BTF linking and deduplication in the Linux kernel using the toolchain

Nick Alcock <nick.alcock@oracle.com>

CTF is becoming BTF++

- The GNU toolchain can already generate CTF and BTF directly (like it can DWARF)
- GNU ld will gain the ability to read in both and dedup them together, using machinery in libctf
- CTFv3 is similar to BTF but not identical
- Why not make CTF identical to BTF?
- https://www.esperi.org.uk/~oranix/2024-cauldron.pdf

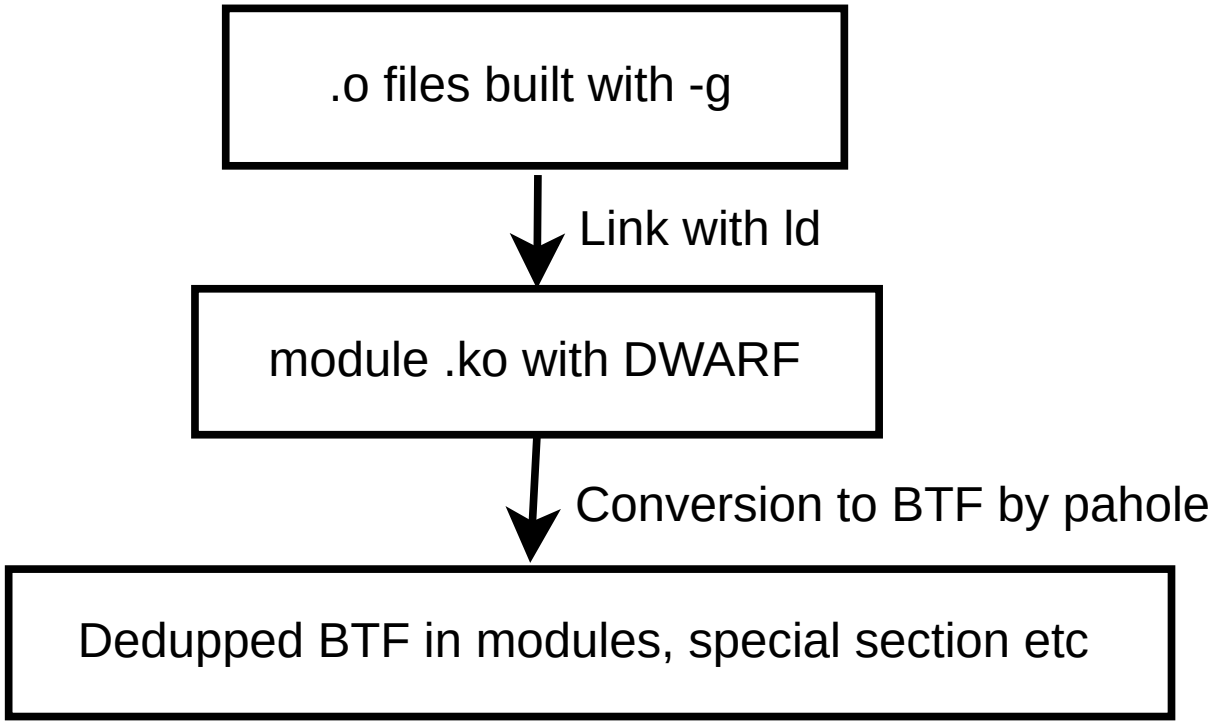
CTF as a BTF superset

- We can’t *quite* make it identical
- CTF has some things (like the ability to associate types with ELF symbols, or the ability to represent multigigabyte types) that BTF doesn’t have and likely won’t ever want
- But we can make it close enough that they use the same type section and all the same header fields: CTF just has a few more

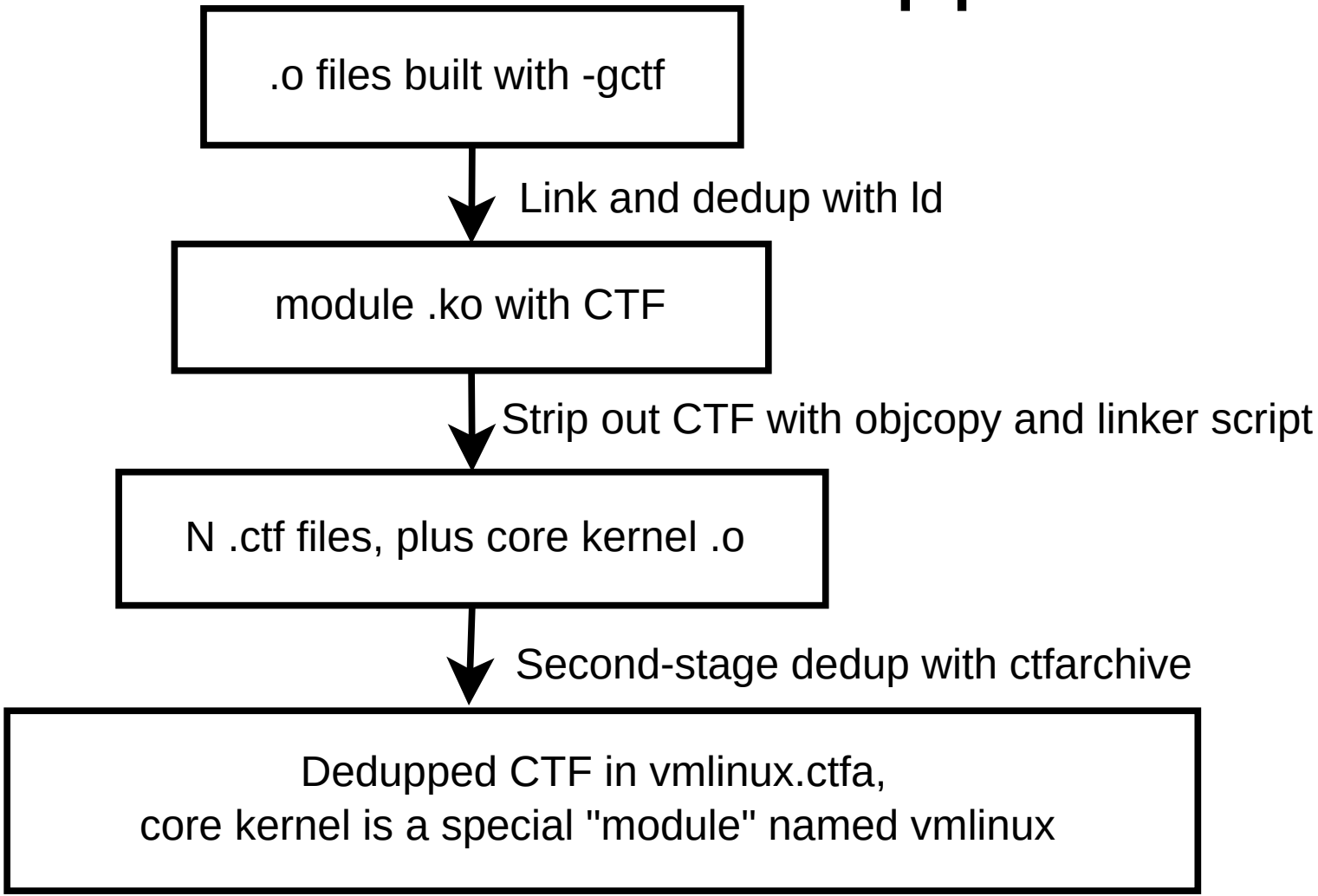
BTF dedup made easier

- If GNU ld can deduplicate BTF... why do we need a deduplicator in pahole? GNU ld (and other related simple tooling, see later) can do the same job and hand the result off to pahole: no need to generate DWARF, faster compiles, and one less deduplicator to maintain

The current approach



The current CTF approach



ctfarchive

- Two-round linker (using the same deduplicator as GNU ld): ~300 lines not counting comments
  - Round 1: dedup each module, fusing all TUs into one, marking all conflicting types as hidden
  - Round 2: dedup all modules and the core kernel against each other

Doing this with BTF

- Using the same methods:
  - Generate BTF instead of CTF (still using the toolchain);
  - Dedup it with ld and ctfarchive (renamed to btfarchive);
  - Emit the BTF into similar archives
  - Hand them off to pahole for further decoration and incorporation into the kernel
- pahole *would not need to do any dedup* or read DWARF. (If it needs DWARF for other reasons, maybe we can smuggle that in via the BTF too!). It could put the BTF exactly where it does now.

Why bother?

- Conflicting types detected and recorded reliably, though consumers need adjusting to *use* those types
- Types used by modules you don’t care about don’t even need to be loaded
- Types may move to the shared parent or disappear but otherwise never move, even when the kernel is reconfigured
- *But* consumers suddenly need to know about archives of types. This might not be ideal... so...

Simpler approaches!
Drop *conflicting types*

This would speed up ctfarchive: all types depending on conflicting types would also be dropped (or point at stubs?)

Simpler approaches!
Drop *CTF archives*

btfarchive would write out a bunch of individual dicts to... somewhere (a new subdirectory?) and tell pahole where they were (we'd want a way to indicate that a file of BTF is a parent versus a child, but that's a good idea anyway: maybe just the filename?)

Simpler approaches!
Drop the *special vmlinux "module"*

Instead, we'd put all the kernel types into the shared dict, even if not used by any modules (roughly doubling its size, but you usually need those types anyway)

Simpler approaches!
Drop *built-in modules*

- This simplest approach of all treats modules built in to the core kernel as if they were in the core kernel too (so types move around if you compile a module into the kernel).
- This has unfortunate implications for any types with conflicting definitions in those modules: combined with earlier simplifications this means all those types disappear!

Simpler approaches!

Put all the above simplifications together and I think you get exactly what you have now: a pile of BTF with one set of types per input module, dedupped against one big set for the object files included in vmlinux.[oa], except they're already dedupped BTF so pahole doesn't need to do any of that.

Format ossification?

- Binutils release cadence: 2/yr; dwarves release cadence: whenever needed, but this seems to be *less* often
- Backporting libctf changes should be easy, as long as you stay beyond the CTFv4 boundary (CTFv4 changes nearly every line). I will backport all relevant changes into the relevant (2.44?) release branch.
- Many distros release binutils from that branch anyway
- ld will automatically pick these up if it is dynamically linked against libctf (RHEL is not :( )
- libctf is GPLv3 right now but should be LGPL in the near futuref, so lld etc can use it too

What do we need from BTF?

- We need to be able to read BTF from more than one kernel version, because binutils is not part of the kernel
- So when BTF changes format or semantics in backwardly-incompatible ways, *please* bump the format version in the header.
- That’s all!

Obscure edge benefits of CTFv4

- The superset format may have things you can use in BTF for userspace or something like that (I can talk about them for hours and *hours* if you want details):
  - We can detect if you import the wrong parent dict (new header field)
  - We can represent static-scope variables (coming to BTF too!)
  - We can associate ELF symbols with types efficiently, including when sparse and in child dicts
  - We can encode *much* larger types (2^64 bytes, 2^32 vlen)

Far-out future possibilities

- Unlike older CTF versions, BTF can support *arbitrarily deep nesting* of split BTF: children can have *other* children of their own!
- So (with the addition of one field in the header to say which file a piece of BTF is a child of), we could have a *three-level tree*: core kernel, modules, then per-translation unit BTF with conflicting types in it!
- Clients would have to adapt, but it would mean we lose *no information at all* about the input type system!

Links and thank you

- More on the deduplicator: https://lpc.events/event/7/contributions/725/
- More on CTF in general (old, but not inaccurate): https://lwn.net/Articles/795384/
- Thank you for listening!