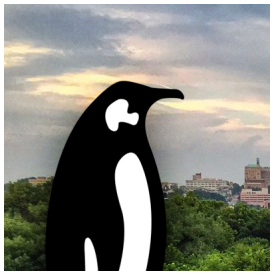

Contribution ID: **313** Type: **not specified**

Compiling for verified targets (BPF)

Monday, 13 November 2023 14:30 (50 minutes)

During the GNU Tools Cauldron conference we had an activity called “The challenges of compiling for verified targets”, with this abstract:

> The Linux kernel BPF is not a sandbox technology: BPF programs do not rely on a sandboxed environment for security, and in fact they are executed in the most privileged kernel mode. Instead, BPF programs are known to be safe before they are executed. It is the job of the kernel verifier to determine whether a given BPF program is safe to be executed, and to reject it if that cannot be determined. Conceptually speaking, an entire BPF program should be as predictable as a single machine instruction. Obviously, this cannot be achieved for any arbitrary BPF program given that the BPF ISA is turing-complete, and so the verifier imposes quite draconian restrictions on the programs to make sure they always terminate, among other things.
>
> BPF programs are sometimes written by hand, but as more kernel subsystems are being expanded to use BPF, the programs are getting bigger and more complicated, and hackers prefer to write BPF programs in high level languages like C or Rust and compile them to BPF object code using an optimizing compiler. Both the GNU Toolchain and clang/llvm provide BPF support.
>
> In ordinary targets the main challenge of the compiler is to generate the optimal[1] machine instructions that implement the same semantics than the program being compiled. In verified targets (like BPF) there is an additional and very important challenge: the generated machine instructions shall be verifiable. While this cannot be guaranteed for every input program, ideally the optimizing compiler shall inform the user if the input program contains source language constructions that inexorably would lead to not-verifiable code, and shall also adjust the optimization passes in order to avoid transformation that lead to non-verifiable code. The better the compiler does this, the more practical compiled BPF will become.
>
> It is not clear how to achieve this. In this talk, we will first state the problem and then examine different alternatives and potential techniques and strategies, some of them already tried by the clang/llvm BPF port with variable success: IR legalization, usage of the static analyzer, verification in assembler, usage of counter-passes vs. pass tailoring (-Overifiable), usage of annotations in source code, tailoring of the front-ends (BPF C), etc. Also we will analyze and discuss the impact that each strategy would have to the rest of the compiler.
>
> Note that this problem is not specific to the GNU Toolchain. Whatever techniques get developed will also serve to the clang/llvm compiler. We will be touching base with them during the LPC conference in November this very year.
>
> [1] Given some criteria like execution speed, or compactness.

The goal of that activity was to gather input and ideas from the GNU toolchain community on the best way to proceed in order to fulfill the very novel needs of verified targets in general, and BPF in particular. We had an interesting and useful discussion in Cauldron.

As a next step, we intend to continue the discussions at LPC with the BPF kernel people and also clang/llvm maintainers. We hope to start developing strategies and techniques to make compilation for verified targets useful in practice, and to keep it that way in a future where proliferation of verifiers is expected to happen.

Primary authors: MARCHESI, Jose E. (GNU Project, Oracle Inc.); SONG, Yonghong

Presenters: MARCHESI, Jose E. (GNU Project, Oracle Inc.); SONG, Yonghong

Session Classification: Toolchains

Track Classification: Toolchains Track