Flexible and Concise Spectre Mitigations for BPF

Luis Gerhorst, Henriette Hofmeier, Timo Hönig
Motivation: High-Performance IO

• **Problem:** User-/kernel switching overhead too high for packet processing, NVME disks, tracing, ...

• **Approaches:** System-call batching (e.g. io_uring, aio), kernel-bypass (e.g. DPDK), **software-based isolation** (BPF)
• Un-/privileged users load bytecode into the kernel
• Verified for type-/memory-safety and a bounded execution time
• JIT-compiled and invoked in kernel mode
• BPF program can call kernel helpers (≈ system calls)

• **Problem:** Expressiveness and performance are limited by mitigations against speculative side-channel attacks
Speculative Side-Channel Attacks

- "Hardware bugs" not considered: Meltdown, load-value injection
- Software-based mitigation: Bounds-check bypass, speculative-store bypass, speculative type-confusion
- Non cache-based side-channels
- Secrets are encoded into side-channels on speculative paths
• Memory-safety: Only access borrowed/owned memory
• Type-safety: Only perform operations valid for the type (pointer/scalar/…)
• Pointers are secrets: Unprivileged programs can not cast pointers to scalars or encode them into side-channel
BPF Spectre Mitigations

- Speculative Store-Bypass (v4) → Fences
- Speculative Bounds-Check Bypass (v1) → Reject / Masking
- Speculative Type Confusion (v1) → Reject

- Evaluation: Collected over 350 programs from 4 projects and analyzed the number of fences and rejections
BPF Spectre Mitigations Limitations

- **Unprivileged**: Hardcoded policy (no speculative breakout with speculative constant-time for pointers) → Limited expressiveness and performance

- **Privileged**: Only some mitigations active → Easily introduce vulnerabilities

- **Privileged and unprivileged**: Secrets unknown to compiler completely unprotected

- **Approaches**: Refine kernel implementation or create an extensible architecture
Approach: Refine Kernel Mitigations

• Replace „no speculative breakout“ with „relative constant-time“ policy
• Improves expressiveness
• Makes the verifier more complex (currently already 13k SLoC)
Approach: Extensible Mitigations

• Introduce BPF instructions to prevent/restrict speculation
• Exposes speculation in Userspace ABI
• Privileged userspace services: Apply concise mitigations to unprivileged programs
• Compilers and programmers: Precisely control mitigations for privileged programs
Summary

• BPF is the only production-ready system for software-fault isolation that fully mitigates Spectre

• Speculative bounds-check bypass and type-confusion mitigations limit expressiveness while speculative store-bypass limits performance

• We will attempt to refine the current mitigation-approach, and create an architecture that allows for flexible and concise user-defined mitigations
Appendix
Speculation Policies

Security depends on system context and hardware

• **Leakage model:** Which instructions (e.g. load) leak which information (e.g. data address)?

• **Attacker model:** None, only remote, local unprivileged users

• **Leakage + attacker model → speculation policy:** No speculation, no speculative breakout, speculative constant-time, relative constant-time, ..., *no Spectre*, arbitrary speculation
Limited Performance

Difference measureable, real-world programs WIP
Limited Expressiveness

Even for small example programs: Many can not be mitigated