Flexible-Array Transformations and Array-bounds checking

Gustavo A. R. Silva
gustavoars@kernel.org
@embeddedgus

Supported by The Linux Foundation & Google

Linux Security Summit Europe
Sep 15, 2022
Dublin, Ireland

Who am I?

- Embedded Systems.
- RTOS & Embedded **Linux**.

Who am I?

- Embedded Systems.
- RTOS & Embedded **Linux**.
- **Upstream first** – 6 years.
- Kernel Engineer & Maintainer.
- Kernel Self-Protection Project (**KSPP**).
- **GOSST -** Linux kernel division.

Who am I?

- Embedded Systems.
- RTOS & Embedded **Linux**.
- **Upstream first** – 6 years.
- Kernel Engineer & Maintainer.
- Kernel Self-Protection Project (**KSPP**).
- **GOSST** - Linux kernel division.
- Volunteer at @kidsoncomputers

Agenda

- **Introduction**
  - Arrays in C and The Land of Possibilities.
  - Trailing arrays as Variable Length Objects (VLOs).
  - Flexible arrays and Flexible structures.
- **Flexible-Array Transformations & Array-bounds checking**
  - Ambiguous flexible-array declarations and problems.
  - Gaining bounds-checking on trailing arrays.
  - The case of UAPI.
  - Current status.
- **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];
};
```

Trailing arrays as Variable Length Objects (VLOs)

- Usually **blobs of** raw **data** (of any type).
- Space is allocated at run-time.
- Their contents are usually **described through a header**.
- drivers/firmware/google/vpd.c:30:

```
struct vpd_cbmem {
        u32 magic;
        u32 version;
        u32 ro_size;
        u32 rw_size;
        u8  blob[];
};
```

Flexible arrays & flexible structures

- Flexible array
  - **Trailing** array as **VLO**.
  - Total 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-array variants

Flexible-array variants

Ambiguous flex-array declarations.

- **Fake** flexible arrays.
  - One-element arrays.
  - Zero-length arrays.
- **True** flexible arrays.
  - “Modern” C99 flexible-array member.

Flexible-array variants

Ambiguous flex-array declarations.

- **Fake** flexible arrays.
  - One-element arrays (**buggy hack**).
  - Zero-length arrays (**GNU extension**).
- **True** flexible arrays.
  - “Modern” C99 flexible-array member.

Flexible-array variants

Ambiguous flex-array declarations.

- **Fake** flexible arrays.
  - **One-element** arrays (**buggy hack**).
  - Always “contributes” with **sizeof-one-element** to the size of the enclosing structure.
  - Potential source of **off-by-one bugs**.

```
struct ancient {
        ...
        size_t count;
        struct foo anxious_array[1];
} *p;
```

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

Flexible-array variants

Ambiguous flex-array declarations.

- **Fake** flexible arrays.
  - Need to audit every use of **sizeof(*p)**
  - Is **struct ancient** being used inside another struct?
  - Need to audit every use of **sizeof(struct foo)**
  - Does the original code contains **OBO** issues?

```
struct ancient {
        ...
        size_t count;
        struct foo anxious_array[1];
} *p;
```

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

Flexible-array variants

Ambiguous flex-array declarations.

- **Fake** flexible arrays.
  - **Zero-length** arrays (**GNU extension**).
  - They **don’t** contribute to the size of the flex struct.
  - **Slightly** less buggy, but still...

```
struct old {
        …
        size_t count;
        struct foo unhappy_array[0];
} *p;
```

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

Flexible-array variants

Ambiguous flex-array declarations.

- **True** flexible arrays.
  - **Flexible-array member** (C99).
  - The last member of an otherwise **non-empty** structure.
  - The compiler enforces this (unlike in the case of [1] & [0])

```
struct modern {
        ...
        size_t count;
        struct foo happy_array[];
} *p;
```

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

Problems with ambiguous flexible-array variants

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

Problems with ambiguous flexible-array variants

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

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

Problems with ambiguous flexible-array variants

sizeof() & The Tale of 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

sizeof() & The Tale of 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) == ? /* Error */
```

Problems with ambiguous flexible-array variants

sizeof() & The Tale of 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) == ? /* Error */
```

Problems with ambiguous flexible-array variants

The Land of Possibilities. ;-)

```
@@ -75,8 +75,8 @@ struct l2t_data {
         struct l2t_entry *rover;
         atomic_t nfree;                 /* n
         rwlock_t lock;
-        struct l2t_entry l2tab[0];
         struct rcu_head rcu_head;
+        struct l2t_entry l2tab[];
 };
```

Problems with ambiguous flexible-array variants

The Land of Possibilities. :-)

- First flexible array transformation in the **KSPP**.
- 76497732932f ("cxgb3/l2t: Fix undefined behaviour")

```
@@ -75,8 +75,8 @@ struct l2t_data {
         struct l2t_entry *rover;
         atomic_t nfree;                 /* n
         rwlock_t lock;
-        struct l2t_entry l2tab[0];
         struct rcu_head rcu_head;
+        struct l2t_entry l2tab[];
 };
```

Problems with ambiguous flexible-array variants

The Land of Possibilities. :-)

- First flexible array transformation in the **KSPP**.
- 76497732932f ("cxgb3/l2t: Fix undefined behaviour")
- Bug introduced in **2011**. Fixed in **2019**.

```
@@ -75,8 +75,8 @@ struct l2t_data {
         struct l2t_entry *rover;
         atomic_t nfree;                 /* n
         rwlock_t lock;
-        struct l2t_entry l2tab[0];
         struct rcu_head rcu_head;
+        struct l2t_entry l2tab[];
 };
```

Problems with ambiguous flexible-array variants

Ambiguity is the enemy.

Gaining bounds-checking on trailing arrays

-Warray-bounds and flexible-array transformations

Gaining bounds-checking on trailing arrays

-Warray-bounds and flexible-array transformations

- Directly indexing flexible arrays is not uncommon.
- We had to fix multiple out-of-bounds issues in **fake flexible arrays** ([0] and [1] trailing arrays).
- Of course, almost all of them were **false positives**.
- However, they needed to be fixed before enabling -Warray-bounds.

Gaining bounds-checking on trailing arrays

-Warray-bounds and flexible-array transformations

- Some examples:
  - This one very simple and straight-forward.
  - Commit c1e4726f465440

```
fs/hpfs/dir.c: In function `hpfs_readdir':
fs/hpfs/dir.c:163:41: warning: array subscript 1 is above
array bounds of `u8[1]' {aka `unsigned char[1]'} [-Warray-bounds]
  163 |         || de ->name[0] != 1 || de->name[1] != 1))
      |                                 ~~~~~~~~^~~
```

```
@@ -356,7 +356,8 @@ struct hpfs_dirent {
    u8 no_of_acls;                    /* number of ACL's (low
    u8 ix;                            /* code page index (of
                                         struct code_page_data
-   u8 namelen, name[1];              /* file name */
+   u8 namelen;                       /* file name length */
+   u8 name[];                        /* file name */
    /* dnode_secno down;   btree down pointer, if present,
                           follows name on next word boundary, o
                           precedes next dirent, which is on a w
```

Gaining bounds-checking on trailing arrays

-Warray-bounds and flexible-array transformations

- Some examples:
  - Others a bit more elaborate.
  - Commit 39107e8577ad

```
drivers/scsi/aacraid/aachba.c:4011:28: warning: array subscript 1 is
above array bounds of ‘struct sge_ieee1212[1]’ [-Warray-bounds]
 4011 |   for (j = 0; j < rio2->sge[i].length / (pages * PAGE_SIZE); ++j) {
      |                   ~~~~~~~~~~^~~
drivers/scsi/aacraid/aachba.c:4012:24: warning: array subscript 1 is
above array bounds of ‘struct sge_ieee1212[1]’ [-Warray-bounds]
 4012 |    addr_low = rio2->sge[i].addrLow + j * pages * PAGE_SIZE;
      |               ~~~~~~~~~~^~~
```

Gaining bounds-checking on trailing arrays

-Warray-bounds and flexible-array transformations

- Some examples:
  - Others a bit more elaborate.
  - Commit 39107e8577ad

```
-               fibsize = sizeof(struct aac_raw_io2) +
-                       ((le32_to_cpu(writecmd2->sgeCnt)-1) * sizeof(struct sge_ieee1212));
+               fibsize = struct_size(writecmd2, sge,
+                                     le32_to_cpu(writecmd2->sgeCnt));
        } else {
                struct aac_raw_io *writecmd;
                writecmd = (struct aac_raw_io *) fib_data(fib);
@@ -3998,7 +3998,7 @@ static int aac_convert_sgraw2(struct aac_raw_io2 *rio2, int pages, int
        if (aac_convert_sgl == 0)
                return 0;

-       sge = kmalloc_array(nseg_new, sizeof(struct sge_ieee1212), GFP_ATOMIC);
+       sge = kmalloc_array(nseg_new, sizeof(*sge), GFP_ATOMIC);
        if (sge == NULL)
                return -ENOMEM;


diff --git a/drivers/scsi/aacraid/aacraid.h b/drivers/scsi/aacraid/aacraid.h
index e3e4ecbea726e..3733df77bc65d 100644
--- a/drivers/scsi/aacraid/aacraid.h
+++ b/drivers/scsi/aacraid/aacraid.h
@@ -1929,7 +1929,7 @@ struct aac_raw_io2 {
        u8              bpComplete;     /* reserved for F/W use */
        u8              sgeFirstIndex;  /* reserved for F/W use */
        u8              unused[4];
-       struct sge_ieee1212     sge[1];
+       struct sge_ieee1212     sge[];
 };
```

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

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

```
1  __FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
2  {
3          size_t dst_size = __builtin_object_size(dst, 1);
4          size_t src_size = __builtin_object_size(src, 1);
5
6          if (__builtin_constant_p(size)) {       /* Compile-time */
7                  if (dst_size < size)
8                          __write_overflow();
9                  if (src_size < size)
10                         __read_overflow2();
11         }
12         if (dst_size < size || src_size < size)
13                 fortify_panic(__func__);        /* Run-time */
14         return __underlying_memcpy(dst, src, size);
15 }
```

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

```
__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.
  - 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
  - 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
```

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) == -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

memcpy() is not currently 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
  - Returns **-1** for **all trailing arrays**.
  - Definitely need to **fix the compiler**. :-/

```
__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
  - Returns **-1** for **all trailing arrays**.
  - Definitely need to **fix the compiler**. :-/
  - **sizeof()** is the only sane one. :-)

```
__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

Wait. 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

- **__builtin_object_size()** and flex arrays. **What do we do?**

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1); /* == SIZE_MAX (-1) */
        size_t src_size = __builtin_object_size(src, 1); /* == SIZE_MAX (-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)
                fortify_panic(__func__);        /* Run-time */
        return __underlying_memcpy(dst, src, size);
}
```

Gaining bounds-checking on trailing arrays

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

- **__builtin_object_size()** and flex arrays. **What do we do?**
  - Make flexible-array declarations **unambiguous**.
  - **Fix the compiler**: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=101836

```
__FORTIFY_INLINE void *memcpy(void *dst, const void *src, size_t size)
{
        size_t dst_size = __builtin_object_size(dst, 1); /* == SIZE_MAX (-1) */
        size_t src_size = __builtin_object_size(src, 1); /* == SIZE_MAX (-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)
                fortify_panic(__func__);        /* Run-time */
        return __underlying_memcpy(dst, src, size);
}
```

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.

- -fsfa**=0** → All trailing arrays are treated as flex arrays.
  - __bos(flex_struct→**any_trailing_array**, 1) == -1

Gaining bounds-checking on trailing arrays

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

- -fsfa**=0** → All trailing arrays are treated as flex arrays.
  - __bos(flex_struct→**any_trailing_array**, 1) == -1
- -fsfa**=1** → Only [1], [0] and [ ] are treated as flex arrays.
  - __bos(flex_struct→**one_element_array**, 1) == -1
  - __bos(flex_struct→**zero_length_array**, 1) == -1
  - __bos(flex_struct→**flex_array_member**, 1) == -1

Gaining bounds-checking on trailing arrays

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

- -fsfa**=0** → All trailing arrays are treated as flex arrays.
  - __bos(flex_struct→**any_trailing_array**, 1) == -1
- -fsfa**=1** → Only [1], [0] and [ ] are treated as flex arrays.
  - __bos(flex_struct→**one_element_array**, 1) == -1
  - __bos(flex_struct→**zero_length_array**, 1) == -1
  - __bos(flex_struct→**flex_array_member**, 1) == -1
- -fsfa**=2** → Only [0] and [ ] are treated as flex arrays.
  - __bos(flex_struct→**zero_length_array**, 1) == -1
  - __bos(flex_struct→**flex_array_member**, 1) == -1

Gaining bounds-checking on trailing arrays

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

- -fsfa**=0** → All trailing arrays are treated as flex arrays.
  - __bos(flex_struct→**any_trailing_array**, 1) == -1
- -fsfa**=1** → Only [1], [0] and [ ] are treated as flex arrays.
  - __bos(flex_struct→**one_element_array**, 1) == -1
  - __bos(flex_struct→**zero_length_array**, 1) == -1
  - __bos(flex_struct→**flex_array_member**, 1) == -1
- -fsfa**=2** → Only [0] and [ ] are treated as flex arrays.
  - __bos(flex_struct→**zero_length_array**, 1) == -1
  - __bos(flex_struct→**flex_array_member**, 1) == -1
- -fsfa**=3** → Only [ ] is treated as flex array. **(GCC only).**
  - __bos(flex_struct→**flex_array_member**, 1) == -1

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**

- Need to finish transforming **ALL fake** flexible arrays into **flexible-array members**.
- Need to enable **-fstrict-flex-arrays=3**
- Then memcpy() will be finally able check for **out-of-bounds** on **trailing arrays** and **ALL** arrays of **fixed size**.

Gaining bounds-checking on trailing arrays

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

- Need to finish transforming **ALL fake** flexible arrays into **flexible-array members**.
- Need to enable **-fstrict-flex-arrays=3**
- Then memcpy() will be finally able check for **out-of-bounds** on **trailing arrays** and **ALL** arrays of **fixed size**. Yeeeii!!! :-)

Gaining bounds-checking on trailing arrays

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

OK. Now we know how to gain bounds-checking on trailing arrays of **fixed size**. :)

Gaining bounds-checking on trailing arrays

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

And what about bounds-checking on flexible-array members?

Gaining bounds-checking on trailing arrays

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

- We need a new attribute.
- __attribute__((__**element_count**__(member))) ?

```
struct bounded_flex_struct {
                ...
                size_t elements;
                struct foo flex_array[]
                  __attribute__((__element_count__(elements)));
};
```

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);
        };
};
```

Flexible array transformations in the Linux kernel

Current status

- **Zero-length** arrays mostly transformed (including **UAPI**).
- However, we cannot prevent new ones from being introduced. Please, **don’t** introduce them. :)
- **One-element** arrays are still in progress.
- **Auditing** them demand a lot more work and **time**.
- Need to make sure there are **no** important **differences** between executables (before and after changes).
- objdump, Ghidra, BinDiff and custom **diffing** tools to the rescue.

Flexible arrays transformations in the Linux kernel

Conclusions

- We need to remove problematic ambiguity from the kernel.
- Flexible-array transformations together with **-fstrict-flex-arrays=3** are an important step forward.
- The security of the kernel can be significantly improved.
- Vulnerabilities discovered over the last years could’ve been prevented with the most recent **memcpy()** and **FORTIFY_SOURCE** updates.
- We have a clear vision about how to gain bounds-checking on **ALL** trailing arrays, fixed and flexible.

Thank you! :)

Gustavo A. R. Silva
gustavoars@kernel.org
@embeddedgus