Objtool for arm64

Linux Plumbers Conference, 2021

Will Deacon <will@kernel.org>

android

Objtool

What is it?

- **A host program in `tools/objtool/` that runs on each `.o` file during kernel build**
  - Currently only supported/used by `x86_64`
  - Merged upstream in v4.6 (Feb 2016)

- **General binary linter and patching utility**
  - Can check and enforce invariants on the vmlinux
  - Helps to catch compiler and asm issues which would otherwise be missed

- **Relies on control flow reconstruction**
  - Can be sensitive to compiler optimisations
  - https://git.kernel.org/linus/3193c0836f20 disabled `-fgcse` for `___bpf_prog_run()`!

android

Objtool

What does it do for x86?

- **Generation of ORC unwinding data**
  - Lightweight alternative to DWARF; avoids needs for frame pointers (esp. in asm)

- **Binary validation of:**
  - Stack frames (relied upon for live-patching)
  - Unreachable instructions, retpoline, uaccess-enabled regions, ‘noinstr’ annotations

- **Binary modification**
  - Convert some `__sanitizer_cov*()` calls to NOPs
  - Generate mcount_loc section and convert `__fentry` calls to NOPs
  - Generate `.static_call_sites` section
  - Arch-specific branch patching (insertion of thunks etc).

android

“So I've started looking at what it would take to get live patching going on ARM64 :-)” -- Ben Herrenschmidt

Subject: [RFC PATCH v2 00/13] objtool: add base support for arm64 -- Julien Thierry

Objtool

Why do we need it for arm64?

- **We want reliable stack-tracing for same reasons as x86**
  - Primarily for kernel live-patching
  - But also useful for unwinding across asynchronous boundaries (e.g. exceptions)

- **Some of the x86 constraints do not apply:**
  - No retpoline or static call table
  - Frame pointers are cheap

- **If we enable objtool as *optional* binary linter then two things will inevitably happen:**
  - It will find kernel-specific issues in toolchain output...
  - … and developers will push to enable objtool's patching capabilities for arm64
  - We must treat failures to track control flow as objtool bugs not compiler bugs!
    - This is very challenging given the current design of objtool

- **How feasible is it to fix these issues in the toolchains instead?**
  - May not be considered bugs by the developers (likely kernel-specific requirements)
  - Both GCC and Clang are widely used for arm64 kernel builds

android

Control-flow analysis and `-fgcse`

```
if (cond_a) {
    took_a=1;
    ...
}

...

if (!took_a) {
    ...
}
```

“Currently objtool will consider the path ‘`cond_a && !took_a`’ and can get into trouble because of that.”

- No tracking of values or interpreter logic
- Do we really want to teach objtool about this?

android

Can the toolchain help us here?

ORC generation

Control flow analysis

Kernel-specific compilation flags

???