The limits of ABI stability in the kernel

Amelia Crate

INTRO

- Details about object re-use
- What parts of the ABI start breaking when you do this
- How those parts of the ABI could be more stable

DEFINITIONS

What is Object Re-use?

Re-use of a pre-compiled ELF object file in subsequent kernel builds.

Build Variations

Subsequent builds might differ in kernel config, kernel version, additional patches, or more.

WHY RE-USE OBJECTS?

- **For Chainguard, FIPS:** Certification can only be assigned to a binary.
- We do not want to certify the kernel binary, because then we cannot update it.

HOW TO RE-USE OBJECTS

- Enable **CONFIG_WERROR** and **CONFIG_OBJTOOL_WERROR**.
- Compile and link-time warnings are indicative of deeper issues.
- **The naive approach:** Mangle your Makefiles to skip compilation.

ABI STABILITY

- **6.6.1** to **6.6.2**:
  Success
- **6.6.1** to **6.7.1**:
  Failure
- Expect breakages about once per kernel release.

WHAT IS BREAKING?

Source vs Binary

Often recompiling the same source code works with no issues. Internal APIs* are actually quite stable.

*the function signature, at least

Function Interfaces

What is actually breaking is mostly function call interfaces.

BUILD OUTCOMES

Scenario	Typical Warnings / Errors
Undefined Symbols	WARN ld: vmlinux.o: in function `get_current': undefined reference to `const_pcpu_hot'
Unreachable Instructions	WARN vmlinux.o: warning: objtool: crypto_sha3_update+0x198: unreachable instruction
BTF ID Mismatches	WARN: multiple IDs found for 'task_struct': 113, 27133 - using 113

RUNTIME OUTCOMES

- **Boot or Page Fault**

- At runtime, this either works or it doesn't. You know right away.

RESOLVABLE BARRIERS

Toolchains

Pick a major version of your compiler and move on.

Compiled Modules

Split into code and modinfo. Solves BTF mismatch issues.

```
WARN: multiple IDs found for 'task_struct':
113, 27133 - using 113
```

MANAGING FUNCTION CALLS

1

The Problem
Function signature type changes break ABI while API remains compatible. Affects function prologues and stack setup.

2

The Fix: Shims
Call indirect with shims. Control the signature to keep the ABI of helper functions stable regardless of internal changes.

HIDDEN FUNCTION CALLS

- **Instrumentation:** KASAN, UBSAN, KCOV.
- Work via compiler instrumentation; subject to change within a major version.
- Can inject function calls into code with the same ABI problems.
- Solution: Disable these for pre-built objects.

THE BUILD-SYSTEM ABI

- **Stack protectors:** 80d47def: x86/stackprotector/64: Convert to normal per-CPU variable
- **ELF section names:** 8d9cc7f15: Rename .data.once to .data..once to fix resetting WARN*_ONCE

Low level changes inlined in every function prologue, epilogue, and object layout.

Could we have a stable API

- **Not without significant changes** to kernel development… but we could have a stable-ish base
- We could have an ABI which is stable enough for distros to build kernel packages with a stable ABI

Changes to support a stable-ish ABI

- **Change to patch acceptance policies to LTS kernels**
- Enforce restrictions on signature changes to EXPORT_SYMBOL()'d functions
- Refuse changes in low-level build system primitives

FIPS SECURITY BENEFITS

- Official pathway helps get a FIPS kernel without forgoing updates.
- Prevents pinning a single kernel forever and accumulating CVEs.
- Avoids making kernels a static target for attackers.
- Smaller, incremental updates are superior to big jumps.

Questions?

Amelia Crate