Flexible and Concise Spectre Mitigations for BPF

Luis Gerhorst
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
Ruhr-Universität Bochum (RUB)
gerhorst@cs.fau.de

Henriette Hofmeier
Ruhr-Universität Bochum (RUB)
henriette.hofmeier@rub.de

Timo Höning
Ruhr-Universität Bochum (RUB)
timo.hoenig@rub.de

1 MOTIVATION
Operating systems rely on system calls to allow the controlled communication of isolated processes with the kernel and other processes. Every system call includes a processor mode switch from the unprivileged user mode to the privileged kernel mode. Although processor mode switches are the essential isolation mechanism to guarantee the system’s integrity, they induce direct and indirect performance costs as they invalidate parts of the processor state. In recent years, high-performance networks and storage hardware has made the user/kernel transition overhead the bottleneck for IO-heavy applications. To make matters worse, security vulnerabilities in modern processors (e.g., Meltdown) have prompted kernel mitigations that further increase the transition overhead.

2 BACKGROUND
Linux’s extended Berkeley Packet Filter (BPF) allow unprivileged user processes to load safety-checked bytecode into the kernel. The code is just-in-time compiled and executes at near-native speed. Invoking BPF programs in the kernel and calling kernel functions from within BPF is much faster than the respective switch to/from user context [4].

3 PROBLEM STATEMENT
To isolate the BPF programs from the kernel, the bytecode has to be statically verified for memory- and type-safety. Initially, only BPF program paths that could actually execute architecturally were statically verified for memory- and type-safety. Initially, only BPF program paths that could actually execute architecturally were statically verified for memory- and type-safety. Initially, only BPF program paths that could actually execute architecturally were statically verified for memory- and type-safety.

4 APPROACH
In this talk, we show that there is significant potential to improve the performance and expressiveness of unprivileged BPF programs in order make them safe\(^1\) even in the face of the Spectre attacks.

\(^1\)Not considering programming errors such as [5].

REFERENCES