

Enhancing spatial safety: Better array-bounds checking in C (and Linux)

Gustavo A. R. Silva
gustavoars@kernel.org
<https://embededor.com/blog/>

Supported by
The Linux Foundation & Alpha-Omega

Open Source Summit Korea
November 4, 2025
Seoul, South Korea

Who am I?



By @shidokou

Who am I?

- **Upstream first** – 9 years.
- Upstream Linux Kernel Engineer.
 - Kernel hardening.
 - Proactive security.



By @shidokou

Who am I?

- **Upstream first** – 9 years.
- Upstream Linux Kernel Engineer.
 - Kernel hardening.
 - Proactive security.
- Kernel Self-Protection Project (**KSPP**).
- Google Open Source Security Team (**GOSST**).
 - Linux Kernel division.



By @shidokou

Agenda

- **Introduction**
 - Fixed-size arrays & trailing arrays
 - Flex arrays, flex structures & flex-array transformations
- **Challenges & innovations towards spatial safety**
 - memcpy() hardening and -fstrict-flex-arrays
 - The new *counted_by* attribute
 - `__builtin_dynamic_object_size()`
 - -Wflex-array-member-not-at-end
- **Conclusions**

Fixed-size arrays

```
int fixed_size_array[10];
```

Fixed-size arrays

- Simple declaration of an array of fixed size.
- C doesn't enforce array's boundaries.
- It's up to the developers to enforce them.

```
int fixed_size_array[10];
```

Fixed-size arrays

- Simple declaration of an array of fixed size.
- C doesn't enforce array's boundaries.
- It's up to the developers to enforce them.
- Size determined at **compile time**.

```
int fixed_size_array[10];
```

Trailing arrays

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10];  
};
```

Trailing arrays

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

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10];  
};
```

Flexible arrays & flexible structures

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

Flexible arrays & flexible structures

- Flexible array
 - **Trailing** array whose size is determined at **run time**.

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

Flexible arrays & flexible structures

- Flexible array
 - **Trailing** array whose 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 structures

- We use a flexible array when we know the size of the trailing array is going to be dynamic.

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

C99 Flexible-Array Members

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)

```
int variable_length_array[n_items];
```

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**

```
struct flex_struct {  
    ...  
    size_t count;  
    struct foo one_element_array[1];  
};
```

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**

```
struct flex_struct {  
    ...  
    size_t count;  
    struct foo zero_length_array[0];  
};
```

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**
- The **last member** in the flex structure –enforced by compilers

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

C99 Flexible-Array Members

- A proper way to declare a flexible array in a struct.
- Not to be confused with Variable Length Arrays (-Wvla)
- Before C99 people would use **[1]** & **[0]**
- The **last member** in the flex structure –enforced by compilers
- The flex struct usually contains a ***counter*** member.

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

Flexible arrays & flexible structures

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

Flexible arrays & flexible structures

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

```
struct flex_struct *p;  
size_t total_size = sizeof(*p) + sizeof(struct foo) * items;
```

```
p = kzalloc(total_size, GFP_KERNEL);  
if (!p)  
    return;
```

```
p->count = items;
```

Flexible arrays & flexible structures

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

```
struct flex_struct *p;  
size_t 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 & flexible structures

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

- struct_size() returns

```
#define SIZE_MAX (~(size_t)0)
```

on overflow.

```
struct flex_struct *p;  
size_t total_size = sizeof(*p) + sizeof(struct foo) * items;
```

```
p = kzalloc(struct_size(p, flex_array, items), GFP_KERNEL);  
if (!p)  
    return;
```

```
p->count = items;
```

Then one day...

Flexible-array transformations (FATs)

```
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)

```
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)

- 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)

- 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)

- 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[];
};
```

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 l2t_entry l2tab[];
};
```

Challenges & Innovations Towards Spatial Safety

BleedingTooth: Linux Bluetooth Zero-Click Remote Code Execution

“[...] allow an unauthenticated remote attacker in short distance to execute arbitrary code with kernel privileges on vulnerable devices.”

BadVibes

(a BleedingTooth vulnerability - 2020)

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

- `len` is not sanity-checked before calling `memcpy()`.

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

- `len` is not sanity-checked before calling `memcpy()`.
- That's not great. :/

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

```
#define HCI_MAX_AD_LENGTH    31

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

“The parser can theoretically receive and route a packet **up to 255 bytes** to this method. If that is possible, we could **overflow *last_adv_data*** and corrupt members up to offset 0xbaf.”

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

BadVibes (a BleedingTooth vulnerability - 2020)

- **last_adv_data** is overflowed and list_head pointers corrupted.

```
#define HCI_MAX_AD_LENGTH    31

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

“The parser can theoretically receive and route a packet **up to 255 bytes** to this method. If that is possible, we could **overflow last_adv_data** and corrupt members up to offset 0xbaf.”

```
static void store_pending_adv_report(..., u8
*data, u8 len)
{
    struct discovery_state *d = ...;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

What's the problem with **memcpy()**?

What's the problem with memcpy()?

- **memcpy()** doesn't know about your true intentions.
- You can read and write data **out of bounds** without restriction.

```
void *memcpy(void *dst, const void *src, size_t size)
```

What's the problem with memcpy()?

- **memcpy()** doesn't know about your true intentions.
- You can read and write data **out of bounds** without restriction.
- It's up to the developers to enforce boundaries for **src** and **dst** before calling **memcpy()**.

```
void *memcpy(void *dst, const void *src, size_t size)
```

So, what was the fix for **BadVibes**?

What's the problem with memcpy()?

- a2ec905d1e16 (“Bluetooth: fix kernel oops in store_...”)

```
static void store_pending_adv_report(...)  
{  
    struct discovery_state *d = ...;  
  
+   if (len > HCI_MAX_AD_LENGTH)  
+       return;  
+  
    ...  
    memcpy(d->last_adv_data, data, len);  
    ...  
}
```

What's the problem with memcpy()?

- a2ec905d1e16 (“Bluetooth: fix kernel oops in store_...”)

```
static void store_pending_adv_report(...)  
{  
    struct discovery_state *d = ...;  
  
+   if (len > HCI_MAX_AD_LENGTH)  
+       return;  
+  
    ...  
    memcpy(d->last_adv_data, data, len);  
    ...  
}
```

memcpy() internals

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

“Fortified” memcpy() (before BadVibes)

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

`__builtin_object_size()`

Hardening memcpy()

__builtin_object_size(OBJ, MODE)

- MODE 0: bytes to the end of the *outer struct*
- MODE 1: bytes to the end of *struct member*

Hardening memcpy()

__builtin_object_size(OBJ, MODE)

- MODE 0: bytes to the end of the *outer struct*
- MODE 1: bytes to the end of *struct member*

```
struct foo {  
    int count;      /* 4 bytes */  
    char name[8];  /* 8 bytes */  
    int secret;    /* 4 bytes */  
    char blob[];   /* flexible array */  
} *instance;      /* 16 bytes total */
```

```
__builtin_object_size(&instance->count, 0) == 16  
__builtin_object_size(instance->name, 0) == 12  
__builtin_object_size(instance->blob, 0) == -1
```

```
__builtin_object_size(&instance->count, 1) == 4  
__builtin_object_size(instance->name, 1) == 8  
__builtin_object_size(instance->blob, 1) == -1
```

Hardening memcpy()

__builtin_object_size(OBJ, MODE)

- MODE 0: bytes to the end of the *outer struct*
- MODE 1: bytes to the end of *struct member*

```
struct foo {  
    int count;      /* 4 bytes */  
    char name[8];  /* 8 bytes */  
    int secret;    /* 4 bytes */  
    char blob[];   /* flexible array */  
} *instance;      /* 16 bytes total */
```

```
__builtin_object_size(&instance->count, 0) == 16
```

```
__builtin_object_size(instance->name, 0) == 12
```

```
__builtin_object_size(instance->blob, 0) == -1
```

```
__builtin_object_size(&instance->count, 1) == 4
```

```
__builtin_object_size(instance->name, 1) == 8
```

```
__builtin_object_size(instance->blob, 1) == -1
```

Hardening memcpy()

__builtin_object_size(OBJ, MODE)

- MODE 0: bytes to the end of the *outer struct*
- MODE 1: bytes to the end of *struct member*

```
struct foo {
    int count;      /* 4 bytes */
    char name[8];  /* 8 bytes */
    int secret;    /* 4 bytes */
    char blob[];   /* flexible array */
} *instance;      /* 16 bytes total */

__builtin_object_size(&instance->count, 0) == 16
__builtin_object_size(instance->name, 0) == 12
__builtin_object_size(instance->blob, 0) == -1

__builtin_object_size(&instance->count, 1) == 4
__builtin_object_size(instance->name, 1) == 8
__builtin_object_size(instance->blob, 1) == -1
```

Hardening memcpy()

__builtin_object_size(OBJ, MODE)

- MODE 0: bytes to the end of the *outer struct*
- MODE 1: bytes to the end of *struct member*

```
struct foo {
    int count;      /* 4 bytes */
    char name[8];  /* 8 bytes */
    int secret;    /* 4 bytes */
    char blob[];   /* flexible array */
} *instance;      /* 16 bytes total */

__builtin_object_size(&instance->count, 0) == 16
__builtin_object_size(instance->name, 0) == 12
__builtin_object_size(instance->blob, 0) == -1

__builtin_object_size(&instance->count, 1) == 4
__builtin_object_size(instance->name, 1) == 8
__builtin_object_size(instance->blob, 1) == -1
```

Hardening memcpy()

```
__builtin_object_size(d->last_adv_data, 0)
```

- __bos() returns the number of bytes from **last_adv_data** to the end of **struct hci_dev**

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

static void store_...(struct hci_dev *hdev, ...)
{
    struct discovery_state *d = &hdev->discovery;
    ...
    memcpy(d->last_adv_data, data, len);
    ...
}
```

Hardening memcpy()

```
__builtin_object_size(d->last_adv_data, 0)
```

- __bos() returns the number of bytes from **last_adv_data** to the end of **struct hci_dev**

```
struct hci_dev {
```

```
    ...  
    struct discovery_state {
```

```
        ...  
        u8 last_adv_data[HCI_MAX_AD_LENGTH];  
        ...  
    } discovery;
```

```
    ...  
    struct list_head {  
        struct list_head *next;  
        struct list_head *prev;  
    } mgmt_pending;
```

```
    ...
```

```
};
```

```
static void store_...(struct hci_dev *hdev, ...)  
{  
    struct discovery_state *d = &hdev->discovery;  
    ...  
    memcpy(d->last_adv_data, data, len);  
    ...  
}
```

What can we do about it?

Hardening memcpy()

What can we do about it?

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

What can we do about it?

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
    size_t dst_size = __builtin_object_size(dst, 0);
    size_t src_size = __builtin_object_size(src, 0);

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

Hardening memcpy()

What can we do about it?

- Replace `__bos(0)` with `__bos(1)`

```
__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();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

```
size_t dst_size = __builtin_object_size(dst, 1);  
size_t src_size = __builtin_object_size(src, 1);
```

This is enough to prevent **BadVibes-like vulnerabilities.** :)

```
size_t dst_size = __builtin_object_size(dst, 1);  
size_t src_size = __builtin_object_size(src, 1);
```

This is enough to prevent **BadVibes-like vulnerabilities.** :)

Life is beautiful! ^.^

OK, but...

OK, but...

what about **intentional cross-member overflows?**

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {  
    ...  
  
    __u8 key_material[MAX_ENCR_KEY_LENGTH];  
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {
    ...

    __u8 key_material[MAX_ENCR_KEY_LENGTH];
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];

    ...
};

keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;
...
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {
    ...

    __u8 key_material[MAX_ENCR_KEY_LENGTH];
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];

    ...
};

keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;
...
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {  
    ...  
    __u8 key_material[MAX_ENCR_KEY_LENGTH];  
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {
    ...

    __u8 key_material[MAX_ENCR_KEY_LENGTH];
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];

    ...
};

keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;
...
memcpy(cmd->key_material, key->key, keymlen);
```

Linux kernel hardening & False positives

- Usually hundreds and even thousands
- FPs usually expose weak or ambiguous code

```
memcpy(cmd->key_material, key->key, keymlen);
```

Linux kernel hardening & False positives

- Usually hundreds and even thousands
- FPs usually expose weak or ambiguous code
- They waste people's time
- They should be fixed

```
memcpy(cmd->key_material, key->key, keymlen);
```

Linux kernel hardening & False positives

- Usually hundreds and even thousands
- FPs usually expose weak or ambiguous code
- They waste people's time
- They should be fixed
- We allocate that pain in the KSPP

```
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Linux had many intentional cross-member **memcpy()** overflows

```
struct mw18k_cmd_set_key {  
    ...  
  
    __u8 key_material[MAX_ENCR_KEY_LENGTH];  
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

Could be fixed by simply adding a named sub-struct

```
struct mwl8k_cmd_set_key {  
    ...  
    struct {  
        __u8 key_material[MAX_ENCR_KEY_LENGTH];  
        __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
        __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
    } tkip;  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(&cmd->tkip, key->key, keymlen);
```

Intentional cross-member overflows

Could be fixed by simply adding a named sub-struct

```
struct mwl8k_cmd_set_key {  
    ...  
    struct {  
        __u8 key_material[MAX_ENCR_KEY_LENGTH];  
        __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
        __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
    } tkip;  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(&cmd->tkip, key->key, keymlen);
```

Intentional cross-member overflows

But now everything must include the name of the sub-struct

```
struct mwl8k_cmd_set_key {  
    ...  
    struct {  
        __u8 key_material[MAX_ENCR_KEY_LENGTH];  
        __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
        __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
    } tkip;  
    ...  
};
```

diff ...

```
- do_something_with(cmd->key_material);  
+ do_something_with(cmd->tkip.key_material);
```

Intentional cross-member overflows

struct_group() was invented to provide both

```
struct mwl8k_cmd_set_key {
    ...
    struct_group(tkip,
        __u8 key_material[MAX_ENCR_KEY_LENGTH];
        __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];
        __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];
    };
    ...
};
```

/* Accessible either way: */

```
do_something_with(cmd->key_material);
do_something_with(cmd->tkip.key_material);
```

The `struct_group()` helper macro

Created by Kees Cook and Keith Packard

```
#define struct_group(NAME, MEMBERS...) \
    union { \
        struct { MEMBERS }; \
        struct { MEMBERS } NAME; \
    }
```

The `struct_group()` helper macro

Created by Kees Cook and Keith Packard

- `struct_group_tagged()`, `struct_group_attr()` & `__struct_group()`

```
#define struct_group(NAME, MEMBERS...) \
    union { \
        struct { MEMBERS }; \
        struct { MEMBERS } NAME; \
    }
```

The `struct_group()` helper macro

Created by Kees Cook and Keith Packard

- `struct_group_tagged()`, `struct_group_attr()` & `__struct_group()`
- Access each member **directly** or through the named struct.

```
#define struct_group(NAME, MEMBERS...) \
    union { \
        struct { MEMBERS }; \
        struct { MEMBERS } NAME; \
    }
```

The `struct_group()` helper macro

Created by Kees Cook and Keith Packard

- `struct_group_tagged()`, `struct_group_attr()` & `__struct_group()`
- Access each member **directly** or through the named struct.
- **Gain bounds-checking** on the group as a whole.

```
#define struct_group(NAME, MEMBERS...) \
    union { \
        struct { MEMBERS }; \
        struct { MEMBERS } NAME; \
    }
```

Intentional cross-member overflows

struct_group() provides the compiler with an identifier for the whole group of members.

```
struct mw18k_cmd_set_key {  
    ...  
    __u8 key_material[MAX_ENCR_KEY_LENGTH];  
    __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
    __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
    ...  
};  
  
keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;  
...  
memcpy(cmd->key_material, key->key, keymlen);
```

Intentional cross-member overflows

struct_group() provides the compiler with an identifier for the whole group of members.

```
struct mwl8k_cmd_set_key {  
    ...  
+   struct_group(tkip,  
                __u8 key_material[MAX_ENCR_KEY_LENGTH];  
                __u8 tkip_tx_mic_key[MIC_KEY_LENGTH];  
                __u8 tkip_rx_mic_key[MIC_KEY_LENGTH];  
+   };  
    ...  
};
```

```
- keymlen = MAX_ENCR_KEY_LENGTH + 2 * MIC_KEY_LENGTH;
```

```
...
```

```
- memcpy(cmd->key_material, key->key, keymlen);
```

```
+ memcpy(&cmd->tkip, key->key, sizeof(cmd->tkip));
```

With **struct_group()** we avoid false positives, gain bounds-checking and can use **__builtin_object_size(1)!** :D

Hardening memcpy()

- Now we can use `__builtin_object_size(1)`

```
__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();
    }
    ...
}
```

Hardening memcpy()

- Now we can use `__builtin_object_size(1)`
- Life's still beautiful. ^.^

```
__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();
    }
    ...
}
```

We were really happy :)

We were really happy...

Then something terrible happened! D:

Let's take another look at flexible-array
members & `__bos(1)`

__builtin_object_size() & flexible arrays

```
struct foo {  
    int count;        /* 4 bytes */  
    char name[8];    /* 8 bytes */  
    int secret;      /* 4 bytes */  
    char blob[];     /* flexible array */  
} *instance;        /* 16 bytes total */
```

```
__builtin_object_size(&instance->count, 0) == 16
```

```
__builtin_object_size(instance->name, 0) == 12
```

```
__builtin_object_size(instance->blob, 0) == -1
```

```
__builtin_object_size(&instance->count, 1) == 4
```

```
__builtin_object_size(instance->name, 1) == 8
```

```
__builtin_object_size(instance->blob, 1) == -1
```

memcpy() & flexible arrays

```
struct flex_struct {
    ...
    size_t count;
    struct foo flex_array[];
} *p;
...
memcpy(p->flex_array, &source, SOME_SIZE);

__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();
    }
    ...
    return __underlying_memcpy(dst, src, size);
}
```

memcpy() & flexible arrays

```
struct flex_struct {  
    ...  
    size_t count;  
    struct foo flex_array[];  
} *p;  
...  
memcpy(p->flex_array, &source, SOME_SIZE);
```

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

memcpy() & flexible arrays

```
struct flex_struct {  
    ...  
    size_t count;  
    struct foo flex_array[];  
} *p;  
...  
memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

memcpy() & flexible arrays

```
struct flex_struct {  
    ...  
    size_t count;  
    struct foo flex_array[];  
} *p;  
...  
memcpy(p->flex_array, &source, SOME_SIZE);
```

- FAMs are objects of incomplete type.

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

memcpy() & flexible arrays

```
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.

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

memcpy() & flexible arrays

```
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.

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

memcpy() & flexible arrays

```
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.
- All this is expected behavior.

```
__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();  
    }  
    ...  
    return __underlying_memcpy(dst, src, size);  
}
```

However... (and here comes the terrible
thing)

`__builtin_object_size()` & trailing arrays

__builtin_object_size() & trailing arrays

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10];  
} *p;
```

__builtin_object_size() & trailing arrays

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10];  
} *p;
```

__builtin_object_size() & trailing arrays

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10]; /* == 40 bytes */  
} *p;
```

__builtin_object_size() & trailing arrays

- For some reason `__bos(1)` returned `-1` for trailing arrays **of any size**.

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10]; /* == 40 bytes */  
} *p;
```

__builtin_object_size() & trailing arrays

- For some reason `__bos(1)` returned `-1` for trailing arrays of any size.

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10]; /* == 40 bytes */  
} *p;
```

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

__builtin_object_size() & trailing arrays

- For some reason `__bos(1)` returned `-1` for trailing arrays **of any size**.

```
struct foo {  
    ...  
    some members;  
    ...  
    int trailing_array[10]; /* == 40 bytes */  
} *p;
```

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

__builtin_object_size() & trailing arrays

- For some reason `__bos(1)` returned `-1` for trailing arrays **of any size**.

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

__builtin_object_size() & trailing arrays

- For some reason `__bos(1)` returned `-1` for trailing arrays of any size.

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

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

But why, exactly?

__builtin_object_size() & trailing arrays

- BSD `sockaddr` (`sys/socket.h`)

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

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

__builtin_object_size() & trailing arrays

- BSD `sockaddr` (`sys/socket.h`)

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

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

__builtin_object_size() & trailing arrays

- BSD `sockaddr` (`sys/socket.h`)

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

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

“A feature, not a bug”

– <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.”

“A feature, not a bug”

- <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.”

“A feature, not a bug”

- <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.**”

“A feature, not a bug”

– <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.”

`__builtin_object_size()` & trailing arrays

So, what can we do about it?

FATs & memcpy() & -fstrict-flex-arrays

- **Compiler side:** Fix it and make it enforce FAMs.
- **Kernel side:** Make flex-array declarations **unambiguous**.

FATs & memcpy() & -fstrict-flex-arrays

- **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**.

FATs & memcpy() & -fstrict-flex-arrays

- **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]` & `[0]`).
 - Only C99 **flexible-array members** should be used as flexible arrays.

FATs & memcpy() & -fstrict-flex-arrays

- **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]` & `[0]`).
 - Only C99 **flexible-array members** should be used as flexible arrays.
 - **Flexible-Array Transformations.**

Gaining bounds-checking on trailing arrays

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

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=0 (default)**

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=0 (default)**
 - **All** trailing arrays are treated as flex arrays.

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

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=0 (default)**
 - **All** trailing arrays are treated as flex arrays.

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

Everything remains the **same**.

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=1**

Gaining bounds-checking on trailing arrays

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

– **-fstrict-flex-arrays=1**

- Only **[1]**, **[0]** and **[]** are treated as flex arrays.

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

Gaining bounds-checking on trailing arrays

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

– **-fstrict-flex-arrays=1**

- Only **[1]**, **[0]** and **[]** are treated as flex arrays.

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

Now **fixed-size** trailing arrays (except **[1]** & **[0]**, of course) **gain** bounds-checking. :)

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=2**

Gaining bounds-checking on trailing arrays

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

– **-fstrict-flex-arrays=2**

- Only **[0]** and **[]** are treated as flex arrays.

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

Gaining bounds-checking on trailing arrays

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

– **-fstrict-flex-arrays=2**

- Only **[0]** and **[]** are treated as flex arrays.

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

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

Gaining bounds-checking on trailing arrays

- **-fstrict-flex-arrays[=n]** – Supported in **GCC-13** and **Clang-16**.
- **-fstrict-flex-arrays=2**
 - Only **[0]** and **[]** are treated as flex arrays.

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

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

Now **fixed-size** trailing arrays (except **[0]**, of course) **gain** bounds-checking. :)

Gaining bounds-checking on trailing arrays

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

Now what's left to be resolved is the case for **zero-length arrays**.

Gaining bounds-checking on trailing arrays

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

Now what's left to be resolved is the case for **zero-length arrays**.

Could that probably be resolved with **-fstrict-flex-arrays=3** ? Maybe?

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`
 - **-Wzero-length-array** (thousands of warnings, as usual)

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`
 - **-Wzero-length-array** (thousands of warnings, as usual)
 - 0-length arrays are not only used as fake flex-arrays.

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`
 - **-Wzero-length-array** (thousands of warnings, as usual)
 - 0-length arrays are not only used as fake flex-arrays.
 - They are used as markers in structs.
 - Under certain configurations some arrays end up having a size zero.

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`
 - **-Wzero-length-array** (thousands of warnings, as usual)
 - 0-length arrays are not only used as fake flex-arrays.
 - They are used as markers in structs.
 - Under certain configurations some arrays end up having a size zero.
 - **So, 0-length arrays are here to stay, but not as VLOs.**

Gaining bounds-checking on trailing arrays

- The case of **Clang** vs `-fstrict-flex-arrays=3`
 - **-Wzero-length-array** (thousands of warnings, as usual)
 - 0-length arrays are not only used as fake flex-arrays.
 - They are used as markers in structs.
 - Under certain configurations some arrays end up having a size zero.
 - **So, 0-length arrays are here to stay, but not as VLOs.**

Fortunately, this issue was promptly resolved. :)

memcpy() & -fstrict-flex-arrays

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

memcpy() & -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 as VLOs.

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

memcpy() & -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 as VLOs.

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

```
__bos(any_struct->any_non_true_flex_array, 1) == sizeof(any_non_true_flex_array)
```

With this **ALL** trailing arrays of fixed size gain bounds-checking.

memcpy() & -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 as VLOs.

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

```
__bos(any_struct->any_non_true_flex_array, 1) == sizeof(any_non_true_flex_array)
```

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

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

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

Ambiguous flex-array declarations

Ambiguous flex-array declarations

Fake flexible arrays.

- One-element arrays (**buggy hack**).
- Zero-length arrays (**GNU extension**).

Ambiguous flex-array declarations

Fake flexible arrays.

- One-element arrays (**buggy hack**).
- Zero-length arrays (**GNU extension**).

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
};
```

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
};
```

Ambiguous flex-array declarations

True flexible arrays.

- “Modern” C99 flexible-array member.
- The last member of an otherwise non-empty structure.

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

Problems with 1-element arrays

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

Problems with 1-element arrays

- Prone to **off-by-one** problems.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

Problems with 1-element arrays

- Prone to **off-by-one** problems.
- Always “contribute” with **size-of-one-element** to the size of the enclosing structure.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

Problems with 1-element arrays

- Prone to **off-by-one** problems.
- Always “contribute” with **size-of-one-element** to the size of the enclosing structure.
- Developers have to remember to subtract **1** from **count**, or **sizeof(struct foo)** from **sizeof(struct fake_flex_1)**.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

```
alloc_size = sizeof(*p) + sizeof(struct foo) * (count - 1);  
p = kmalloc(alloc_size, GFP_KERNEL)  
p->count = count;
```

Problems with 1-element arrays

- Prone to **off-by-one** problems.
- Always “contribute” with **size-of-one-element** to the size of the enclosing structure.
- Developers have to remember to subtract **1** from **count**, or **sizeof(struct foo)** from **sizeof(struct fake_flex_1)**.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

```
alloc_size = sizeof(*p) + sizeof(struct foo) * (count - 1);  
p = kmalloc(alloc_size, GFP_KERNEL)  
p->count = count;
```

Problems with 1-element arrays

- *-Warray-bounds=2* false positives.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

```
...  
for(i = 0; i < 10; i++)  
    p->fake_flex[i] = thing;
```

Problems with 1-element arrays

- **-Warray-bounds=2** false positives.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

```
...  
for(i = 0; i < 10; i++)  
    p->fake_flex[i] = thing;
```

```
i == 0 is fine :)  
i >= 1 is not :/
```

Problems with 1-element arrays

- **-Warray-bounds=2** false positives.

```
struct fake_flex_1 {  
    ...  
    size_t count;  
    struct foo fake_flex[1];  
} *p;
```

```
...  
for(i = 0; i < 10; i++)  
    p->fake_flex[i] = thing;    i == 0 is fine :)  
                                i >= 1 is not :/
```

warning: array subscript 1 is above array bounds of 'struct foo[1]' [-Warray-bounds]

GNU extension: 0-length arrays

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
} *p;
```

GNU extension: 0-length arrays

- Not part of the C standard –90s Compiler extension

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
} *p;
```

GNU extension: 0-length arrays

- Not part of the C standard –90s Compiler extension
- Size is zero –may add trailing padding to the struct

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
} *p;
```

GNU extension: 0-length arrays

- Not part of the C standard –90s Compiler extension
- Size is zero –may add tailing padding to the struct

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
} *p;
```

```
alloc_size = sizeof(*p) + sizeof(struct foo) * count;  
p = kmalloc(alloc_size, GFP_KERNEL)  
p->count = count;
```

GNU extension: 0-length arrays

- Not part of the C standard –90s Compiler extension
- Size is zero –may add tailing padding to the struct
- Slightly less buggy, but still...
- Be aware of `sizeof(p->fake_flex) == 0`

```
struct fake_flex_0 {  
    ...  
    size_t count;  
    struct foo fake_flex[0];  
} *p;
```

```
alloc_size = sizeof(*p) + sizeof(struct foo) * count;  
p = kmalloc(alloc_size, GFP_KERNEL)  
p->count = count;
```

FATs - The case of UAPI

FATs - The case of UAPI

```
include/uapi/linux/in.h:
```

```
struct ip_msfilter {  
    __be32          imsf_multiaddr;  
    __be32          imsf_interface;  
    __u32           imsf_fmode;  
    __u32           imsf_numsrc;  
    __be32          imsf_slist[1];  
};
```

FATs - The case of UAPI

```
include/uapi/linux/in.h:
```

```
struct ip_msfilter {  
    __be32          imsf_multiaddr;  
    __be32          imsf_interface;  
    __u32           imsf_fmode;  
    __u32           imsf_numsrc;  
    __be32          imsf_slist[1];  
};
```

FATs - The case of UAPI

- Cannot simply change the size of structs in UAPI

```
include/uapi/linux/in.h:
```

```
struct ip_msfilter {  
    __be32          imsf_multiaddr;  
    __be32          imsf_interface;  
    __u32           imsf_fmode;  
    __u32           imsf_numsrc;  
    __be32          imsf_slist[1];  
};
```

FATs - The case of UAPI

- Cannot simply change the size of structs in UAPI
- We never break user-space

```
include/uapi/linux/in.h:  
struct ip_msfilter {  
    __be32          imsf_multiaddr;  
    __be32          imsf_interface;  
    __u32           imsf_fmode;  
    __u32           imsf_numsrc;  
    __be32          imsf_slist[1];  
};
```

FATs - The case of UAPI

- Cannot simply change the size of structs in UAPI
- We never break user-space –on purpose ^.^

```
include/uapi/linux/in.h:
```

```
struct ip_msfilter {  
    __be32          imsf_multiaddr;  
    __be32          imsf_interface;  
    __u32           imsf_fmode;  
    __u32           imsf_numsrc;  
    __be32          imsf_slist[1];  
};
```

FATs - The case of UAPI

```
struct ip_msfilter {
-   __be32      imsf_multiaddr;
-   __be32      imsf_interface;
-   __u32       imsf_fmode;
-   __u32       imsf_numsrc;
-   __be32      imsf_slist[1];
+   union {
+       struct {
+           __be32      imsf_multiaddr_aux;
+           __be32      imsf_interface_aux;
+           __u32       imsf_fmode_aux;
+           __u32       imsf_numsrc_aux;
+           __be32      imsf_slist[1];
+       };
+       struct {
+           __be32      imsf_multiaddr;
+           __be32      imsf_interface;
+           __u32       imsf_fmode;
+           __u32       imsf_numsrc;
+           __be32      imsf_slist_flex[];
+       };
+   };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – First attempts.

- Duplicate the original struct within a **union**.

```
struct ip_msfilter {  
-     __be32      imsf_multiaddr;  
-     __be32      imsf_interface;  
-     __u32       imsf_fmode;  
-     __u32       imsf_numsrc;  
-     __be32      imsf_slist[1];  
+     union {  
+         struct {  
+             __be32      imsf_multiaddr_aux;  
+             __be32      imsf_interface_aux;  
+             __u32       imsf_fmode_aux;  
+             __u32       imsf_numsrc_aux;  
+             __be32      imsf_slist[1];  
+         };  
+         struct {  
+             __be32      imsf_multiaddr;  
+             __be32      imsf_interface;  
+             __u32       imsf_fmode;  
+             __u32       imsf_numsrc;  
+             __be32      imsf_slist_flex[];  
+         };  
+     };  
};
```

FATs - The case of UAPI

- One-element array will be used by **user-space**.
- Flexible-array will be used by **kernel-space**.

```
    struct ip_msfilter {
-       __be32      imsf_multiaddr;
-       __be32      imsf_interface;
-       __u32       imsf_fmode;
-       __u32       imsf_numsrc;
-       __be32      imsf_slist[1];
+       union {
+           struct {
+               __be32      imsf_multiaddr_aux;
+               __be32      imsf_interface_aux;
+               __u32       imsf_fmode_aux;
+               __u32       imsf_numsrc_aux;
+               __be32      imsf_slist[1];
+           };
+           struct {
+               __be32      imsf_multiaddr;
+               __be32      imsf_interface;
+               __u32       imsf_fmode;
+               __u32       imsf_numsrc;
+               __be32      imsf_slist_flex[];
+           };
+       };
    };
};
```

FATs - The case of UAPI

- One-element array will be used by **user-space**.
- Flexible-array will be used by **kernel-space**.

```
struct ip_msfilter {
-     __be32      imsf_multiaddr;
-     __be32      imsf_interface;
-     __u32       imsf_fmode;
-     __u32       imsf_numsrc;
-     __be32      imsf_slist[1];
+     union {
+         struct {
+             __be32      imsf_multiaddr_aux;
+             __be32      imsf_interface_aux;
+             __u32       imsf_fmode_aux;
+             __u32       imsf_numsrc_aux;
+             __be32      imsf_slist[1];
+         };
+         struct {
+             __be32      imsf_multiaddr;
+             __be32      imsf_interface;
+             __u32       imsf_fmode;
+             __u32       imsf_numsrc;
+             __be32      imsf_slist_flex[];
+         };
+     };
};
```

FATs - The case of UAPI

- One-element array will be used by **user-space**.
- Flexible-array will be used by **kernel-space**.

```
struct ip_msfilter {
-     __be32      imsf_multiaddr;
-     __be32      imsf_interface;
-     __u32       imsf_fmode;
-     __u32       imsf_numsrc;
-     __be32      imsf_slist[1];
+     union {
+         struct {
+             __be32      imsf_multiaddr_aux;
+             __be32      imsf_interface_aux;
+             __u32       imsf_fmode_aux;
+             __u32       imsf_numsrc_aux;
+             __be32      imsf_slist[1];
+         };
+         struct {
+             __be32      imsf_multiaddr;
+             __be32      imsf_interface;
+             __u32       imsf_fmode;
+             __u32       imsf_numsrc;
+             __be32      imsf_slist_flex[];
+         };
+     };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the **__DECLARE_FLEX_ARRAY()** helper in a union.

```
struct ip_msfilter {
    __be32          imsf_multiaddr;
    __be32          imsf_interface;
    __u32           imsf_fmode;
    __u32           imsf_numsrc;
    union {
        __be32          imsf_slist[1];
        __DECLARE_FLEX_ARRAY(__be32, imsf_slist_flex);
    };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the `__DECLARE_FLEX_ARRAY()` helper in a union.

```
struct ip_msfilter {
    __be32          imsf_multiaddr;
    __be32          imsf_interface;
    __u32           imsf_fmode;
    __u32           imsf_numsrc;
    union {
        __be32
        __be32          imsf_slist[1];
        imsf_slist_flex[];
    };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the `__DECLARE_FLEX_ARRAY()` helper in a union.
- FAMs in unions –GCC >= 15 (April 2025)

```
struct ip_msfilter {
    __be32          imsf_multiaddr;
    __be32          imsf_interface;
    __u32           imsf_fmode;
    __u32           imsf_numsrc;
    union {
        __be32
        __be32          imsf_slist[1];
        imsf_slist_flex[];
    };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the **__DECLARE_FLEX_ARRAY()** helper in a union.
- FAMs in unions –GCC >= 15 (April 2025)

```
struct ip_msfilter {
    __be32          imsf_multiaddr;
    __be32          imsf_interface;
    __u32           imsf_fmode;
    __u32           imsf_numsrc;
    union {
        __be32          imsf_slist[1];
        __DECLARE_FLEX_ARRAY(__be32, imsf_slist_flex);
    };
};
```

FATs - The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the `__DECLARE_FLEX_ARRAY()` helper in a union.
- FAMs in unions –GCC >= 15 (April 2025)

```
#define __DECLARE_FLEX_ARRAY(TYPE, NAME) \
    struct { \
        struct { } __empty_ ## NAME; \
        TYPE NAME[]; \
    }
```

FATs & memcpy() & -fstrict-flex-arrays=3

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

FATs & memcpy() & -fstrict-flex-arrays=3

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

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

FATs & memcpy() & -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.

FATs & memcpy() & -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.**

FATs & memcpy() & -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! :D**

Let's take a respite and enjoy this victory for a brief moment, shall we? :)

Let's take a respite and enjoy this victory for a brief moment, shall we? :)

A moment of reflection... 🙏

Okay, let's get back to business...

Okay, let's get back to business...

So, what about bounds checking on
flexible-array members?

The new ***counted_by*** attribute

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)

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

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

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

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

“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>

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

“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>

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

“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>

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

“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>

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

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

The new *counted_by* attribute

- `__attribute__((__counted_by__(member)))`
- Released in **Clang-18** (LLVM id=76348) (Bill Wendling)
- Released in **GCC-15** (bugzilla id=108896) (Qing Zhao)

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

`__builtin_dynamic_object_size()`

Fortified `memcpy()` and `__builtin_dynamic_object_size()`

__builtin_dynamic_object_size()

Fortified `memcpy()` and `__builtin_dynamic_object_size()`

```
__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);
+   size_t dst_size = __builtin_dynamic_object_size(dst, 1);
+   size_t src_size = __builtin_dynamic_object_size(src, 1);
  ...
}
```

__builtin_dynamic_object_size()

Fortified `memcpy()` and `__builtin_dynamic_object_size()`

```
__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);
+   size_t dst_size = __builtin_dynamic_object_size(dst, 1);
+   size_t src_size = __builtin_dynamic_object_size(src, 1);
  ...
}
```

__builtin_dynamic_object_size()

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

- **__builtin_dynamic_object_size()** replaced **__bos()**
- It gets hints from **__alloc_size__** and from **counted_by**
- Adds **run-time bounds-checking coverage on FAMS.**

__builtin_dynamic_object_size()

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

- **__builtin_dynamic_object_size()** replaced **__bos()**
- It gets hints from **__alloc_size__** and from **counted_by**
- Adds **run-time bounds-checking coverage on FAMS.**
- Greater fortification for **memcpy()**.
- **CONFIG_FORTIFY_SOURCE=y** benefits from all this.

__builtin_dynamic_object_size()

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

- **__builtin_dynamic_object_size()** replaced **__bos()**
- It gets hints from **__alloc_size__** and from **counted_by**
- Adds **run-time bounds-checking coverage on FAMS.**
- Greater fortification for **memcpy()**.
- **CONFIG_FORTIFY_SOURCE=y** benefits from all this.

With *counted_by* & **__bdos(1)**, we gain
bounds-checking on flexible arrays! :D

__builtin_dynamic_object_size()

Fortified `memcpy()` and `__builtin_dynamic_object_size()`

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

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

__builtin_dynamic_object_size()

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

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

    if (__builtin_constant_p(size)) { /* Compile-time */
        if (dst_size < size)
            __write_overflow();
        if (src_size < size)
            __read_overflow2();
    }
    if (dst_size < size || src_size < size) /* Run-time */
        fortify_panic(__func__);
    return __underlying_memcpy(dst, src, size);
}
```

“How to use the new **counted_by** attribute
in C (and Linux)”

blogpost

How to use the *counted_by* attribute

How to use the *counted_by* attribute

Requirements

- **count** must be initialized before first reference to **fam**
- **fam** has at least **count** number of elements

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

How to use the *counted_by* attribute

Requirements

- **count** must be initialized before first reference to **fam**
- **fam** has at least **count** number of elements
- data is the FAM, and datalen the counter

```
struct brcmf_fweh_queue_item {
    u8 ifaddr[ETH_ALEN];
    struct brcmf_event_msg_be emsg;
    u32 datalen;
-   u8 data[];
+   u8 data[] __counted_by(datalen);
};
```

How to use the *counted_by* attribute

Requirements

- **count** must be initialized before first reference to **fam**
 - **fam** has at least **count** number of elements
 - data is the FAM, and datalen the counter
- ```
- event = kzalloc(sizeof(*event) + datalen, gfp);
+ event = kzalloc(struct_size(event, data, datalen), gfp);
 if (!event)
 return;

+ event->datalen = datalen;
...
 memcpy(event->data, data, datalen);
- event->datalen = datalen;
```

# How to use the *counted\_by* attribute

## Requirements

- **count** must be initialized before first reference to **fam**
  - **fam** has at least **count** number of elements
  - **data** is the FAM, and **datalen** the counter
- ```
- event = kzalloc(sizeof(*event) + datalen, gfp);  
+ event = kzalloc(struct_size(event, data, datalen), gfp);  
  if (!event)  
      return;  
  
+ event->datalen = datalen;  
...  
  memcpy(event->data, data, datalen);  
- event->datalen = datalen;
```

How to use the *counted_by* attribute

Requirements

- **count** must be initialized before first reference to **fam**
 - **fam** has at least **count** number of elements
 - **data** is the FAM, and **datalen** the counter
- ```
- event = kzalloc(sizeof(*event) + datalen, gfp);
+ event = kzalloc(struct_size(event, data, datalen), gfp);
 if (!event)
 return;

+ event->datalen = datalen;
...
 memcpy(event->data, data, datalen);
- event->datalen = datalen;
```

# How to use the *counted\_by* attribute

## Requirements

- **count** must be initialized before first reference to **fam**
- **fam** has at least **count** number of elements
- **data** is the FAM, and **datalen** the counter

```
- event = kzalloc(sizeof(*event) + datalen, gfp);
+ event = kzalloc(struct_size(event, data, datalen), gfp);
 if (!event)
 return;

+ event->datalen = datalen;
...
 memcpy(event->data, data, datalen);
- event->datalen = datalen;
```

# How to use the *counted\_by* attribute

## Requirements

- **count** must be initialized before first reference to **fam**
  - **fam** has at least **count** number of elements
  - **data** is the FAM, and **datalen** the counter
- ```
- event = kzalloc(sizeof(*event) + datalen, gfp);  
+ event = kzalloc(struct_size(event, data, datalen), gfp);  
  if (!event)  
      return;  
  
+ event->datalen = datalen;  
...  
  memcpy(event->data, data, datalen);  
- event->datalen = datalen;
```

How to use the *counted_by* attribute

Requirements

- **count** must be initialized before first reference to **fam**
- **fam** has at least **count** number of elements
- **data** is the FAM, and **datalen** the counter

```
- event = kzalloc(sizeof(*event) + datalen, gfp);  
+ event = kzalloc(struct_size(event, data, datalen), gfp);  
  if (!event)  
      return;  
  
+ event->datalen = datalen;  
...  
  memcpy(event->data, data, datalen);  
- event->datalen = datalen;
```

The upcoming `__counted_by_ptr` helper

The upcoming `__counted_by_ptr` helper

- [PATCH 0/3] compiler_types: Introduce `__counted_by_ptr()`

```
struct some_struct {  
    int a, b, c;  
    char *buffer __counted_by_ptr(bytes);  
    short nrBars;  
    struct bar *bars __counted_by_ptr(nrBars);  
    size_t bytes;  
};
```

The upcoming `__counted_by_ptr` helper

- [PATCH 0/3] compiler_types: Introduce `__counted_by_ptr()`
- Supported since GCC ≥ 16
- Supported since Clang ≥ 20

```
struct some_struct {  
    int a, b, c;  
    char *buffer __counted_by_ptr(bytes);  
    short nrBars;  
    struct bar *bars __counted_by_ptr(nrBars);  
    size_t bytes;  
};
```

So far, we've seen the **evolution of array bounds-checking in C**, and in the **Linux kernel** over the past 6 years of **hard work. :)**

-Wflex-array-member-not-at-end (GCC-14)

Bleeding-edge kernel hardening

-Wflex-array-member-not-at-end (GCC-14)

Bleeding-edge kernel hardening

```
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 (GCC-14)

Bleeding-edge kernel hardening

- 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 ↴ ↴ */

    ...
};
```

-Wflex-array-member-not-at-end (GCC-14)

Bleeding-edge kernel hardening

- 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; /* warning! */

    ...
};
```

-Wflex-array-member-not-at-end (GCC-14)

Four different categories of False Positives

- C1: Some **FAMs not used at all.**
 - commit f4b09b29f8b4
- C2: **FAMs never accessed.**
 - commit 5c4250092fad
- C3: **Implicit unions** between FAMs and fixed-size arrays.
 - commit 38aa3f5ac6d2
- C4: The same as case 3 but **on-stack.**
 - commit 34c34c242a1b

Conclusions

Conclusions

Problem:

- **BadVibes-like** bugs.
- **Unintentional** cross-member overflows.

Conclusions

Problem:

- **BadVibes-like** bugs.
- **Unintentional** cross-member overflows.

Solution:

- Update memcpy() to use
__builtin_dynamic_object_size(1)

Conclusions

Problem:

- **Intentional** cross-member overflows –false positives

Conclusions

Problem:

- **Intentional** cross-member overflows –false positives

Solution:

- Use the **struct_group()** family of helpers.
- Fixed tons of false positives.

Conclusions

Problem:

- Trailing array **ambiguity**.
- Lack of bounds-checking on trailing **arrays of fixed size**.

Conclusions

Problem:

- Trailing array **ambiguity**.
- Lack of bounds-checking on trailing **arrays of fixed size**.

Solution:

- Flexible-array **transformations** (**[1]** & **[0]** → **[]**)
- **-fstrict-flex-arrays=3**

Conclusions

Problem:

- Lack of bounds-checking on **flexible arrays**.

Conclusions

Problem:

- Lack of bounds-checking on **flexible arrays**.

Solution:

- Annotate FAMs with **counted_by()**
- `__builtin_dynamic_object_size(1)`
- GCC \geq 15
- Clang \geq 18

Conclusions

Problem:

- Lack of bounds-checking on **ptrs in structs**

Conclusions

Problem:

- Lack of bounds-checking on **ptrs in structs**

Solution:

- Annotate FAMs with **counted_by_ptr()**
- `__builtin_dynamic_object_size(1)`
- GCC \geq 16
- Clang \geq 20
- Work in progress

Conclusions

Problem:

- Flexible arrays **in the middle.**

Conclusions

Problem:

- Flexible arrays **in the middle.**

Solution:

- Enable **-Wflex-array-member-not-at-end**
- Work in progress.

Conclusions

- Clear strategy to enable **-Wflex-array-member-not-at-end** in mainline, soon.
- Build your kernel with **CONFIG_FORTIFY_SOURCE=y** & **CONFIG_UBSAN_BOUNDS=y** (-fsanitize=bounds).

Conclusions

- Clear strategy to enable **-Wflex-array-member-not-at-end** in mainline, soon.
- Build your kernel with **CONFIG_FORTIFY_SOURCE=y** & **CONFIG_UBSAN_BOUNDS=y** (-fsanitize=bounds).
- **Kernel security is being significantly improved. :)**

Thank you, Seoul! 🇰🇷

Gustavo A. R. Silva
gustavoars@kernel.org
fosstodon.org/@gustavoars
<https://embededor.com/blog/>



By @shidokou

Resources

- **Safer flexible arrays for the kernel:**
<https://lwn.net/Articles/908817/>
- **How to use the new counted_by attribute in C (and Linux):**
https://embededor.com/blog/2024/06/18/how-to-use-the-new-counted_by-attribute-in-c-and-linux/
- **GCC features to help harden the kernel:**
<https://lwn.net/Articles/946041/>
- <https://best.openssf.org/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.html>