex-array-member...”)

```
    struct ima_max_digest_data hash; /* struct with implicit union */
+   struct ima_digest_data *hash_hdr = container_of(&hash.hdr,
+                           struct ima_digest_data, hdr);

... hash_hdr is now a pointer to flex struct ima_digest_data

        /* read data from the FAM digest */
-       memcpy(digest_hash, hash.hdr.digest, digest_hash_len);
+       memcpy(digest_hash, hash_hdr->digest, digest_hash_len);
```

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

Case 4: The same as case 3 but **on-stack**.

- For those we use **DECLARE_FLEX()** and **DECLARE_RAW_FLEX()** helpers.
- Some examples:
  - 6c85a13b133f (“platform/chrome: cros_ec_proto:…”)
  - 4d69c58ef2e4 (“fsnotify: Avoid -Wflex-array-mem…”)
  - 215c4704208b (“Bluetooth: L2CAP: Avoid -Wflex-...”)

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

Patches landed mainline already.

- -Wflex-array-member-not-at-end patches in mainline.
- From 650 to **less than 400 warnings now!** :D
- **~30%** of warnings have been addressed in the last couple of months.

Conclusions

Conclusions

- **-fstrict-flex-arrays=3** enabled in Linux 6.5
- **__counted_by()** attribute is a reality now.
- **__builtin_dynamic_object_size()** increased bounds-checking coverage.
- Gaining **bounds-checking** on **FAMs** is closer than ever!
- **FORTIFY_SOURCE** and **UBSAN_BOUNDS** better every time.

Conclusions

- Vulnerabilities discovered over the last years could’ve been prevented with the most recent version of **memcpy()** and **FORTIFY_SOURCE** updates.
- We’ve been finding and fixing bugs in both **kernel-space** and **user-space**.
- We have a clear strategy to fix **-Wflex-array-member-not-at-end** warnings and enable the option in mainline, soon.
- **The security of the kernel is being significantly improved. :)**

Thank you, Lund!

Gustavo A. R. Silva
gustavoars@kernel.org
fosstodon.org/@gustavoars