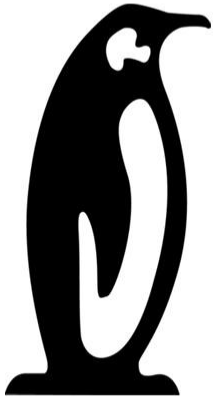


東京 2025 LINUX PLUMBERS CONFERENCE

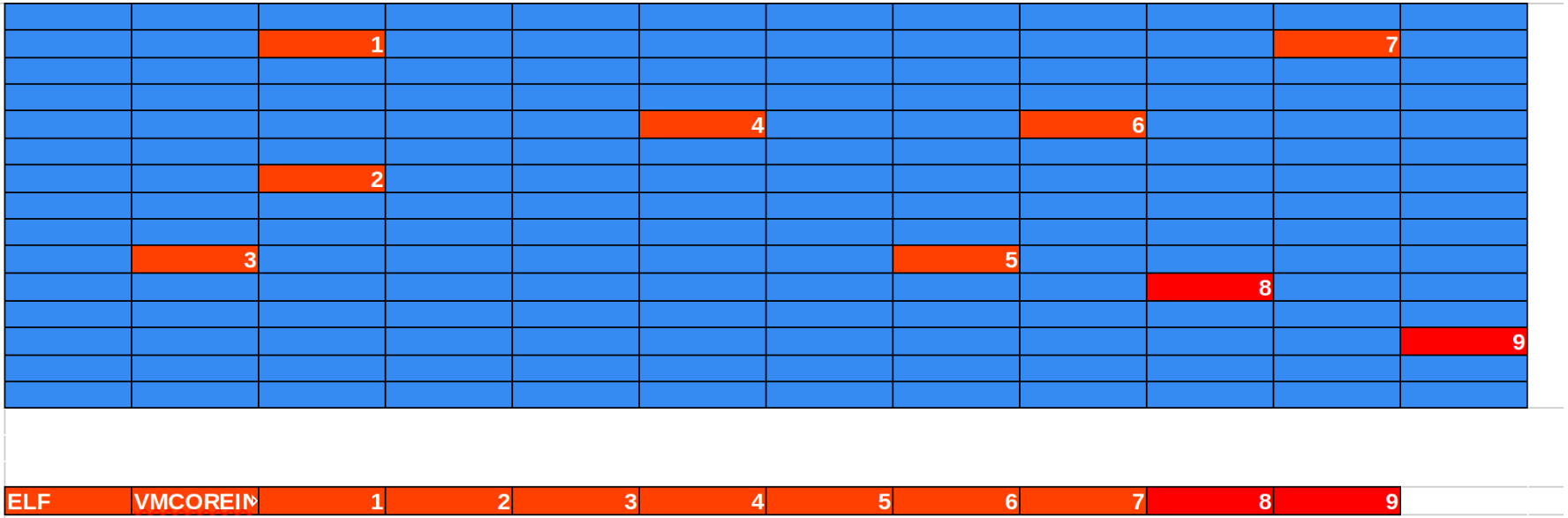
TOKYO, JAPAN / DECEMBER 11-13, 2025

meminspect - Kernel memory selection and inspection mechanism

Eugen Hristev, engineer at Linaro
eugen.hristev@linaro.org

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

At a glance



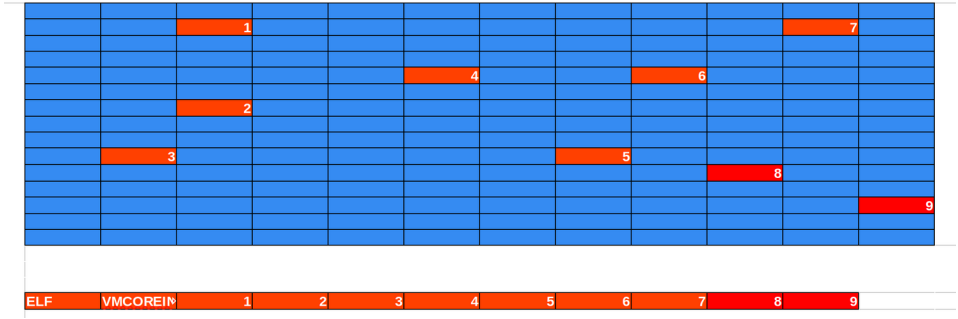
Motivation

To analyze a kernel, we **do not need** the whole memory.
Similar with what *makedumpimage* does, we can analyze the kernel with a **much smaller** memory footprint.

Why is this relevant ?
Production case is where storage and connection is **limited**.
Storing multiple dumps is also required.

Drawbacks
Cannot fully analyze the kernel. This is something we have to accept.

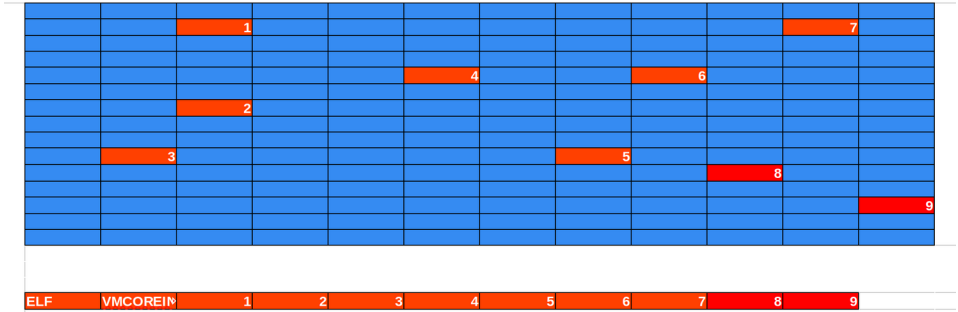
Create something that can be analyzed:
Use existing `**crash**` tool, use **vmcoreinfo**



Full motivation at Linaro Connect 2025:
LIS25 304 Kernel debugging where is my debug data :
https://www.youtube.com/watch?v=r4gII7MX9zQ

Linux-MM alignment session on Sep 17th:
https://drive.google.com/file/d/1TisXRWKiDGDqH28UvhahAFeuOBQfptua/view

Proposal



Mark specific memory regions that are vital or required for a minimal successful debug/analysis of a given kernel.
We cannot analyze everything, but it would be useful to be able to have:
nr_irqs, nr_threads, cpu count, jiffies, architecture, linux_banner, init_uts_ns, prb, etc.

Once the regions are registered, one can:

- **Traverse** the region table and consult, store, copy or analyze
- Be **notified** when a new region appears or a region is removed

Goal:

- Avoid to interfere with existing code **as much as possible**
- Make it **simple** for anyone to use

Real use case:

- Firmware can recreate the table (Qualcomm minidump e.g.)
- Android Kinfo debug information

Previous proposals:

- Use pstore as a backend, create some different kind of pstore zones[1].
- Kmemdump 3 RFC versions [2][3][4]
- Minidump driver by Mukesh Ohja[5]
- Meminspect v1 patch series[6]

[1] https://lore.kernel.org/all/20250217101706.2104498-1-eugen.hristev@linaro.org/
[2] https://lore.kernel.org/lkml/20250422113156.575971-1-eugen.hristev@linaro.org/
[3] https://lore.kernel.org/all/20250724135512.518487-1-eugen.hristev@linaro.org/
[4] https://lore.kernel.org/all/20250912150855.2901211-1-eugen.hristev@linaro.org/
[5] https://lore.kernel.org/lkml/20240131110837.14218-1-quic_mojha@quicinc.com/
[6] https://lore.kernel.org/all/20251119154427.1033475-1-eugen.hristev@linaro.org/

Static variables

Separate section for annotations:

```
#define MEMINSPECT_ENTRY(idx, sym, sz)                                          \
        static const struct inspect_entry __UNIQUE_ID(__inspect_entry_##idx)    \
        __used __section(".inspect_table") = { .id = idx,                       \
                                               .va = (void *)&(sym),            \
                                               .size = (sz),                    \
                                             }

#define MEMINSPECT_SIMPLE_ENTRY(sym)    \
        MEMINSPECT_ENTRY(MEMINSPECT_ID_##sym, sym, sizeof(sym))
```

In the code:

```
diff --git a/kernel/irq/irqdesc.c b/kernel/irq/irqdesc.c
index db714d3014b5..89538324a95a 100644
--- a/kernel/irq/irqdesc.c
+++ b/kernel/irq/irqdesc.c
@@ -16,6 +16,7 @@
+#include <linux/meminspect.h>

[...]
 static unsigned int nr_irqs = NR_IRQS;
+MEMINSPECT_SIMPLE_ENTRY(nr_irqs);
```

Dynamic memory

memblock allocs with a flag

```
/* Independent flag for allocating with MEMBLOCK_INSPECT */
#define MEMBLOCK_ALLOC_INSPECT          2
```

New flag for block

```
enum memblock_flags {
        MEMBLOCK_RSRV_NOINIT    = 0x10, /* don't initialize struct pages */
        MEMBLOCK_RSRV_KERN      = 0x20, /* memory reserved for kernel use */
        MEMBLOCK_KHO_SCRATCH    = 0x40, /* scratch memory for kexec handover */
+       MEMBLOCK_INSPECT        = 0x80, /* memory selected for kernel inspection */
 };
```

API to mark blocks as inspectable

```
int __init_memblock memblock_mark_inspect(phys_addr_t base, phys_addr_t size)
int __init_memblock memblock_clear_inspect(phys_addr_t base, phys_addr_t size)
```

In code:

```
mem_section = memblock_alloc_or_panic(size, align);
memblock_mark_inspect(virt_to_phys(mem_section), size);
```

Dynamic memory

What about in a driver or other cases ?

```
void meminspect_register_id_pa(enum meminspect_uid id, phys_addr_t zone,
                               size_t size, unsigned int type);
#define meminspect_register_pa(...) \
        meminspect_register_id_pa(MEMINSPECT_ID_DYNAMIC, __VA_ARGS__, MEMINSPECT_TYPE_REGULAR)

#define meminspect_register_id_va(id, va, size) \
        meminspect_register_id_pa(id, virt_to_phys(va), size, MEMINSPECT_TYPE_REGULAR)

#define meminspect_register_va(...) \
        meminspect_register_id_va(MEMINSPECT_ID_DYNAMIC, __VA_ARGS__)
```

e.g.:

```
/* Register 8 bytes before and after, to catch the marker too */
meminspect_lock_register_id_va(MEMINSPECT_ID_CONFIG,
                                    (void *)&kernel_config_data - 8,
                                   &kernel_config_data_end - &kernel_config_data + 16);
```

User drivers

Traverse the table

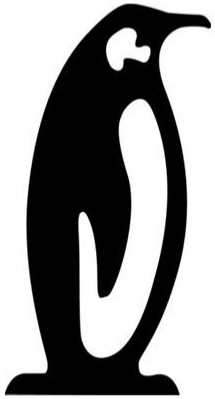
```
typedef void (*MEMINSPECT_ITERATOR_CB)(void *priv, const struct inspect_entry *ie);
void meminspect_lock_traverse(MEMINSPECT_ITERATOR_CB cb);
```

Get notified

```
int meminspect_notifier_register(struct notifier_block *n);
int meminspect_notifier_unregister(struct notifier_block *n);
```

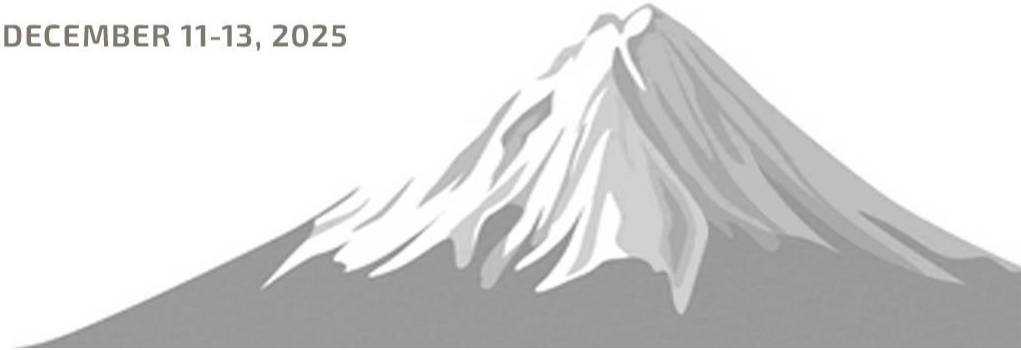
e.g. :

```
static int qcom_md_notifier_cb(struct notifier_block *nb,
                               unsigned long code, void *entry)
{       struct minidump *md = container_of(nb, struct minidump, nb);

        if (code == MEMINSPECT_NOTIFIER_ADD)
                register_md_region(md, entry);
        else if (code == MEMINSPECT_NOTIFIER_REMOVE)
                unregister_md_region(md, entry);

        return 0;
}

static struct notifier_block qcom_minidump_meminspect_nb = {
        .notifier_call = qcom_md_notifier_cb,
};
```



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025



Additional info

```
      KERNEL: /home/eugen/linux-minidump/vmlinux  [TAINTED]
    DUMPFILE: /home/eugen/a
        CPUS: 8 [OFFLINE: 7]
        DATE: Thu Jan  1 02:00:00 EET 1970
      UPTIME: 00:01:21
       TASKS: 0
    NODENAME: qemuarm64
     RELEASE: 6.18.0-rc2-00030-gceb0b3df607e
     VERSION: #46 SMP PREEMPT Wed Nov 19 14:39:12 EET 2025
     MACHINE: aarch64  (unknown Mhz)
      MEMORY: 0
       PANIC: ""
crash>

crash> log
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd4b2]
[    0.000000] Linux version 6.18.0-rc2-00030-gceb0b3df607e (eugen@eugen-station) (aarch64-none-linux-gnu-gcc (Arm GNU Toolchain
    13.3.Rel1 (Build arm-13.24)) 13.3.1 20240614, GNU ld (Arm GNU Toolchain 13.3.Rel1 (Build arm-13.24)) 2.42.0.20240614) #46 SMP
    PREEMPT Wed Nov 19 14:39:12 EET 2025
[    0.000000] KASLR enabled
[    0.000000] Machine model: Qualcomm SA8775P Ride

crash> p nr_irqs
nr_irqs = $1 = 378
crash> p jiffies
jiffies = $2 = 4294912580
crash> p /x jiffies
$3 = 0xffff2a44
```

Additional info

```
crash> p node_states
node_states = $5 =
 {{
    bits = {1}
  }, {
    bits = {1}
  }, {
    bits = {1}
  }, {
    bits = {1}
  }, {
    bits = {1}
  }, {
    bits = {0}
  }}

crash> p crash_notes
p: read error: kernel virtual address: ffffc6d4acdb9c48  type: "gdb_readmem_callback"
p: gdb request failed: p crash_notes
```

Additional info

```
Elf64_Nhdr:
               n_namesz: 11 ("VMCOREINFO")
               n_descsz: 3404
                 n_type: 0 (unused)
                         OSRELEASE=6.18.0-rc2-00030-gceb0b3df607e
                         BUILD-ID=112d2b2150a31968a667edb81d3ea792a055fdce
                         PAGESIZE=4096
                         SYMBOL(init_uts_ns)=ffffc6d4acd58dd0
                         OFFSET(uts_namespace.name)=0
                         SYMBOL(node_online_map)=ffffc6d4ac83b368
                         SYMBOL(swapper_pg_dir)=ffffc6d4abcc3000
                         SYMBOL(_stext)=ffffc6d4a9e90000
                         NUMBER(VMALLOC_START)=0xffff800080000000
                         SYMBOL(vmemmap)=fffffdffbe000000
                         SYMBOL(mem_section)=ffff000ecfca5800
                         LENGTH(mem_section)=8192
                         SIZE(mem_section)=16
                         OFFSET(mem_section.section_mem_map)=0
                         NUMBER(SECTION_SIZE_BITS)=27
                         NUMBER(MAX_PHYSMEM_BITS)=48
                         SIZE(page)=64
                         SIZE(pglist_data)=9856
                         SIZE(zone)=1664
                         SIZE(free_area)=104
                         SIZE(list_head)=16
                         SIZE(nodemask_t)=8
                         OFFSET(page.flags)=0
                         OFFSET(page._refcount)=52
                         OFFSET(page.mapping)=24
                         OFFSET(page.lru)=8
                         OFFSET(page._mapcount)=48
                         OFFSET(page.private)=40
                         OFFSET(page.compound_head)=8
```