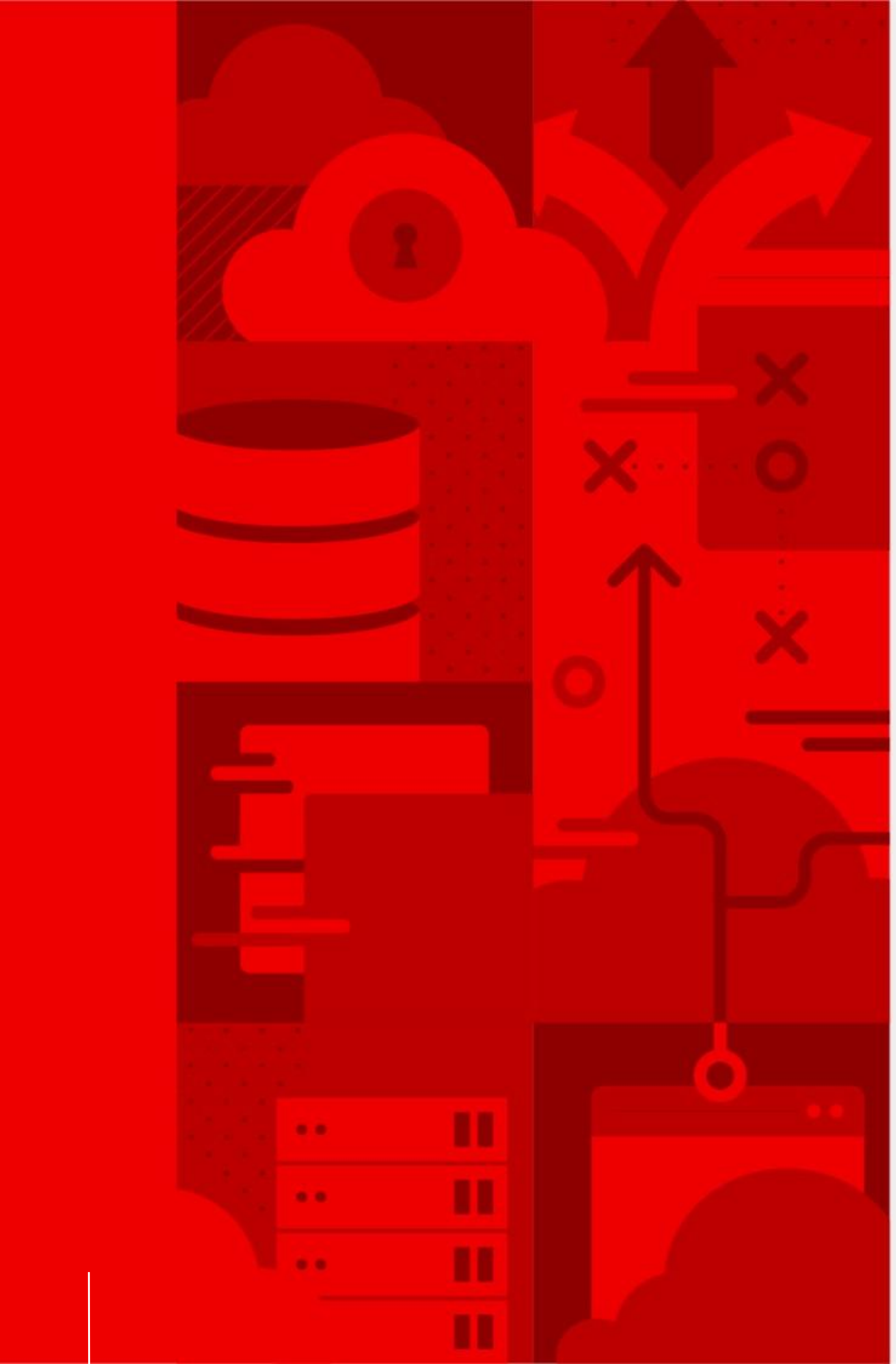


Addressing Duplicated Symbol Names in kallsyms: Introducing kas_alias

Alessandro Carminati
Principal Software Engineer

Red Hat

Who am I:

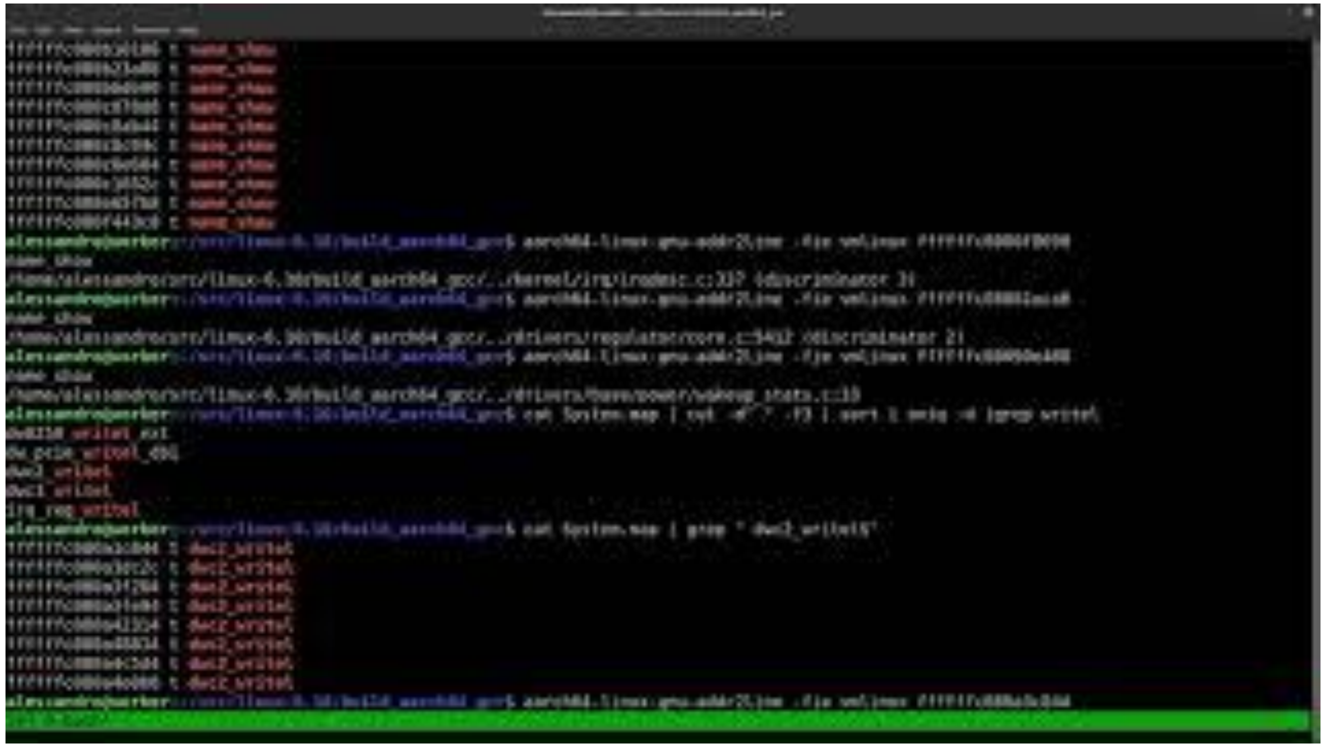
https://t.ly/BYfsA

Red Hat	Kernel Developer in Automotive team
ELISA Enabling Linux in Safety Applications	Technical Steering Committee Member Lead of the Linux Features for Safety-Critical Systems
open source	Open Source Software Contributor

Is There Really a Problem with Duplicate Symbols in the Linux Kernel?

- **Common assumption:**
  - Monolithic nature makes duplicates seem unlikely.
  - Even experienced developers may overlook this issue, I did.
- **Reality:** Duplicate symbols are there, waiting to cause trouble.
- **Personal experience:** They bit me, and they can bite you too.

Is There Really a Problem with Duplicate Symbols in the Linux Kernel?



```
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
ffffffc0[illegible] t name_show
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ aarch64-linux-gnu-addr2line -fie vmlinux ffffffc0[illegible]
name_show
/home/alessandro/src/linux-6.10/build_aarch64_gcc/../kernel/irq/irqdesc.c:[illegible] (discriminator 3)
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ aarch64-linux-gnu-addr2line -fie vmlinux ffffffc0[illegible]
name_show
/home/alessandro/src/linux-6.10/build_aarch64_gcc/../drivers/regulator/core.c:[illegible] (discriminator 2)
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ aarch64-linux-gnu-addr2line -fie vmlinux ffffffc0[illegible]
name_show
/home/alessandro/src/linux-6.10/build_aarch64_gcc/../drivers/base/power/wakeup_stats.c:[illegible]
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ cat System.map | cut -d' ' -f3 | sort | uniq -d |grep writel
[illegible]_writel_ext
dw_pcie_writel_dbi
dwc2_writel
dwc3_writel
irq_reg_writel
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ cat System.map | grep " dwc2_writel$"
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
ffffffc0[illegible] t dwc2_writel
alessandro@[illegible]:~/src/linux-6.10/build_aarch64_gcc$ aarch64-linux-gnu-addr2line -fie vmlinux ffffffc0[illegible]
```

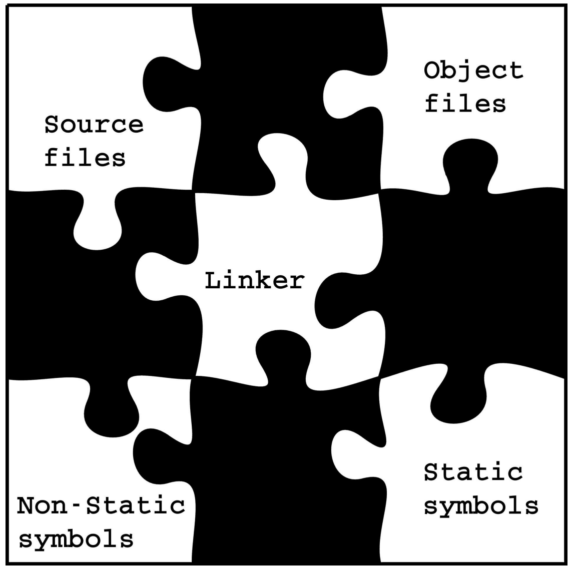
https://t.ly/P6oXu

Why do we care about duplicate symbols?

- Duplicate symbols may seem irrelevant to ordinary users.
  - If the kernel works, why should I care?
- In debug session, when you're **tracing** the kernel, they might be a concern.
- **Live patching**: replacing a function in a working production machine, can rise concerns.

Why Do Duplicate Symbols Happen in the Kernel?

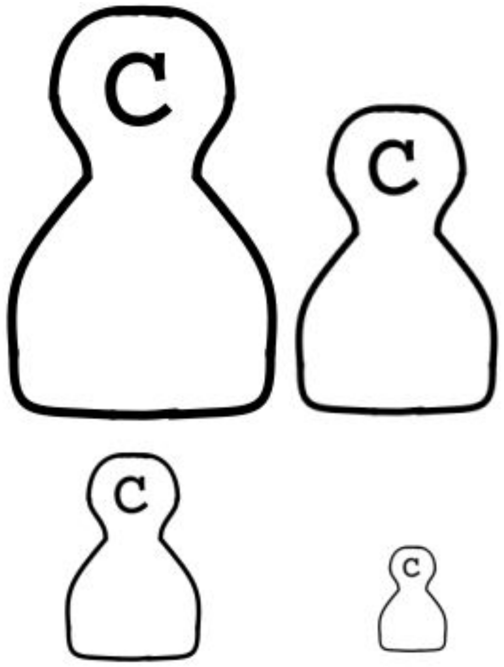
- Kernel is monolithic, but it is not a single giant source file.
- Made up of many source code files compiled into object files linked together.
- Source file depends on other source files, mostly headers.
- Static declared objects can create duplicates since they are not used at link time.
- Header files inclusion contributes to duplicates.
- C file includes another C file, can create duplicate names with different bodies.



The "Include C" Case

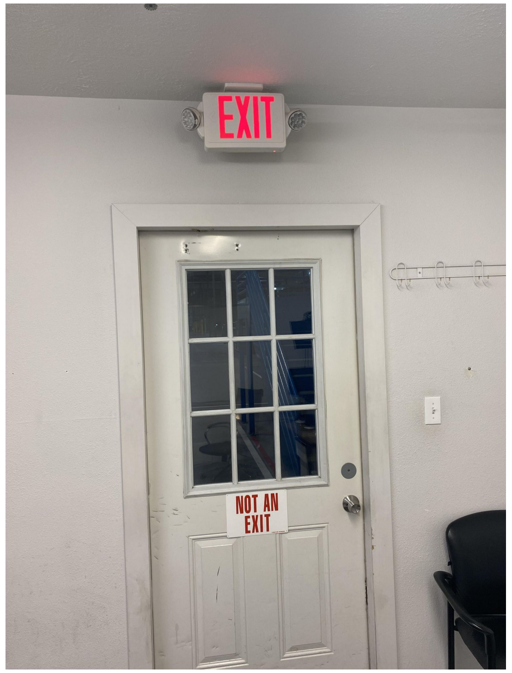
https://t.ly/3YF1B

- Occurs when a kernel C file includes another C file.
- Affects only 0.4% of kernel source files.
- Occasionally present in less popular drivers, but also present in `compat_binfmt_elf.c` which is very popular.
- C file inclusion duplicates code, similar to header files.
- Symbols contained in C file are typically complex and can depend on macros.
- `#line` directive to modify the debug information included in the object.
  - PoC available at the URL in the QR.



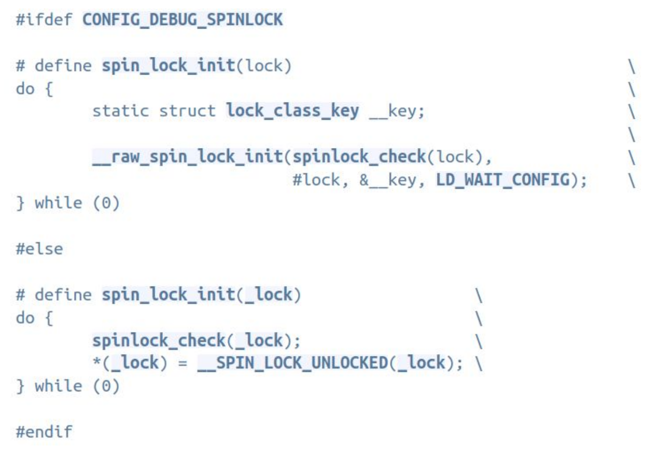
Does LTO Mitigate This Situation?

- Link-time optimization seeks to reduce or eliminate duplicate calculations by analyzing the entire program.
- LTO is supported by **GCC** and **LLVM**
- Kernel builds can have LTO only using LLVM
- LLVM has two modes of LTO
  - **monolithic LTO**
  - **ThinLTO**
- LTO is expected to handle duplicate objects that come when objects are linked together.
  - Only **monolithic LTO** provides this by mangling equal name objects.
- Problem solved?
  - While **monolithic LTO** handles duplicates by mangling their name, it does not provide any mean to distinguish them.

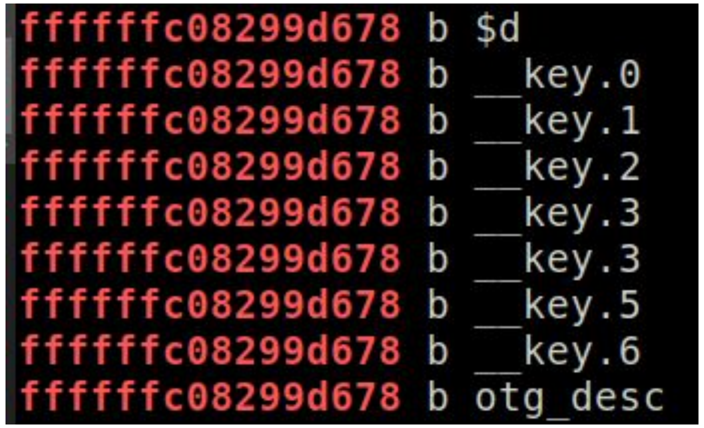


Sometimes We Also Have Duplicate Addresses

- Duplicate symbol names aren't the only issue in the kallsyms table.
- Can also find symbols sharing the same address.
- More common for data symbols than text symbols.
- Zero-sized objects causing duplicate addresses for data.
- `Lock_class_key` is zero-sized if `CONFIG_LOCKDEP` is not defined



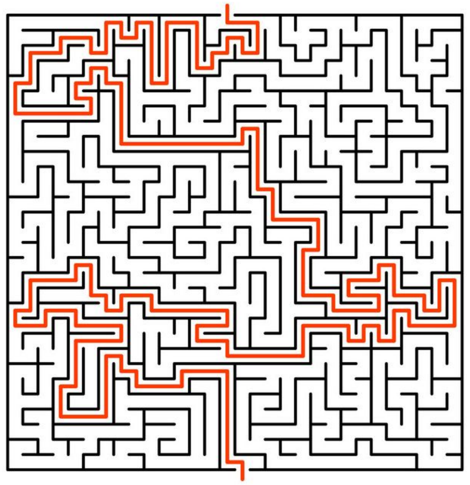
```
#ifdef CONFIG_DEBUG_SPINLOCK

# define spin_lock_init(lock)                                   \
do {                                                            \
        static struct lock_class_key __key;                     \
                                                                \
        __raw_spin_lock_init(spinlock_check(lock),              \
                             #lock, &__key, LD_WAIT_CONFIG);    \
} while (0)

#else

# define spin_lock_init(_lock)                  \
do {                                            \
        spinlock_check(_lock);                  \
        *(_lock) = __SPIN_LOCK_UNLOCKED(_lock); \
} while (0)

#endif
```



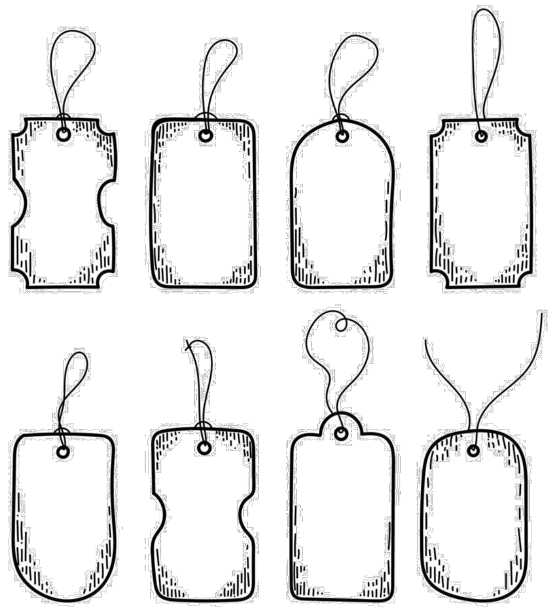
```
ffffffc08299d678 b $d
ffffffc08299d678 b __key.0
ffffffc08299d678 b __key.1
ffffffc08299d678 b __key.2
ffffffc08299d678 b __key.3
ffffffc08299d678 b __key.3
ffffffc08299d678 b __key.5
ffffffc08299d678 b __key.6
ffffffc08299d678 b otg_desc
```

What Do You Propose to Address These Duplicate Symbols?

- Create aliases for symbols that appear to be duplicates.
- Aliases avoid the disruption to kallsyms users caused by sudden changes.
- Duplicate symbols have been managed locally over time.
  - Live patch uses `kallsyms_on_each_match_symbol`to handle duplicates.
  - Functions like `compare_symbol_name`address LTO mangled names.
- Aliases maintain existing function behavior while supporting alias-aware computations.



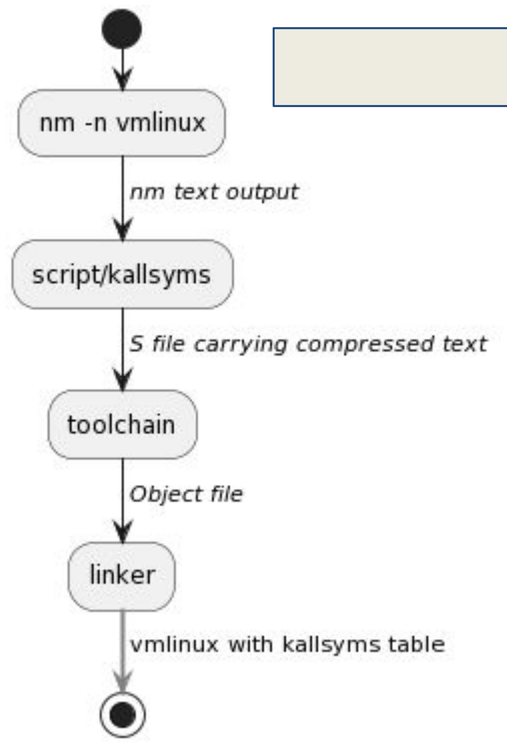
How to Tag Symbols?

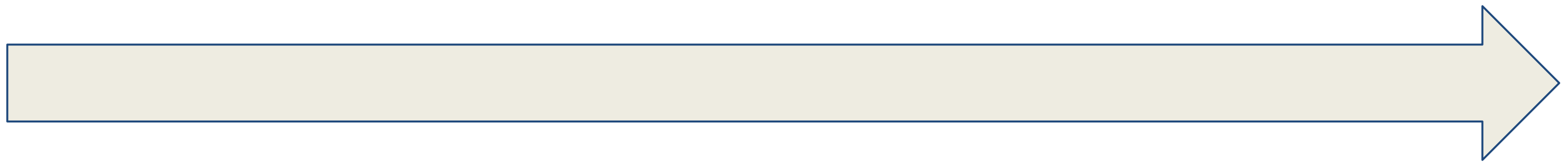


- Duplicate symbols can occur even within a single compiler unit.
  - **Static local variables** for **data**.
  - **Nested functions** for **text**.
  - Compilers usually mangle these names, but the symbols’ identity issue can still persist.
  - Not aware kernel code uses any, but possible.
- My tagging strategy: tag symbols with the source file name and line number

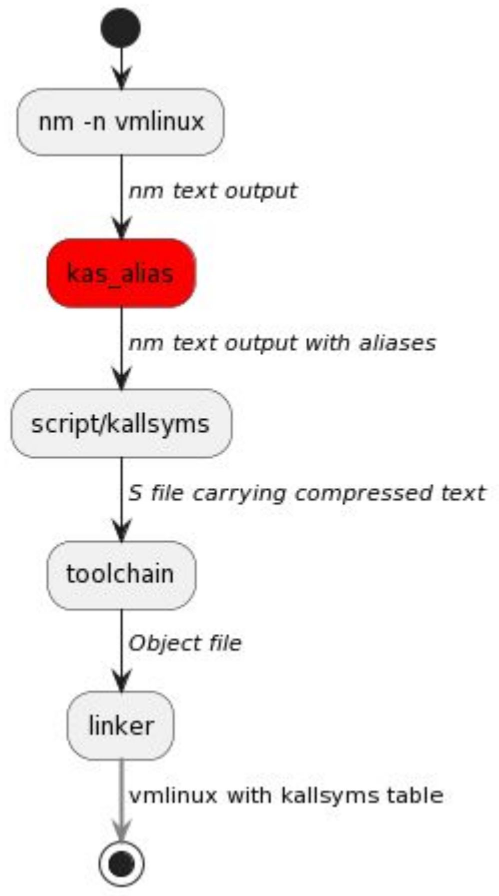
➕ Allows immediate identification of the symbol.
➕ Includes duplicates within compiler’s unit.
➖ Symbol table not consistent across kernel source code versions.

Handle duplicate in vmlinux





- The current pipeline uses **nm** to gather raw data for kallsyms.
- At vmlinux linking time, scripts produce **`System.map`** and kallsyms data.
- Kallsyms data is converted to fit as an object in the kernel image.
- **Proposal:** Tap into this pipeline to add aliases.
- **Result:** kallsyms data will embed aliases.
- This process enhances only the kernel image.


nm -n vmlinux
nm text output
kas_alias
nm text output with aliases
script/kallsyms
S file carrying compressed text
toolchain
Object file
linker
vmlinux with kallsyms table

Handle duplicate in modules

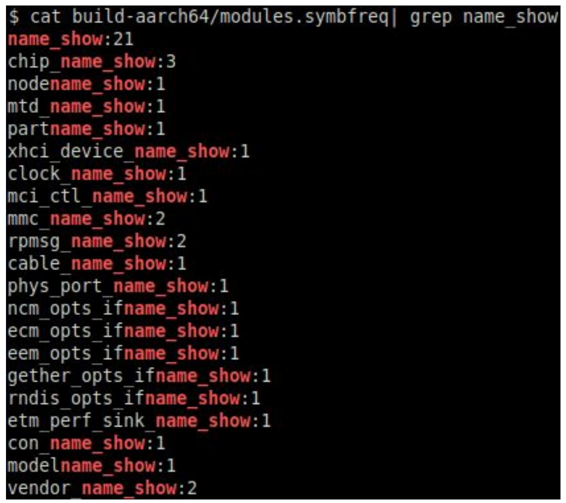
- Requirements:
  - To have the consistency of all symbols in a given kernel build, all objects need to be analyzed.
  - Best if the process of analysis for tagging is executed once.
  - Modules needs their own strategy, since nm pipeline tapping can be only used for vmlinux.

- Strategy:
  - Reuse the same requirement already introduces for BTF production.
  - Trigger a single computation at vmlinux link time.
  - Use **`objcopy`** to modify the symbol table and create aliases.

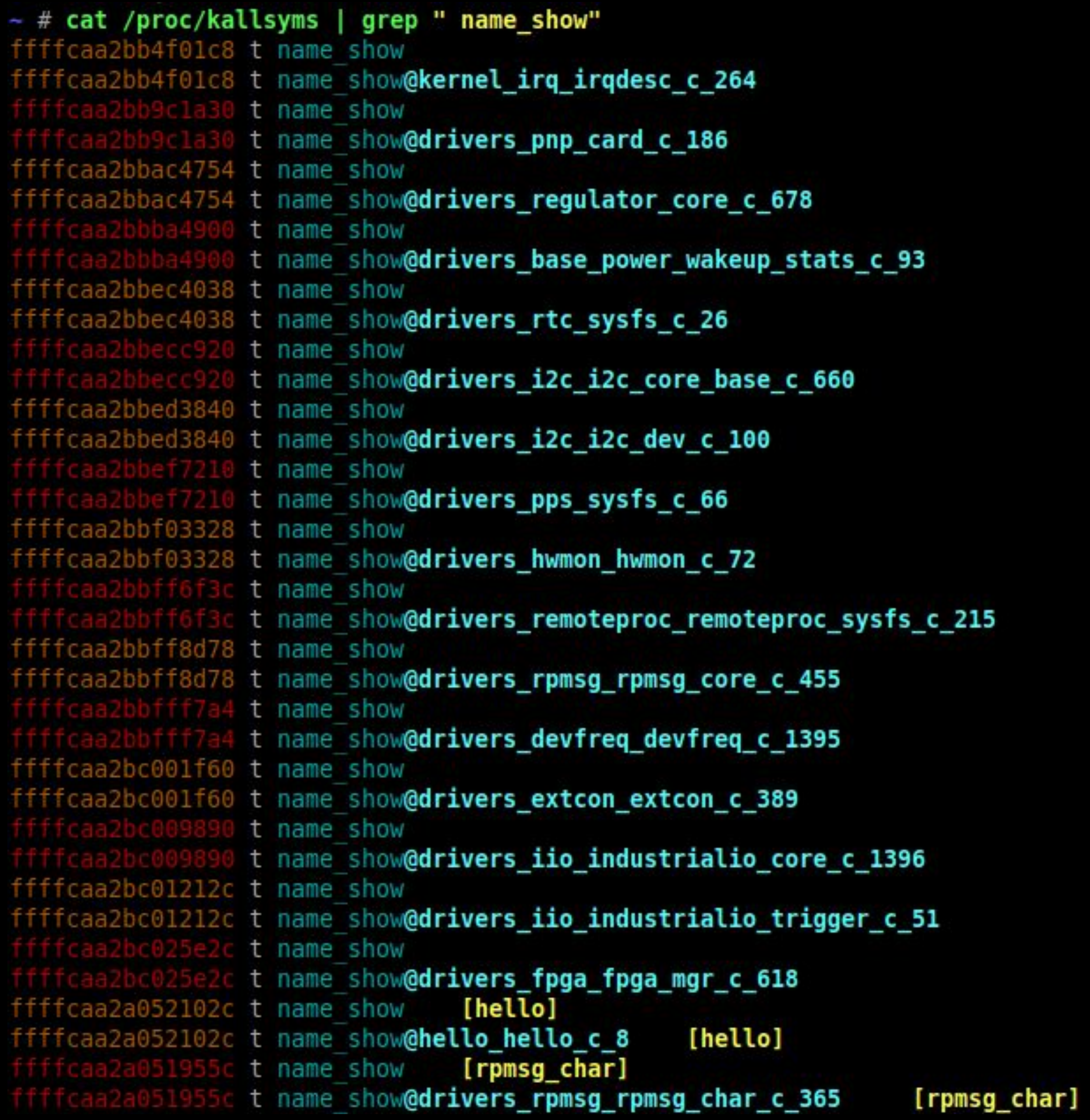
Red Hat

Managing later builds for both OoTs and in-trees Modules

- Requirements:
  - Have symbols' statistic available at build time.
- Pros vs Cons:
  - Aliases can be generated only for the new objects according to the existing statistics.
  - A new file needs to be added to the kernel distribution artifacts, the file that contains symbols' statistic.
  - The aliases can be added only to the new module's symbols. If the need for a new alias is added, only the new module can have one.
    - If the need for a new alias is for the kernel image, it generally comes first, allowing plain names for in-tree symbols and aliases for new module symbols.
    - If the need for a new alias is for a module, the order isn't guaranteed, but chances often works in favor of the in-tree module.



```
$ cat build-aarch64/modules.symbfreq| grep name_show
name_show:21
chip_name_show:3
nodename_show:1
mtd_name_show:1
partname_show:1
xhci_device_name_show:1
clock_name_show:1
mci_ctl_name_show:1
mmc_name_show:2
rpmsg_name_show:2
cable_name_show:1
phys_port_name_show:1
ncm_opts_ifname_show:1
ecm_opts_ifname_show:1
eem_opts_ifname_show:1
gether_opts_ifname_show:1
rndis_opts_ifname_show:1
etm_perf_sink_name_show:1
con_name_show:1
modelname_show:1
vendor_name_show:2
```



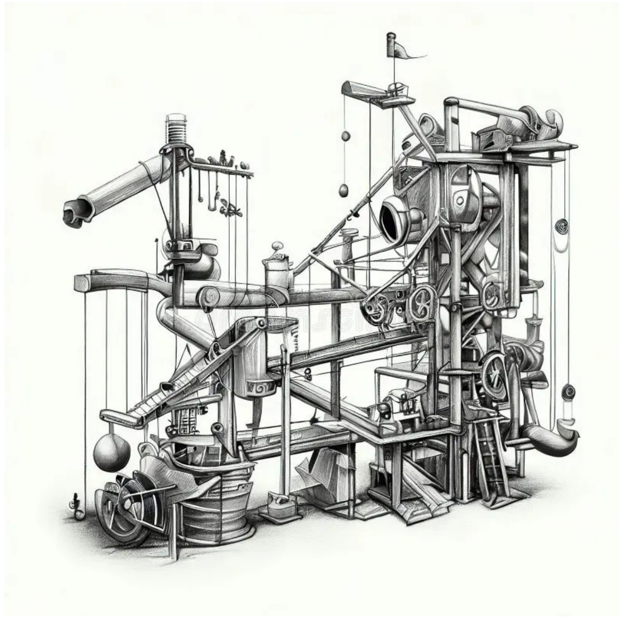
```
~ # cat /proc/kallsyms | grep " name_show"
ffffcaa2bb4f01c8 t name_show
ffffcaa2bb4f01c8 t name_show@kernel_irq_irqdesc_c_264
ffffcaa2bb9c1a30 t name_show
ffffcaa2bb9c1a30 t name_show@drivers_pnp_card_c_186
ffffcaa2bbac4754 t name_show
ffffcaa2bbac4754 t name_show@drivers_regulator_core_c_678
ffffcaa2bbba4900 t name_show
ffffcaa2bbba4900 t name_show@drivers_base_power_wakeup_stats_c_93
ffffcaa2bbec4038 t name_show
ffffcaa2bbec4038 t name_show@drivers_rtc_sysfs_c_26
ffffcaa2bbecc920 t name_show
ffffcaa2bbecc920 t name_show@drivers_i2c_i2c_core_base_c_660
ffffcaa2bbed3840 t name_show
ffffcaa2bbed3840 t name_show@drivers_i2c_i2c_dev_c_100
ffffcaa2bbef7210 t name_show
ffffcaa2bbef7210 t name_show@drivers_pps_sysfs_c_66
ffffcaa2bbf03328 t name_show
ffffcaa2bbf03328 t name_show@drivers_hwmon_hwmon_c_72
ffffcaa2bbff6f3c t name_show
ffffcaa2bbff6f3c t name_show@drivers_remoteproc_remoteproc_sysfs_c_215
ffffcaa2bbff8d78 t name_show
ffffcaa2bbff8d78 t name_show@drivers_rpmsg_rpmsg_core_c_455
ffffcaa2bbfff7a4 t name_show
ffffcaa2bbfff7a4 t name_show@drivers_devfreq_devfreq_c_1395
ffffcaa2bc001f60 t name_show
ffffcaa2bc001f60 t name_show@drivers_extcon_extcon_c_389
ffffcaa2bc009890 t name_show
ffffcaa2bc009890 t name_show@drivers_iio_industrialio_core_c_1396
ffffcaa2bc01212c t name_show
ffffcaa2bc01212c t name_show@drivers_iio_industrialio_trigger_c_51
ffffcaa2bc025e2c t name_show
ffffcaa2bc025e2c t name_show@drivers_fpga_fpga_mgr_c_618
ffffcaa2a052102c t name_show    [hello]
ffffcaa2a052102c t name_show@hello_hello_c_8    [hello]
ffffcaa2a051955c t name_show    [rpmsg_char]
ffffcaa2a051955c t name_show@drivers_rpmsg_rpmsg_char_c_365     [rpmsg_char]
```

Open Issues in current implementation

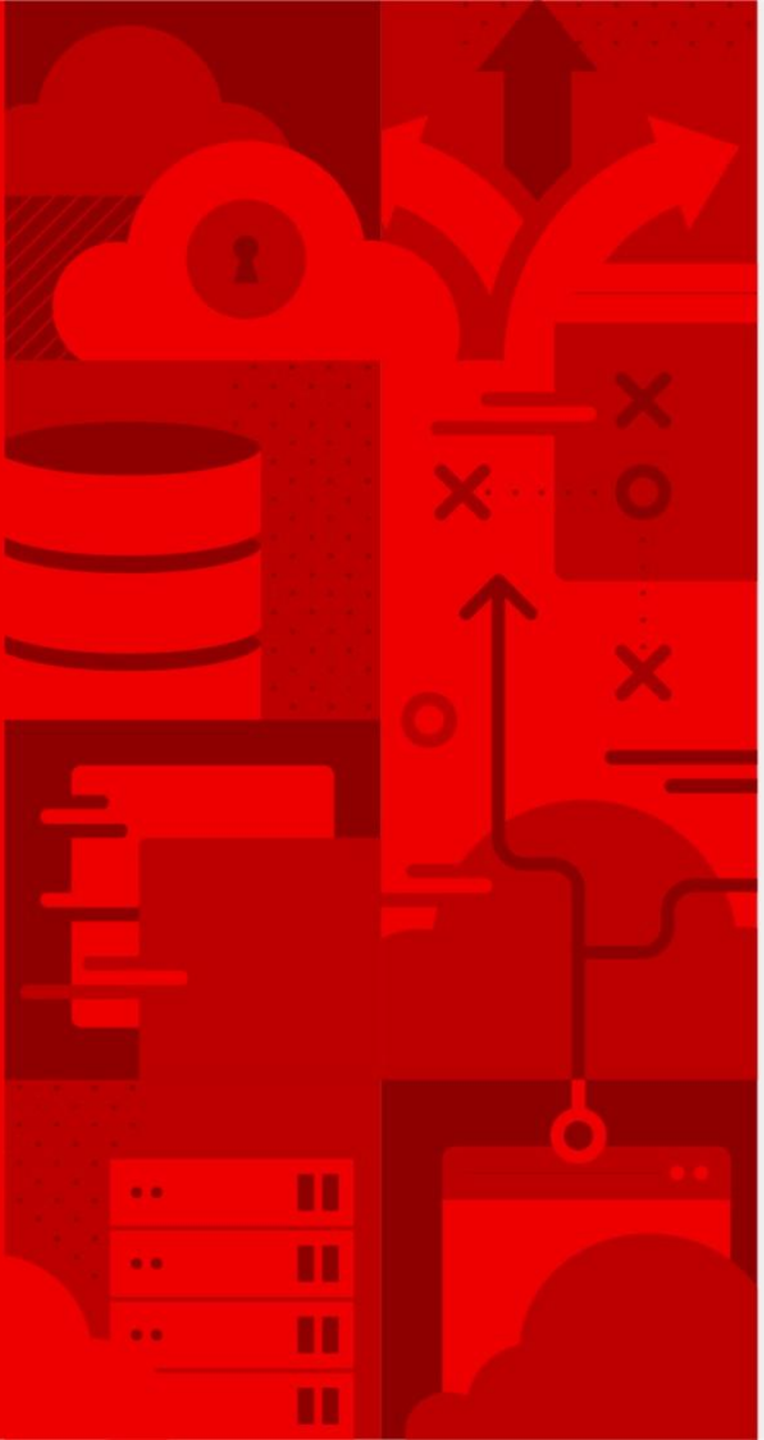
https://t.ly/470An

Open Issues in Version 7:

- **Current Symbol Table Handling:** Blatantly breaking the Makefile rule… modifying input files. Need to find a cleaner approach that respects the rules and avoids overcomplicating the build process.
- **LTO-Mangled Symbols:** Expanding support to include these in the duplicate audience.
- **Misleading Debug Info:** Need patches to fix issues when C files are included… `#line` is my friend.
- **Community Feedback:** Mixed reactions, especially around using `addr2line` for tagging. Seeking more input to refine the work.



Questions



Thank you