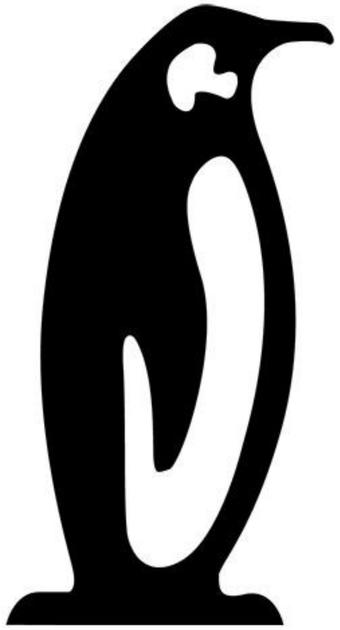


Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

GCC's -fanalyzer and the Linux kernel

David Malcolm <dmalcolm@redhat.com>

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

Outline

- Overview of GCC’s -fanalyzer
- My attempts to use it on the kernel

GCC’s -fanalyzer

- Added by me in GCC 10 (do not use)
- Rewrote heavily in GCC 11
- Rewritten further in GCC 12
- Further work for upcoming GCC 13
- https://gcc.gnu.org/wiki/StaticAnalyzer

GCC’s -fanalyzer

- GCC 10: 15 new warnings
- GCC 11: 7 new warnings
- GCC 12: 5 new warnings
- GCC 13: 14 new warnings so far
- https://gcc.gnu.org/wiki/StaticAnalyzer

GCC’s -fanalyzer

- Explores “interesting” interprocedural paths through the code via “**symbolic execution**” looking for bugs to warn about
  (for some definitions of “interesting” and of “bugs”)
- Can have false positives and false negatives

GCC's -fanalyzer

- Tracks the (approximate) state of memory
- Models various APIs via state machines (e.g. resource acquisition/release)

GCC’s -fanalyzer

- Lots of anecdotal reports that it’s finding genuine bugs in people’s code, but...
- Really only good for C code for now (but the kernel doesn’t use C++ anyway)
- Don’t use the GCC 10 version
- Expect false positives

Trying it on the kernel

- **106358:** [meta-bug] tracker bug for building the Linux kernel with -fanalyzer
  - **106218:** Analyzer false positives with Linux kernel's err.h
  - **106229:** False positives from -Wanalyzer-tainted-array-index with unsigned char index
  - ~~**104954:** Analyzer takes a very long time on Linux kernel drivers/gpu/drm/amd/display/dc/calcs/dce_calcs.c~~
    - **104955:** Analyzer slowdown with many diagnostics
    - ~~**104943:** Analyzer fails to purge state for local structs~~
  - ~~**106204:** False positive from -Wanalyzer-use-of-uninitialized-value with -ftrivial-auto-var-init=zero~~
  - ~~**106225:** False positives from -Wanalyzer-tainted-divisor~~
  - ~~**106284:** False positives from -Wanalyzer-tainted-array-index with optimized conditionals~~
  - ~~**106319:** False positives from -Wanalyzer-va-arg-type-mismatch on int promotion~~
  - ~~**106321:** False positives from -Wanalyzer-tainted-array-index with switch with ranged cases~~
  - ~~**106359:** -fanalyzer takes a very long time on Linux kernel: sound/soc/codecs/cs47l{85,90}.c~~
  - ~~**106373:** False positives from -Wanalyzer-tainted-array-index on comparison with non-const~~
  - ~~**106374:** [13 Regression] -fanalyzer ICE with certain const static vars~~
  - ~~**106383:** False positives from -Wanalyzer-va-list-exhausted~~
  - ~~**106394:** False positive from -Wanalyzer-allocation-size with empty array~~

Kernel specific tests?

- Infoleaks: leaking secrets/uninitialized data to user space
- Using attacker-controlled data without sanitization (“taint”)
- Both involve the user space boundary

Infoleak example (1)

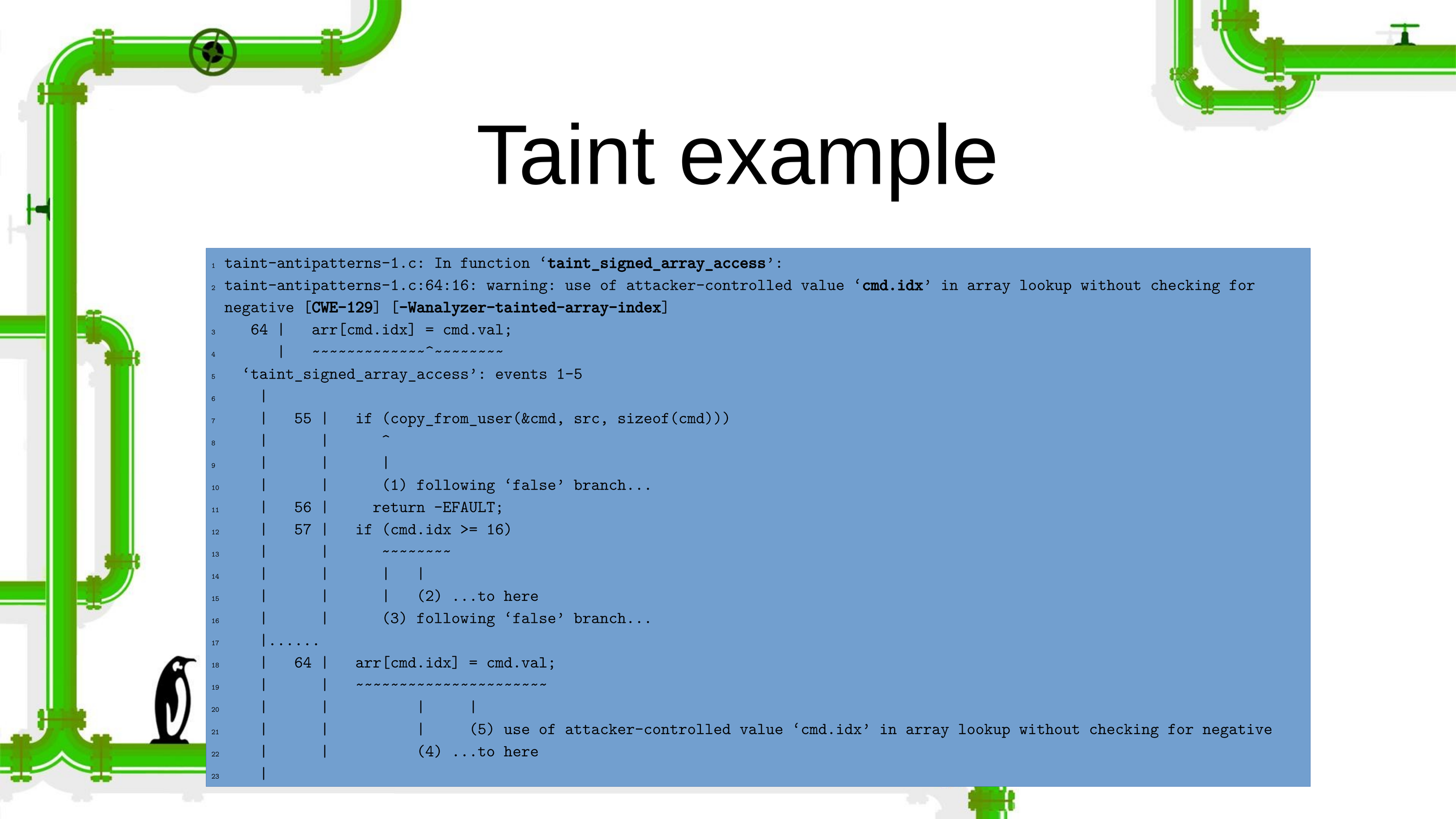
```
infoleak-CVE-2011-1078-2.c: In function ‘test_1’:
infoleak-CVE-2011-1078-2.c:28:9: warning: potential exposure of sensitive information by copying uninitialized data from
 stack across trust boundary [CWE-200] [-Wanalyzer-exposure-through-uninit-copy]
   28 |         copy_to_user(optval, &cinfo, sizeof(cinfo));
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ‘test_1’: events 1-3
    |
    |   21 |         struct sco_conninfo cinfo;
    |      |                             ^~~~~
    |      |                             |
    |      |                             (1) region created on stack here
    |      |                             (2) capacity: 6 bytes
    |......
    |   28 |         copy_to_user(optval, &cinfo, sizeof(cinfo));
    |      |         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    |      |         |
    |      |         (3) uninitialized data copied from stack here
    |
```

Infoleak example (2)

```
infoleak-CVE-2011-1078-2.c:28:9: note: 1 byte is uninitialized
   28 |         copy_to_user(optval, &cinfo, sizeof(cinfo));
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
infoleak-CVE-2011-1078-2.c:14:15: note: padding after field ‘dev_class’ is uninitialized (1 byte)
   14 |         __u8  dev_class[3];
      |               ^~~~~~~~
infoleak-CVE-2011-1078-2.c:21:29: note: suggest forcing zero-initialization by providing a ‘{0}’ initializer
   21 |         struct sco_conninfo cinfo;
      |                             ^~~~~
      |                                  = {0}
```

Taint example

```
taint-antipatterns-1.c: In function ‘taint_signed_array_access’:
taint-antipatterns-1.c:64:16: warning: use of attacker-controlled value ‘cmd.idx’ in array lookup without checking for negative [CWE-129] [-Wanalyzer-tainted-array-index]
   64 |   arr[cmd.idx] = cmd.val;
      |   ~~~~~~~~~~~~~^~~~~~~~
  ‘taint_signed_array_access’: events 1-5
    |
    |   55 |   if (copy_from_user(&cmd, src, sizeof(cmd)))
    |      |      ^
    |      |      |
    |      |      (1) following ‘false’ branch...
    |   56 |     return -EFAULT;
    |   57 |   if (cmd.idx >= 16)
    |      |      ~~~~~~~~
    |      |      |   |
    |      |      |   (2) ...to here
    |      |      (3) following ‘false’ branch...
    |......
    |   64 |   arr[cmd.idx] = cmd.val;
    |      |   ~~~~~~~~~~~~~~~~~~~~~~
    |      |          |     |
    |      |          |     (5) use of attacker-controlled value ‘cmd.idx’ in array lookup without checking for negative
    |      |          (4) ...to here
    |
```



How to implement “trust boundaries”?

- Have tried many approaches...

Approach #1

- Special-casing **copy_from_user** and **copy_to_user** in the analyzer
- Horrible hack
- Sometimes worked, but...
- Randomly breaks depending on optimization settings (ugh!)

Approach #2

- Enough attributes to allow kernel headers to indicate to the analyzer that **copy_from_user** and **copy_to_user** cross a security boundary
- Showed this at LPC 2021...

Approach #2

```
1  extern long copy_to_user(void __user *to, const void *from,
2                           unsigned long n)
3      __attribute__((access (untrusted_write, 1, 3),
4                     access (read_only, 2, 3)));
5  extern long copy_from_user(void *to, const void __user
 *from, unsigned long n)
6      __attribute__((access (write_only, 1, 3),
7                     access (untrusted_read, 2, 3)));
```

Approach #2

- Feedback at LPC 2021 was: use the __**user** annotations
- So that’s what I’ve been trying...
- But I did get __**attribute**__**((tainted_args));** into GCC12

tainted_args

```
1  #define __SYSCALL_DEFINEx(x, name, ...) \
2    asmlinkage __attribute__((tainted_args)) \
3    long sys##name(__SC_DECL##x(__VA_ARGS__))
4  struct configfs_attribute {
5    /* ... */
6    ssize_t (*store)(struct config_item *, const char *, size_t)
7       __attribute__((tainted_args));
8  };
9
```

tainted_args

```
1  SYSCALL_DEFINE5(osf_getsysinfo, unsigned long, op, void __user *, buffer,
2                  unsigned long, nbytes, int __user *, start,
3                  void __user *, arg)
4
5 taint-CVE-2011-2210-1.c: In function ‘sys_osf_getsysinfo’:
6 taint-CVE-2011-2210-1.c:69:21: warning: use of attacker-controlled value
7   ‘nbytes’ as size without upper-bounds checking [CWE-129] [-Wanalyzer-tainted-size]
8    69 |                     if (copy_to_user(buffer, hwrpb, nbytes) != 0)
9       |                         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

tainted_args

```
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -1973,8 +1973,10 @@ struct file_operations {
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
-       long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
-       long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
+       long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long)
+               ANALYZER_TAINTED_ARGS;
+       long (*compat_ioctl) (struct file *, unsigned int, unsigned long)
+               ANALYZER_TAINTED_ARGS;
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
```

tainted_args

- Where else to use **__attribute__((tainted_args)) ?**
- Ideas?

Approaches #3 and #4

- **[PATCH 0/6] RFC: adding support to GCC for detecting trust boundaries**
- https://gcc.gnu.org/pipermail/gcc-patches/2021-November/584372.html

Approach #3

- v1 of custom address spaces (2021-11-13):
- [PATCH 1a/6] RFC: Implement "**#pragma GCC custom_address_space**"
- https://gcc.gnu.org/pipermail/gcc-patches/2021-November/584375.html
- but this implementation was unfinished/didn't work

Approach #4

- **[PATCH 1b/6] Add __attribute__((untrusted))**
- https://gcc.gnu.org/pipermail/gcc-patches/2021-November/584370.html
- **__attribute__((untrusted))** for types
- Implemented in terms of pointer types and function types
- Ran into issues with **__user foo *** vs **foo __user ***
  - sparse seems to handle where the attribute goes differently from GCC

Approach #5

- v2 of custom address spaces
- I have an implementation that seems to work on the gcc side (not yet posted)
- ...but got bogged down with kernel issues...

Approach #5

```
1#pragma GCC custom_address_space(__as_user)
2#pragma GCC custom_address_space(__as_iomem)
3#define __user BTF_TYPE_TAG(user) __as_user
4#define __iomem __as_iomem
```

Approach #5

```
../../src/asm-offsets.c: In function ‘get_current’:
../../src/asm-offsets.c:8818:5: error: ‘__as_percpu’ specified for auto variable ‘pscr_ret__’
 8818 |     typeof(current_task) pscr_ret__;
      |     ^~~~~~
extern __percpu
    __attribute__((section(".data..percpu"
                           ""))) __typeof__(struct task_struct *) current_task;

static inline __attribute__((__gnu_inline__)) __attribute__((__unused__))
__attribute__((no_instrument_function))
__attribute__((__always_inline__)) struct task_struct *
get_current(void) {
  return ({
    typeof(current_task) pscr_ret__;
```

Approach #5

```
../../src/asm-offsets.c: In function ‘rb_link_node_rcu’:
../../src/asm-offsets.c:19464:20: error: cast to ‘__as_rcu’ address space pointer from disjoint generic address space pointer [-Werror]
19464 |                     ((typeof(*((typeof(*rb_link))_r_a_p__v))
      |                      ^
../../src/asm-offsets.c:19463:61: error: assignment from pointer to non-enclosed address space
19463 |                 *(volatile typeof(*&*rb_link) *)&(*&*rb_link) =
      |                                                             ^
../../src/asm-offsets.c:19463:61: note: expected ‘struct rb_node *’ but pointer is of type ‘__as_rcu struct rb_node *’
```

Approach #5

```
../../src/asm-offsets.c: In function ‘raw_copy_from_user’:
../../src/asm-offsets.c:38095:33: error: cast to generic address space
 pointer from disjoint ‘__as_user’ address space pointer [-Werror]
38095 |   return copy_user_generic(dst, (void *)src, size);
      |                                 ^
../../src/asm-offsets.c: In function ‘raw_copy_to_user’:
../../src/asm-offsets.c:38102:28: error: cast to generic address space
 pointer from disjoint ‘__as_user’ address space pointer [-Werror]
38102 |   return copy_user_generic((void *)dst, src, size);
      |                            ^
```

Approach #5

```
1 static __always_inline __must_check unsigned long
2 raw_copy_from_user(void *dst, const void __user *src, unsigned long size)
3 {
4   return copy_user_generic(dst, (__force void *)src, size);
5 }
6
7 static __always_inline __must_check unsigned long
8 raw_copy_to_user(void __user *dst, const void *src, unsigned long size)
9 {
10  return copy_user_generic((__force void *)dst, src, size);
11}
```

Approach #5

- I have an implementation of an equivalent attribute: **__attribute__((allow_address_space_cast))**
- It kind-of works…
- ...but doesn't seem to exactly match what sparse's **__force** is doing

Approach #6

- Chicken-and-egg problem: how can GCC provide something useful to the kernel…
- Needs to be supportable from the GCC side
- Needs to be useful to kernel developers
- How to prototype given GCC’s annual release cycle?

Approach #6

- Reluctant realization: use a GCC plugin as a stop-gap
- Add the bulk of the functionality to GCC
- Use a relatively small GCC plugin for the special-casing
- [committed] analyzer: add support for plugin-supplied known function behaviors
  https://gcc.gnu.org/pipermail/gcc-patches/2022-September/601387.html
- [committed] analyzer: implement trust boundaries via a plugin for Linux kernel
  https://gcc.gnu.org/pipermail/gcc-patches/2022-September/601388.html
- 240 line GCC plugin – how to make it smaller?

Other warnings

```
--- a/drivers/misc/lkdtm/perms.c
+++ b/drivers/misc/lkdtm/perms.c
@@ -108,9 +108,15 @@ static void lkdtm_WRITE_RO(void)
        /* Explicitly cast away "const" for the test and make volatile. */
        volatile unsigned long *ptr = (unsigned long *)&rodata;

+       __diag_push();
+       __diag_ignore(GCC, 11, "-Wanalyzer-write-to-const",
+                     "deliberate attempt to write to const");
+
        pr_info("attempting bad rodata write at %px\n", ptr);
        *ptr ^= 0xabcd1234;
        pr_err("FAIL: survived bad write\n");
+
+       __diag_pop();
 }
```

Other warnings

```
diff --git a/drivers/scsi/aic7xxx/aic79xx_osm.c b/drivers/scsi/aic7xxx/aic79xx_osm.c
index 928099163f0f..ccf807069c43 100644
--- a/drivers/scsi/aic7xxx/aic79xx_osm.c
+++ b/drivers/scsi/aic7xxx/aic79xx_osm.c
@@ -999,7 +1000,15 @@ ahd_linux_setup_iocell_info(u_long index, int instance, int targ, int32_t
value)
                uint8_t *iocell_info;

                iocell_info = (uint8_t*)&aic79xx_iocell_info[instance];
+
+               __diag_push();
+               __diag_ignore(GCC, 11, "-Wanalyzer-write-to-const",
+                             "Write to const aic79xx_iocell_info might be"
+                             " acceptable in __startup function"
+                             " (TODO: is it?)");
                iocell_info[index] = value & 0xFFFF;
+               __diag_pop();
+
```

Questions/discussion

- Should I try to have GCC type-check __user vs __kernel, or leave it to sparse?
- Which approach?
  - Custom address space?
  - Attribute?
  - More kernel-specific tests?

LPC 2022 - Overview

Conference Details

The Linux Plumbers Conference is the premier event for developers working at all levels of the plumbing layer and beyond.

Taking place on Monday 12th, Tuesday 13th and Wednesday 14th of September, this year we will be both in person and remote (hybrid). However to minimize technical issues, we'd appreciate most of the content presenters being in-person.

The in-person venue is the Clayton Hotel on Burlington Road, Dublin.

Clayton Hotel Burlington Road Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise.
The conference information will be shared in Irish Standard Time (IST, Europe/Dublin timezone). Clayton Hotel Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise Time.

Conference Details

The Linux Plumbers Conference is the premier event for developers working at all levels of the plumbing layer and beyond.

Taking place on Monday 12th, Tuesday 13th and Wednesday 14th of September, this year we will be both in person and remote (hybrid). However to minimize technical issues, we'd appreciate most of the content presenters being in-person.

The in-person venue is the Clayton Hotel on Burlington Road, Dublin.
Clayton Hotel Burlington Road Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise, the conference information will be shared in Irish Standard UTC+01:00, Europe/Dublin timezone)..
Clayton Hotel Leeson Street Upper,..

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

Conference Details

The Linux Plumbers Conference is the premier event for developers working at all levels of the plumbing layer and beyond.

LPC 2022 - Overview

Conference Details

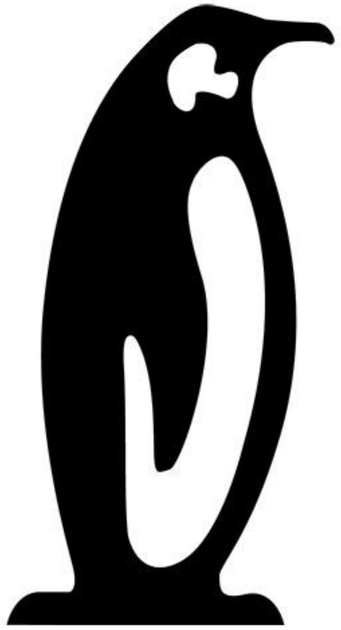
The Linux Plumbers Conference is the premier event for developers working at all levels of the plumbing layer and beyond.

Taking place on Monday 12th, Tuesday 13th and Wednesday 14th of September, this year we will be both in person and remote (hybrid). However to minimize technical issues, we'd appreciate most of the content presenters being in-person.

The in-person venue is the Clayton Hotel on Burlington Road, Dublin.

Clayton Hotel Burlington Road Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise.
The conference information will be shared in Irish Standard Time (IST, Europe/Dublin timezone). Clayton Hotel Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise Time.
The Linux Plumbers Conference is the premier event for developers working at all levels of the plumbing layer and beyond.

Taking place on Monday 12th, Tuesday 13th and Wednesday 14th of September, this year we will be both in person and remote (hybrid). However to minimize technical issues, we'd appreciate most of the content presenters being in-person. The in-person venue is the Clayton Hotel on Burlington Road, Dublin.
Clayton Hotel Burlington Road Leeson Street Upper, Dublin, D04 A318, Ireland nless specified otherwise, the conference information will be shared in Irish Standard UTC+01:00, Europe/Dublin timezone)..
Clayton Hotel Leeson Street Upper,..



Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022