Evolving the BPF Type Format (BTF)

Alan Maguire
Linux Kernel Networking, Oracle
alan.maguire@oracle.com
blogs.oracle.com/linuxkernel
November 2023

The value of the BPF Type Format

The BPF Type Format compact description of types, functions, variables that is embedded in kernel/modules for most distros benefits

- BPF users, by unlocking advanced BPF tracing features (fentry, fexit), Compile Once - Run Everywhere (CO-RE), struct_ops, kfuncs, etc
- ftrace users,
  by providing argument types and return values to allow more powerful trac ing
- Debuggers, since having always-available information about kernel types, functions etc is a major win. See Brendan Gregg's "Fast by Friday" talk; BTF can help realize that vision.

Once we can assume BTF availability, all sorts of solutions become possible!

Goals

So we should seek (in order of priority)

- To maximize availability of BTF, handling issues that limit its adoption.
- To maximize the value of BTF to users once present, by solving the usability issues they care about.
- To make it easy for developers to evolve the format when necessary.

Such evolution should strive to

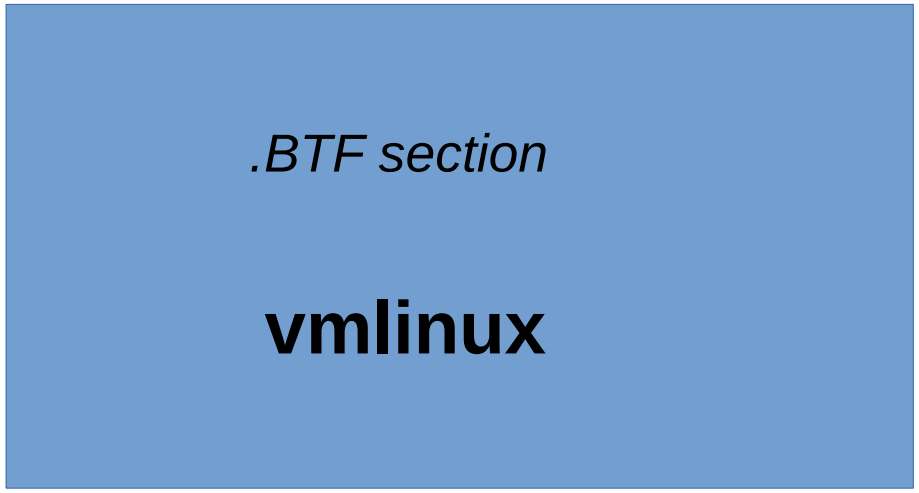
- Avoid breaking existing tools where possible.
- Minimize coordination pain between kernel and toolchains when changes are made.

BTF adoption issues

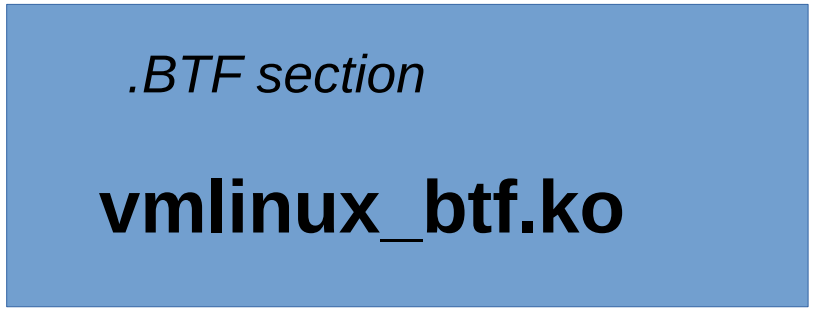
- Issue #1: BTF is too big to be stored in vmlinux for embedded systems.
- Solution: support CONFIG_DEBUG_INFO_BTF=m, where BTF is delivered in vmlinux_btf.ko instead. Since modules can live on different partitions for such systems, module-based delivery of BTF solves the problem. When vmlinux_btf.ko is loaded, vmlinux BTF should still appear in /sys/kernel/btf/vmlinux; this will require no toolchain changes and mean the user experience is identical once vmlinux_btf.ko is loaded.

BTF adoption issues – solving vmlinux BTF size problems

CONFIG_DEBUG_INFO_BTF=y



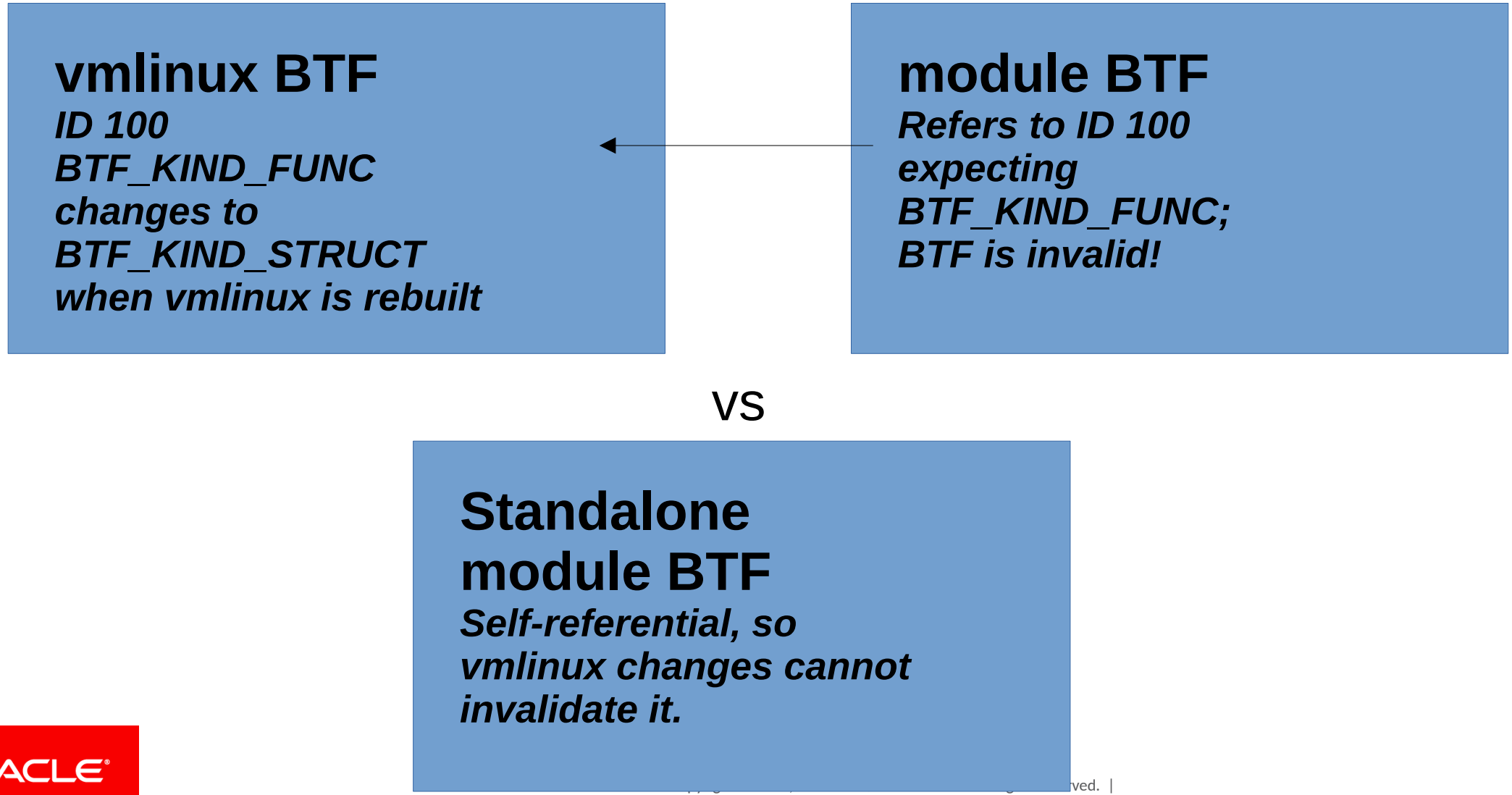
CONFIG_DEBUG_INFO_BTF=m



BTF adoption issues

- Issue #2: modules build less frequently than the underlying kernel can end up with invalidated BTF, as that BTF is defined relative to base vmlinux BTF.
- Solution: support “standalone” BTF, which allows module builders to generate self-contained BTF. Generating such BTF is easy, but the problem is toolchains assume module BTF *is* split BTF. We can address this at BTF load time by “renumbering” the standalone BTF in-kernel representation to start at last_vmlinux_btf_id + 1 (as split BTF would).
- If we do so, it looks just like split BTF to tools; it just happens to be fully self-referential – no toolchain changes are then needed.
- Renumbering needs to handle the BTF_ids section too, since it will contain the old BTF ids.

Split BTF versus standalone BTF



BTF adoption issues

- Part of the solution for both is having BTF_BASE variable in Makefile.modfinal
- When vmlinux BTF is delivered via a module, BTF_BASE will refer to the temporary BTF location (since the vmlinux_btf.ko module may not exist yet); modules will build their BTF relative to that, while the temporary BTF will be copied to the vmlinux_btf.ko module when that module is built.
- For standalone modules, they can specify standalone BTF by specifying an empty BTF_BASE via

```
make BTF_BASE= M=path/2/module
```

- Tools will work the same as for split BTF, for example we can run

```
bpftool btf dump file /sys/kernel/btf/bpf_testmod_standalone
```

BTF usability issues

- Issue #1: Optimizations can mean that static kernel functions have .isra.0, .constprop.0 suffixes, and these were not represented in BTF. This meant we cannot trace all the functions in /proc/kallsyms.
- Solution:
  - Optimized functions are now encoded in BTF when `btf_gen_optimized` is specified for pahole (now the default)
  - However we still need to unambiguously link them to the symbol they represent, since their BTF name is the unsuffixed version ("foo" not "foo.isra.0"
  - This can be solved by fixing the second issue…

BTF usability issues

- Issue #2: Multiple static functions can have the same name but different function signatures. Currently we omit such functions from BTF to avoid ambiguity (via --skip_encoding_btf_inconsistent_proto/consistent).
- Solution:
  - BTF needs to encode addresses for functions. A simple “symbol table” would suffice, i.e. in the BTF header we could have

```
__u32   symtab_off;  /* offset of symtab section     */
__u32   symtab_len;  /* length of symtab section     */
```

  - Table format would not need linkage or names (since they are in the BTF_KIND_FUNC encoding), just `<__u32 id, __u64 addr>` pair per symbol. A BTF id (function signature) might have multiple pairs.

BTF usability issues

- Why a symbol table?
  - Tried and tested solution, simple semantics: what addresses are associated with this function signature?
  - Backwards compatible: continue to use KIND_FUNC, KIND_VAR, just have extra info in symtab.
  - Simplifies deduplication: if a new kind incorporating symbol address was added, multiple instances of the same static function would not be dedup-ed. This sort of thing can have cascading effects on BTF size, so a separation is cleaner.
  - Simpler structure: multiple instances of the same function simply mean multiple symbol table entries, so sort the symtab by BTF id
  - When BTF is loaded, we add a O(1) index from BTF id → index in symtab
  - Add API like **int bpf_get_type_addresses(struct btf *btf, __u32 id, void **addresses);**

BTF id	address
100	ffffffffc04c42a0
100	ffffffffc04c4010
120	ffffffffc08ad010

BTF usability issues

- Issue #3: Only per-CPU global variables are currently included in BTF. Having all global variables would simplify BPF code, help debuggers utilize BTF, etc.
- Solution:
  - An RFC was proposed to add variables. In investigating it the ambiguous name issue was identified for variables too.
  - Hence variables could make use of the symbol table also; again we just need a `<__u32 id, __u64 addr>` pair per symbol.

BTF usability issues

- Issue #4: If BTF encounters a BTF kind it does not recognize, it cannot parse the BTF further. So if a tool built with an older libbpf is used on newer BTF, it falls over when trying to parse it.
- Solution:
  - At BTF encoding time, encode the kind layout for each kind known about. Then parsers can use the layout to parse the BTF, even if they cannot *use* all the kinds available.
  - This allows bpftool btf dump to work in raw mode, even for kinds that are not known to it (provided it knows how to read kind layout info)

BTF usability issues

```
$ bpftool btf dump file /sys/kernel/btf/vmlinux format meta
...
kind 0    UNKNOWN    flags 0x0    info_sz 0    elem_sz 0
kind 1    INT        flags 0x0    info_sz 4    elem_sz 0
kind 2    PTR        flags 0x0    info_sz 0    elem_sz 0
kind 3    ARRAY      flags 0x0    info_sz 12   elem_sz 0
kind 4    STRUCT     flags 0x0    info_sz 0    elem_sz 12
...
```

BTF usability issues

- Issue #5: There is no mechanism to explicitly determine if kernel/module BTF are matched; when loading a module, we simply parse module BTF based on the vmlinux BTF until something fails.
- Solution:
  - Generate CRCs for vmlinux, module BTF. When module BTF is added, store its CRC and the vmlinux base CRC it was encoded relative to.
  - When loading a module, we can explicitly reject it if its base CRC != vmlinux BTF CRC
  - We can also use the presence of a module CRC and the absence of base BTF CRC to signify it is a standalone BTF module.

Issues evolving BTF

- Issue #1: BTF generation consisted of specifying a confusing set of opt-in and opt-out parameters; these needed to be coordinated with pahole versions since if the option didn't exist, pahole would exit in error.
- Solution:
  - pahole now supports the simpler --btf_features options; a set of opt-in features is supplied.
  - If features do not exist in the –-btf_features list, pahole ignores them.
  - This allows us to no longer version-check pahole for feature support (aside from checking it is the first version that supports –-btf_features), and new features no longer need to be tied to specific pahole versions.

Issues evolving BTF

- Issue #2: When new BTF kinds are added, older toolchains break
- Solution:
  - See earlier: this is an issue both for users of BTF and a tax imposed on evolving BTF; supporting kind layout can at least solve the BTF parsing problem.

The big picture

- The set of changes described previously suggest some UAPI additions to struct btf_header:

```
struct btf_header {
        __u32   type_len;       /* length of type section       */
        __u32   str_off;        /* offset of string section     */
        __u32   str_len;        /* length of string section     */
+       __u32   kind_layout_off;/* offset of kind layout section */
+       __u32   kind_layout_len;/* length of kind layout section */
+       __u32   symtab_off;     /* offset of kind layout section */
+       __u32   symtab_len;     /* length of kind layout section */
+       __u32   crc;            /* crc of BTF; used if flags set
BTF_FLAG_CRC_SET */
+       __u32   base_crc;       /* crc of base BTF; used if flags
set BTF_FLAG_BASE_CRC_SET */
 };
```

The big picture

What are the costs of all of this?

- For updated BTF header, + 24 bytes for kind layout, symtab, CRC info.
- 80 bytes (4 * NR_BTF_KINDS) for kind layout section.
- For function addresses, symtab entry (12 bytes) * number of functions (~50000) = ~0.5Mb for vmlinux BTF.
- Variables add around 2Mb to vmlinux BTF (source here)
- CRC verification required on module load for modules specifying CRCs

Current status

Work done

- Support for --btf_features in pahole
- Encoding of optimized functions in BTF

Work in progress

- V4 patch contains
  - Support for kind layout, CRC+ verification, standalone BTF + renumbering

To be done

- CONFIG_DEBUG_INFO_BTF=m, BTF symbol table, BTF variable support

Conclusion

We can make life a bit easier for folks who want to

- Adopt BTF but who are currently blocked.
- Utilize BTF, either in the BPF context or elsewhere (ftrace, debuggers, etc).
- Evolve BTF to add new features.

However since such changes involve UAPI updates, it's best to try and do them in a coordinated manner, thinking through the interactions. Happily doing so finds a bunch of synergies between the solutions needed.

Thank you!

References

- ftrace series using BTF

  https://lore.kernel.org/bpf/169272153143.160970.15584603734373446082.stgit@devnote2/

- Fast by Friday, presented by Brendan Gregg at the eBPF summit
  https://www.youtube.com/watch?v=s1mobd8t_u0

- Issues with vmlinux BTF for embedded systems
  https://lore.kernel.org/bpf/1b9e4d2c-34d4-2809-6c91-d14092061581@oracle.com/

- Issues with module BTF:
  https://lore.kernel.org/bpf/CAEf4Bzbi7XiNVKYmhmiywsU0PWVg30=EOhsBWFd_xsj2vpy1xg@mail.gmail.com/

- Issues with optimization-suffixed functions

  https://lore.kernel.org/lkml/20230109094247.1464856-1-imagedong@tencent.com/

References

- Support in pahole for “--btf_features”

  https://lore.kernel.org/bpf/20231023095726.1179529-1-alan.maguire@oracle.com/

- RFC for BTF variable support.
  https://lore.kernel.org/bpf/20221104231103.752040-1-stephen.s.brennan@oracle.com/

- BTF kind layout, CRC, standalone module support v4 patch series

  https://lore.kernel.org/bpf/20231112124834.388735-1-alan.maguire@oracle.com/

  pahole patch to support kind layout, CRCs in vmlinux/module BTF

  https://lore.kernel.org/bpf/20231110111533.64608-1-alan.maguire@oracle.com/


ORACLE®