Gaining bounds-checking on trailing arrays in the Upstream Linux Kernel

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

Supported by
The Linux Foundation & Google

Kernel Recipes 10th edition!!!
Sep 27, 2023
Paris, France

Who am I?

Who am I?

- **Upstream first** – 7 years.
- Upstream Linux Kernel Engineer.
    - Focused on security.

Who am I?

- **Upstream first** – 7 years.
- Upstream Linux Kernel Engineer.
    - Focused on security.

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

Agenda

- **Introduction**
  - Arrays in C and The Land of Possibilities.
  - Trailing arrays as Variable Length Objects (VLOs).
  - Flexible arrays and Flexible structures.
- **Gaining bounds-checking on trailing arrays**
  - Ambiguous flexible-arrays declarations
  - Problems and flexible-array transformations.
  - Fortified memcpy() and trailing arrays.
  - The case of UAPI.
- **Conclusions**

Arrays in C and The Land of Possibilities

```
int happy_array[10];
```

Arrays in C and The Land of Possibilities

- Contiguously allocated objects of the same element type.
- We can iterate over it through indexes from 0 to N - 1, where N is the maximum number of elements in the array.

```
int happy_array[10];
```

indexes: [0-9]

Arrays in C and The Land of Possibilities

- Contiguously allocated objects of the same element type.
- We can iterate over it through indexes from 0 to N - 1, where N is the maximum number of elements in the array.
- However, C doesn’t enforce array’s boundaries.
- It’s up to the developers to enforce them.

```
int happy_array[10];
```

indexes: [0-9]

Arrays in C and The Land of Possibilities

- Contiguously allocated objects of the same element type.
- We can iterate over it through indexes from 0 to N - 1, where N is the maximum number of elements in the array.
- However, C doesn’t enforce array’s boundaries.
- It’s up to the developers to enforce them.
- Otherwise, you arrive in The Land of Possibilities (a.k.a. UB).

```
int happy_array[10];
```

indexes: [0-9]

Arrays in C and The Land of Possibilities

miserable_array[-1]

Trailing arrays

Trailing arrays in the kernel

- Arrays declared at the end of a structure.

```
struct trailing {
        ...
        some members;
        int happy_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 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[];
};
```

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.

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 fake flexible arrays

- Three different ways to declare a Variable Length Object (VLO).

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*** 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*** false positives.

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

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

Problems with 1-element arrays

- ***-Warray-bounds*** false positives.

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

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

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

GNU extension: 0-length arrays

- Not part of the C standard.
- They don’t contribute to the size of the flex 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;
```

The Land of Possibilities

Undefined Behavior

The Land of Possibilities

Undefined Behavior – The bug

- e48f129c2f20 ("[SCSI] cxgb3i: convert cdev->l2opt to use…")

```
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;
};
```

The Land of Possibilities

Undefined Behavior – The bug

- e48f129c2f20 ("[SCSI] cxgb3i: convert cdev->l2opt to use…")
- Compilers cannot detect dangerous code like this.

```
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;
};
```

The Land of Possibilities

Undefined Behavior – The fix

- 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[];
};
```

The Land of Possibilities

Undefined Behavior – The fix

- 76497732932f ("cxgb3/l2t: Fix undefined behavior")
- **Kick-off** of flexible array transformations in the **KSPP**.

```
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[];
};
```

The Land of Possibilities

Undefined Behavior – The fix

- 76497732932f ("cxgb3/l2t: Fix undefined behavior")
- **Kick-off** of flexible array transformations in the KSPP.
- Bug introduced in **2011**. 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[];
};
```

The Land of Possibilities

Undefined Behavior – The bug

- f5823fe6897c ("qed: Add ll2 option to limit the number of bds per packet")

```
#define ETH_TX_MAX_BDS_PER_NON_LSO_PACKET    18

struct qed_ll2_tx_packet {
        ...
+       /* Flexible Array of bds_set determined by max_bds_per_packet */
        struct {
                struct core_tx_bd *txq_bd;
                dma_addr_t tx_frag;
                u16 frag_len;
-       } bds_set[ETH_TX_MAX_BDS_PER_NON_LSO_PACKET];
+       } bds_set[1];
 };
```

The Land of Possibilities

Undefined Behavior – The bug

- f5823fe6897c ("qed: Add ll2 option to limit the number of bds per packet")
- **Fake** flex-array transformation.

```
#define ETH_TX_MAX_BDS_PER_NON_LSO_PACKET    18

struct qed_ll2_tx_packet {
        ...
+       /* Flexible Array of bds_set determined by max_bds_per_packet */
        struct {
                struct core_tx_bd *txq_bd;
                dma_addr_t tx_frag;
                u16 frag_len;
-       } bds_set[ETH_TX_MAX_BDS_PER_NON_LSO_PACKET];
+       } bds_set[1];
 };
```

The Land of Possibilities

Undefined Behavior – The bug

- f5823fe6897c ("qed: Add ll2 option to limit the number of bds per packet")
- Now there is a 1-element array nested in the middle of struct **qed_ll2_tx_queue**

```
struct qed_ll2_tx_queue {
        ...
-       struct qed_ll2_tx_packet *descq_array;
+       void *descq_mem; /* memory for variable sized qed_ll2_tx_packet*/
        struct qed_ll2_tx_packet *cur_send_packet;
        struct qed_ll2_tx_packet cur_completing_packet;
        ...
        u16 cur_completing_frag_num;
        bool b_completing_packet;
};
```

The Land of Possibilities

Undefined Behavior – The fix

- a93b6a2b9f46 ("qed/red_ll2: Replace one-element array with flexible … ")

```
struct qed_ll2_tx_packet {
        struct core_tx_bd *txq_bd;
        dma_addr_t tx_frag;
        u16 frag_len;
-   } bds_set[1];
+   } bds_set[];
};
```

```
struct qed_ll2_tx_queue {
    ...
-   struct qed_ll2_tx_packet cur_completing_packet;
    ...
    u16 cur_completing_frag_num;
    bool b_completing_packet;
    ...
+   struct qed_ll2_tx_packet cur_completing_packet;
};
```

The Land of Possibilities

Undefined Behavior – The fix

- a93b6a2b9f46 ("qed/red_ll2: Replace one-element array with flexible … ")
- Bug introduced in **2017**. Fixed in **2020**.

```
struct qed_ll2_tx_packet {
        struct core_tx_bd *txq_bd;
        dma_addr_t tx_frag;
        u16 frag_len;
-   } bds_set[1];
+   } bds_set[];
};
```

```
struct qed_ll2_tx_queue {
    ...
-   struct qed_ll2_tx_packet cur_completing_packet;
    ...
    u16 cur_completing_frag_num;
    bool b_completing_packet;
    ...
+   struct qed_ll2_tx_packet cur_completing_packet;
};
```

Then something happened on **Saturday**…

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

```
struct qed_ll2_info {

    ...

    struct qed_ll2_tx_queue tx_queue;
    struct qed_ll2_cbs cbs;
};
```

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

```
struct qed_ll2_info {
    ...
    struct qed_ll2_tx_queue tx_queue;
    struct qed_ll2_cbs cbs;
};
```

```
struct qed_ll2_info {
    ...
    struct qed_ll2_tx_queue {
        ...
        struct qed_ll2_tx_packet {
            ...
            struct {
                struct core_tx_bd *txq_bd;
                dma_addr_t tx_frag;
                u16 frag_len;
            } bds_set[];
        };
    };
    struct qed_ll2_cbs cbs;
};
```

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

Undefined Behavior – The bug

```
struct qed_ll2_info {
    ...
    struct qed_ll2_tx_queue tx_queue;
    struct qed_ll2_cbs cbs;
};
```

```
struct qed_ll2_info {
    ...
    struct qed_ll2_tx_queue {
        ...
        struct qed_ll2_tx_packet {
            ...
            struct {
                struct core_tx_bd *txq_bd;
                dma_addr_t tx_frag;
                u16 frag_len;
            } bds_set[];
        };
    };
    struct qed_ll2_cbs cbs;
};
```

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

Undefined Behavior – The bug

- Structure full of function pointers.

```
struct qed_ll2_cbs {
        qed_ll2_complete_rx_packet_cb rx_comp_cb;
        qed_ll2_release_rx_packet_cb rx_release_cb;
        qed_ll2_complete_tx_packet_cb tx_comp_cb;
        qed_ll2_release_tx_packet_cb tx_release_cb;
        qed_ll2_slowpath_cb slowpath_cb;
        void *cookie;
};
```

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

Undefined Behavior – The fix

- https://lore.kernel.org/linux-hardening/ZQ+Nz8DfPg56pIzr@work/

```
struct qed_ll2_info {
    ...
    struct qed_ll2_tx_queue tx_queue;
    struct qed_ll2_cbs cbs;
};
```

```
struct qed_ll2_info {
    ...
+   struct qed_ll2_cbs cbs;
    struct qed_ll2_rx_queue rx_queue;
    struct qed_ll2_tx_queue tx_queue;
-   struct qed_ll2_cbs cbs;
 };
```

$ git grep -nwW '**struct**\s**qed_ll2_tx_queue**'

Undefined Behavior – The fix

- https://lore.kernel.org/linux-hardening/ZQ+Nz8DfPg56pIzr@work/
- Bug introduced in **2017**. Fixed in **2023**.
- Will appear in mainline, soon.

```
struct qed_ll2_info {
   ...
+  struct qed_ll2_cbs cbs;
   struct qed_ll2_rx_queue rx_queue;
   struct qed_ll2_tx_queue tx_queue;
-  struct qed_ll2_cbs cbs;
 };
```

“Nice find! Was this located with **-Wflex-array-member-not-at-end** ?”

– Kees Cook

https://lore.kernel.org/linux-hardening/94131E7C-BC22-423B-8B05-234BB2E09EFD@kernel.org/

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

GCC **new compiler option** (coming soon in GCC 14).

- https://gcc.gnu.org/pipermail/gcc-patches/2023-March/614794.html
- https://gcc.gnu.org/pipermail/gcc-patches/2023-March/614793.html
- https://gcc.gnu.org/pipermail/gcc-patches/2023-March/614790.html

“A structure or a union with a C99 flexible array member is the middle field of another structure, for example:

```
struct flex  { int length; char data[]; };
struct mid_flex { int m; struct flex flex_data; int n; };
```

In the above, 'mid_flex.flex_data.data[]' has undefined behavior. Compilers do not handle such case consistently, Any code relying on such case should be modified to ensure that flexible array members only end up at the ends of structures.

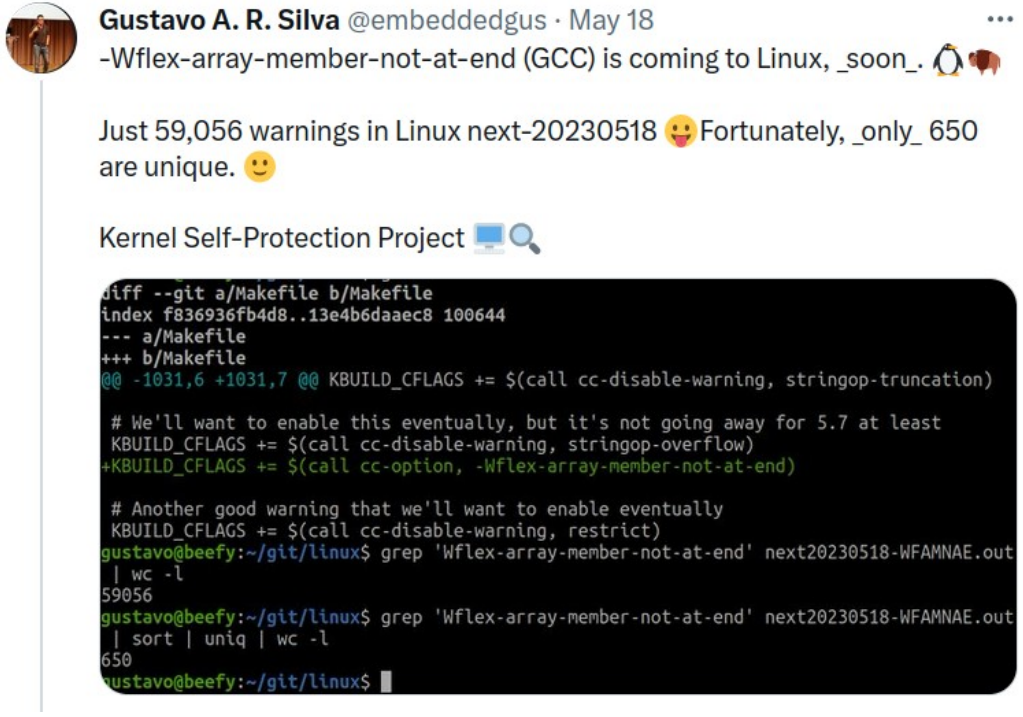
Please use warning option '-Wflex-array-member-not-at-end' to identify all such cases in the source code and modify them. This warning will be on by default starting from GCC 14.”

-Qing Zhao

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

GCC new compiler option (coming soon in GCC 14)

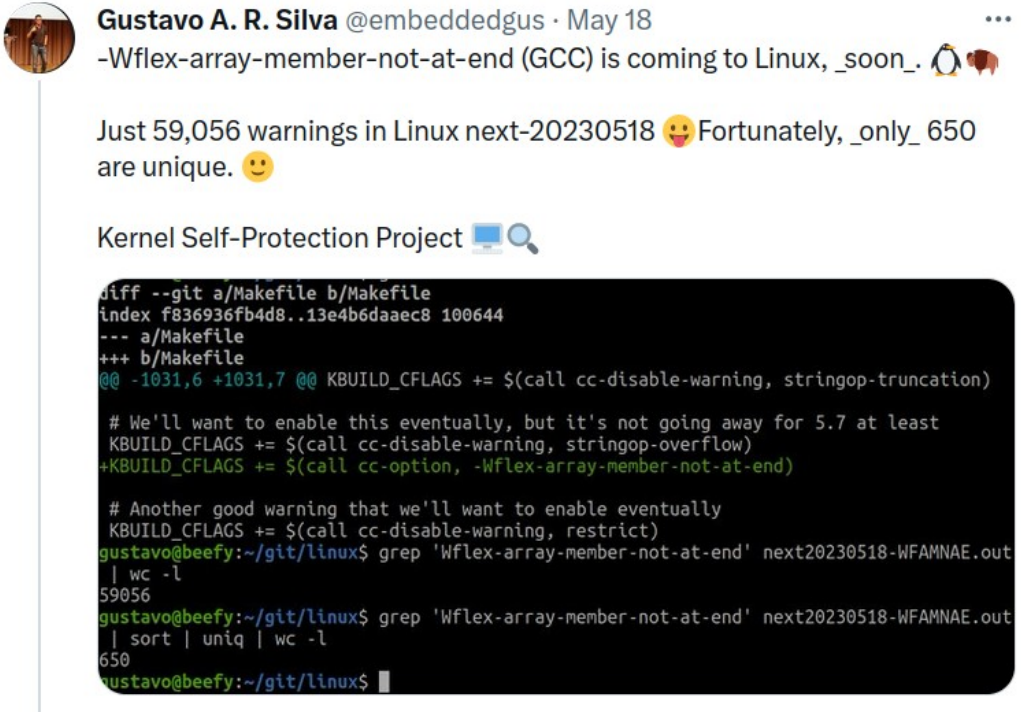
- **59,056 warnings** in Linux next-20230518.



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

GCC new compiler option (coming soon in GCC 14)

- **59,056 warnings** in Linux next-20230518.
- Fortunately, only **650 are unique**.



So I went and took a look at my build logs from that time...

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

GCC new compiler option (coming soon in GCC 14)

- **It works!**

```
drivers/net/ethernet/qlogic/qed/qed_ll2.h:

100 struct qed_ll2_info {
...
114         struct qed_ll2_tx_queue tx_queue;
115         struct qed_ll2_cbs cbs;
116 };

In file included from drivers/net/ethernet/qlogic/qed/qed_dev.c:33:
drivers/net/ethernet/qlogic/qed/qed_ll2.h:114:33: warning: structure
containing a flexible array member is not at the end of another structure
[-Wflex-array-member-not-at-end]
  114 |         struct qed_ll2_tx_queue tx_queue;
      |                                 ^~~~~~~~
```

Problems with ambiguous flexible-array variants

- The Tale of **sizeof()** & the Three Trailing Arrays.

Problems with ambiguous flexible-array variants

- The Tale of **sizeof()** & the Three Trailing Arrays.

```
sizeof(flex_struct->one_element_array) == size-of-element-type
```

Problems with ambiguous flexible-array variants

- The Tale of **sizeof()** & the Three Trailing Arrays.

```
sizeof(flex_struct->one_element_array) == size-of-element-type

sizeof(flex_struct->zero_length_array) == 0
```

Problems with ambiguous flexible-array variants

- The Tale of **sizeof()** & the Three Trailing Arrays.

```
sizeof(flex_struct->one_element_array) == size-of-element-type

sizeof(flex_struct->zero_length_array) == 0

sizeof(flex_struct->flex_array_member) == ? /* Build error */
```

Problems with ambiguous flexible-array variants

- The Tale of **sizeof()** & the Three Trailing Arrays.
  - `sizeof()` returns **different results**.
  - And that's another source of **problems**.
  - Found multiple issues in the kernel.

```
sizeof(flex_struct->one_element_array) == size-of-element-type

sizeof(flex_struct->zero_length_array) == 0

sizeof(flex_struct->flex_array_member) == ? /* Build error */
```

Problems with ambiguous flexible-array variants

Ambiguity is the enemy.

Gaining bounds-checking on trailing arrays

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- Common use of **memcpy()** and flex arrays.

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

...

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size() was** used to determine the size of both **source** and **destination**.
- 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();
        }
        ...
}
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size() was** used to determine the size of both **source** and **destination**.
- 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();
        }
        ...
}
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- Common use of **memcpy()** and flex arrays.

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

...

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

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

__builtin_object_size(p->flex_array, 1) == -1  /* flex-array size? */
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** if cannot determine the size of the object.

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** if cannot determine the size of the object.

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

The size of a flexible-array member cannot be determined -- **it's an object of incomplete type**.

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** if cannot determine the size of the object.
  - The size of a flexible-array member cannot be determined (**it's an object of incomplete type**).

OK; but, what about **fake** flexible arrays?

Those do have a size.

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

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

Gaining bounds-checking on trailing arrays

- Hardening **memcpy()** and flexible-array transformations
  - **__builtin_object_size()** and flexible arrays

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

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

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

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

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible 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

- Hardening **memcpy()** and flexible-array transformations
  - **__builtin_object_size()** and flexible 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
```

It’s not able to reason about the size of the **fake** flex arrays either. Returns **-1** for **all three** cases.

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** for **all** three cases.
  - It **doesn't know** the size of the **fake** flex arrays either.

```
__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

sizeof(flex_struct->one_element_array) == size-of-element-type
sizeof(flex_struct->zero_length_array) == 0
sizeof(flex_struct->flex_array_member) == ? /* Error */
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** for **all** three cases.
  - It **doesn't know** the size of the **fake** flex arrays either.
  - A bit **confusing**, isn't it?

```
__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

sizeof(flex_struct->one_element_array) == size-of-element-type
sizeof(flex_struct->zero_length_array) == 0
sizeof(flex_struct->flex_array_member) == ? /* Error */
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** for **all** three cases.
  - It **doesn't know** the size of the **fake** flex arrays either.
  - A bit **confusing**, isn't it?

```
__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

sizeof(flex_struct->one_element_array) == size-of-element-type
sizeof(flex_struct->zero_length_array) == 0
sizeof(flex_struct->flex_array_member) == ? /* Error */

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays
  - Returns **-1** for **all** three cases.
  - It **doesn't know** the size of the **fake** flex arrays either.
  - A bit **confusing**, isn't it?

```
__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

sizeof(flex_struct->one_element_array) == size-of-element-type
sizeof(flex_struct->zero_length_array) == 0
sizeof(flex_struct->flex_array_member) == ? /* Error */

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

Gaining bounds-checking on trailing arrays

- Hardening **memcpy()** and flexible-array transformations
  - **__builtin_object_size()** and flexible arrays

What is going on?!

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

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

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

A case for:

“Go fix the compiler!”

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible arrays

But why, exactly?

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

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

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

```
/*
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
        unsigned char   sa_len;         /* total length */
        sa_family_t     sa_family;      /* address family */
        char            sa_data[14];    /* actually longer; address value */
};
#define SOCK_MAXADDRLEN    255         /* longest possible addresses */
```

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **__builtin_object_size()** and flexible 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**.”

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

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

So, what do we do?

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **Kernel**: Make flexible-array declarations **unambiguous**.
  - Get rid of **fake** flexible arrays.
  - Only C99 **flexible-array members** should be used as flexible arrays.

Gaining bounds-checking on trailing arrays

Hardening **memcpy()** and flexible-array transformations

- **Kernel**: Make flexible-array declarations **unambiguous**.
  - Get rid of **fake** flexible arrays.
  - Only C99 **flexible-array members** should be used as flexible arrays.
- **Compiler**: Fix it.
  - Fix **__builtin_object_size()**
  - Add new option **-fstrict-flex-arrays**[=n]

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
```

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)
  - 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, that issue is now resolved. :)

Gaining bounds-checking on trailing arrays

-fstrict-flex-arrays[=n] – Supported 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
```

Gaining bounds-checking on trailing arrays

-fstrict-flex-arrays[=n] – Supported 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
```

Now **ALL** trailing arrays of **fixed-size gain** bounds-checking. :D

Gaining bounds-checking on trailing arrays

-fstrict-flex-arrays[=n] – Supported 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
```

Now **ALL** trailing arrays of **fixed-size gain** bounds-checking. :D

This is what we **want**!

Gaining bounds-checking on trailing arrays

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

When will we have nice things?

Gaining bounds-checking on trailing arrays

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

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

Gaining bounds-checking on trailing arrays

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

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

Gaining bounds-checking on trailing arrays

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

- Globally enabled in **Linux 6.5**. Yeeiii!! Mega yeeiii!!
- CONFIG_UBSAN_BOUNDS and CONFIG_FORTIFY_SOURCE benefit from this.
- Only C99 flexible-array members are considered to be dynamically sized.

Gaining bounds-checking on trailing arrays

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

- Globally enabled in **Linux 6.5**. Yeeiii!! Mega yeeiii!!
- CONFIG_UBSAN_BOUNDS and CONFIG_FORTIFY_SOURCE benefit from this.
- Only C99 flexible-array members are considered to be dynamically sized.
- **Therefore, we've gained bounds-checking on trailing arrays of fixed-size.**

Gaining bounds-checking on trailing arrays

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

Gaining bounds-checking on trailing arrays

We need a new attribute

Gaining bounds-checking on trailing arrays

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)));
};
```

Gaining bounds-checking on trailing arrays

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-14** and **Clang-18**

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

Gaining bounds-checking on trailing arrays

We need a new attribute

- How about __attribute__((__**counted_by**__(member))) ?
- Coming soon in **GCC-14** and **Clang-18**

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

“Hey! but you said that memcpy() **WAS** internally using *__builtin_object_size()*?”

Gaining bounds-checking on trailing arrays

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

Gaining bounds-checking on trailing arrays

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().

The case of UAPI

The case of UAPI

One-element arrays in UAPI – First attempts.

- Duplicate the original struct within a union.
- Flexible-array will be used by **kernel-space**.
- One-element array will be used by **user-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[];
+               };
+       };
 };
```

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);
        };
};
```

The case of UAPI

One-element arrays in UAPI – Better code.

- Just use the **__DECLARE_FLEX_ARRAY**() helper in a union.
- The bad news is that the **sizeof**(flex_struct) will remain the same.

```
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);
        };
};
```

Conclusions

Conclusions

- **-fstrict-flex-arrays=3** enabled in Linux 6.5
- **__counted_by()** attribute is just around the corner. :D
- **__builtin_dynamic_object_size()** increased bounds-checking coverage.
- FORTIFY_SOURCE and UBSAN bounds-checking better every time.
- Vulnerabilities discovered over the last years could’ve been prevented with the most recent **memcpy()** and **FORTIFY_SOURCE** updates.
- We’ve been finding and fixing bugs in both **kernel-space** and **user-space**.
- **The security of the kernel is being significantly improved. :)**

Conclusions

- Next: Replace DECLARE_FLEX_ARRAY() with **DECLARE_BOUNDED_ARRAY()**:

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

struct ip_msfilter {
        ...
        __u32           imsf_numsrc;
        union {
                __be32          imsf_slist[1];
                __DECLARE_BOUNDED_ARRAY(__be32, imsf_slist_flex, imsf_numsrc);
        };
};
```

Conclusions

- Next: *__counted_by()* **on pointers** should be possible.

```
struct foo {
        ...
        unsigned char   items;
        ...
        int             *data __counted_by(items);
        ...
};
```

Conclusions

- Next: *__counted_by()* **on pointers** should be possible.

```
struct foo {
        ...
        unsigned char   items;
        ...
        int             *data __counted_by(items);
        ...
};
```

- Next: **-Wflex-array-member-not-at-end** has proved to catch bugs.

Thank you! :)

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