Mitigating Spectre-PHT using Speculation Barriers in Linux eBPF

LPC'24 — September 20, 2024

Luis Gerhorst[1], Henriette Herzog[2], Peter Wägemann[1], Maximilian Ott[1], Rüdiger Kapitza[1], Timo Hönig[2]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[2] Ruhr-Universität Bochum, Germany

Friedrich-Alexander-Universität
Faculty of Engineering

RUHR
UNIVERSITÄT
BOCHUM

Motivation: Unprivileged eBPF

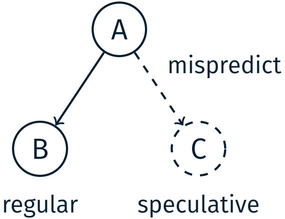
- Unprivileged applications
  - Network Traffic Filters
  - `io_uring`
  - Seccomp-eBPF
- Limited expressiveness
  - eBPF enables Spectre-attacks
  - Mitigations reject programs
- **VeriFence** enables more eBPF-based applications

eBPF’s Transient Execution Attack Mitigations

Spectre Attacks & Mitigations

- **Branch Target Buffer (v2):** `retpoline`

Program States with Spectre-PHT:

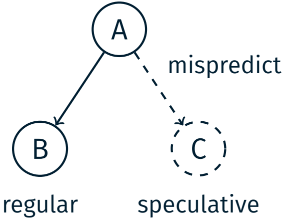


eBPF’s Transient Execution Attack Mitigations

Spectre Attacks & Mitigations

- **Branch Target Buffer (v2):** `retpoline`
- **Store to Load (v4):** Speculation barriers

Program States with Spectre-PHT:

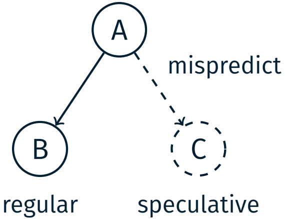


eBPF's Transient Execution Attack Mitigations

Spectre Attacks & Mitigations

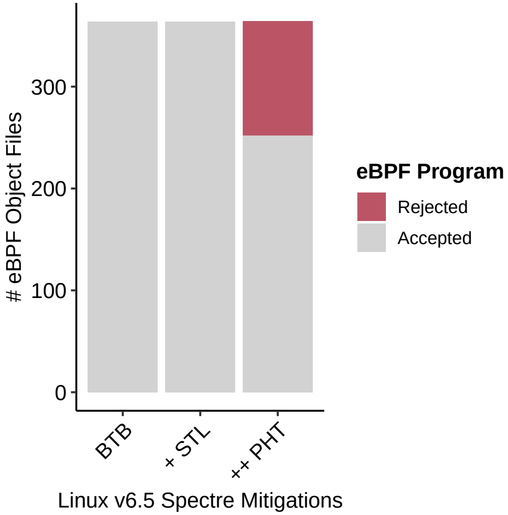
- **Branch Target Buffer (v2):** `retpoline`
- **Store to Load (v4):** Speculation barriers
- **Pattern History Table (v1):**
  - Index masking
  - Simulate speculative paths → rejection

Program States with Spectre-PHT:



Problem Statement

- Collect eBPF object files from open-source projects
  - 50 applications
  - 314 tests/examples
- **31% are rejected** because of Spectre-PHT mitigations



Agenda

Linux eBPF's Spectre Defenses

VeriFence: Fence or Verify

Evaluation with BCC, Parca, and Loxilb

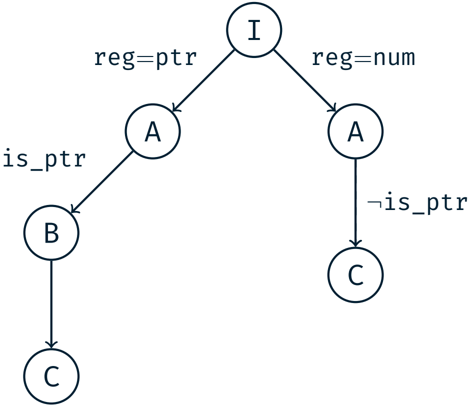
Optimizations and Discussion

Verification

—— regular

---- speculative

```
I: reg = is_ptr ? ptr : num
A: if (!is_ptr) goto C
B: value = *reg
   covert_channel[value]
C: exit()
```

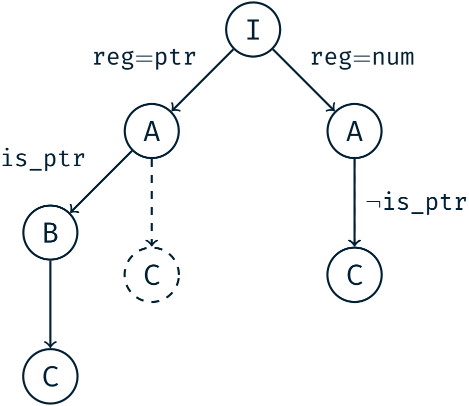


Verification with Spectre-PHT

—— regular

---- speculative

```
I: reg = is_ptr ? ptr : num
A: if (!is_ptr) goto C
B: value = *reg
   covert_channel[value]
C: exit()
```

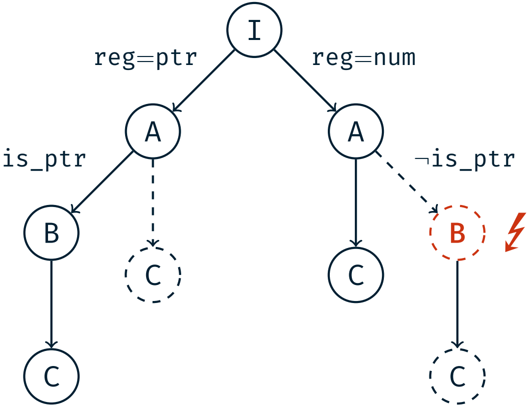


Verification with Spectre-PHT

—— regular

---- speculative

```
I: reg = is_ptr ? ptr : num
A: if (!is_ptr) goto C
B: value = *reg
   covert_channel[value]
C: exit()
```

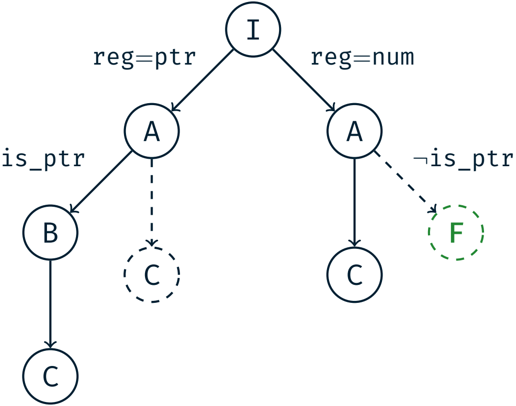


Verification with Spectre-PHT: VeriFence

—— regular

---- speculative

```
I: reg = is_ptr ? ptr : num
A: if (!is_ptr) goto C
F: lfence
B: value = *reg
   covert_channel[value]
C: exit()
```



Implementation

- Separate barriers for Spectre-PHT and -STL
  - `nospec_v4`
  - `nospec_v1`
- Catch speculative verification errors
  - Refactor code to allow easy *catching*
  - Insert barriers
- Treat existing barriers as exits

Agenda

Linux eBPF’s Spectre Defenses

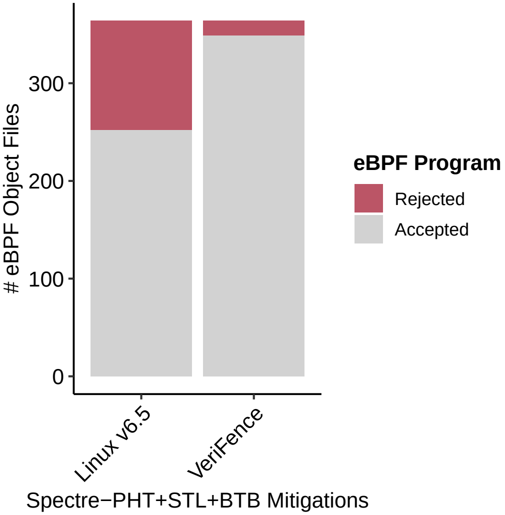
VeriFence: Fence or Verify

Evaluation with BCC, Parca, and Loxilb

Optimizations and Discussion

Rejections

- 15 **test programs** from the Linux selftests are still rejected
- All solveable
  - Reduce complexity
  - Remove unsupported variable stack accesses

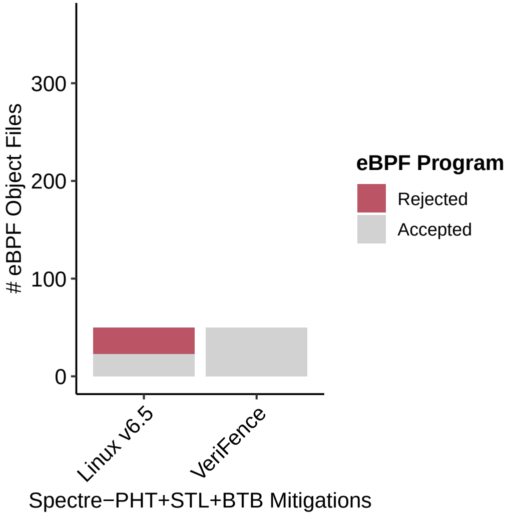


Rejections for Application Programs

Application Programs

- Cilium, Linux selftests selection
- BCC, Parca, Loxilb

All 50 accepted with VeriFence



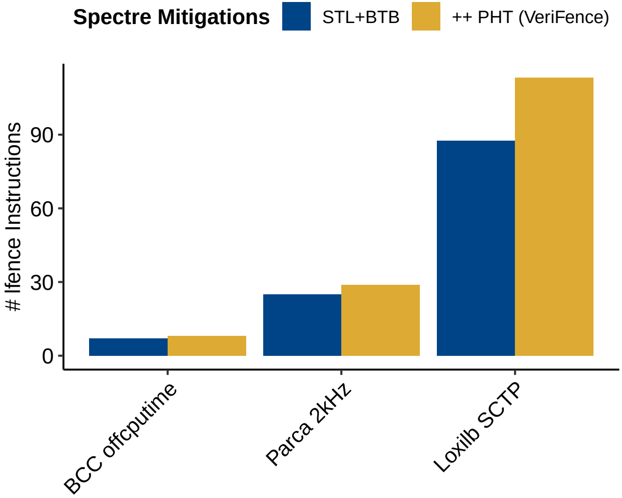
Number of Speculation Barriers

Overall

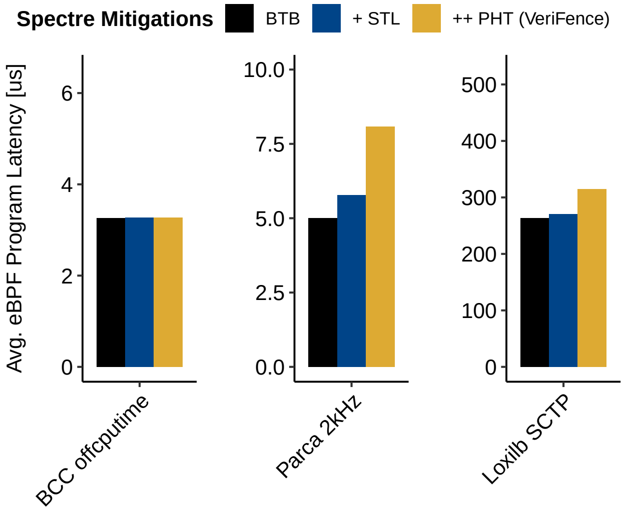
- Analyze number of barriers for 364 programs
- 1.0% `lfence` instructions with Spectre-STL
- 1.8% with Spectre-PHT using VeriFence

Plot

Applications with highest macrobenchmark overhead

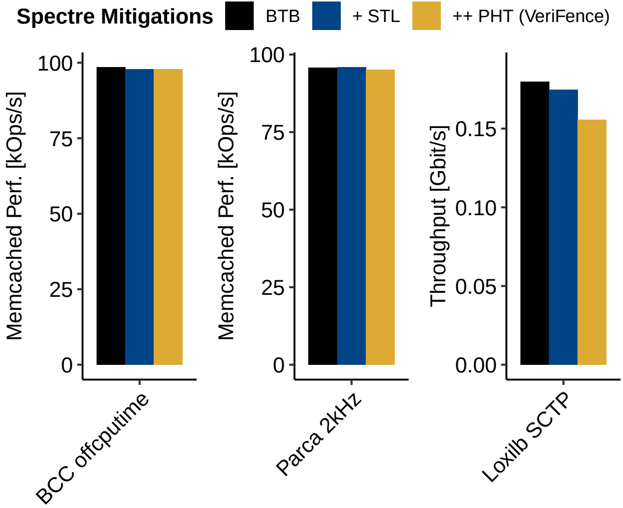


eBPF Execution Time



- Spectre-PHT barriers have higher impact
- Loxilb SCTP takes hundreds of microseconds

Impact on Application Performance



- Mostly insignificant
  - BCC Tracers
  - Parca
  - Loxilb TCP
- 14% for Loxilb SCTP

Agenda

Linux eBPF’s Spectre Defenses

VeriFence: Fence or Verify

Evaluation with BCC, Parca, and Loxilb

Optimizations and Discussion

Potential for Further Optimization

- Optimize number of barriers
  - Only one PHT-barrier per basic block
  - Apply *Fence or Verify* to Spectre-STL

Potential for Further Optimization

- Optimize number of barriers
  - Only one PHT-barrier per basic block
  - Apply *Fence or Verify* to Spectre-STL
- Poison speculation instead of using a barrier

Potential for Further Optimization

- Optimize number of barriers
  - Only one PHT-barrier per basic block
  - Apply *Fence or Verify* to Spectre-STL
- Poison speculation instead of using a barrier
- Be less strict during speculation
  - Unsafe speculation
    $\subset$ Unsafe **architectural** behavior
  - E.g., allow NULL-pointer dereferences

Towards Unprivileged eBPF?

- Verifier bugs remain an issue
  - Apply formal methods to the verifier
  - Sandbox
    - Memory layout is a challenge
    - MPKs should still work under speculation
  - Trusted compilation from `safe` Rust?
- Spectre gadgets by-mistake in priviledged eBPF?

Summary

- VeriFence
  - Reuses architectural verification
  - Only fences off unsafe speculative behavior
- All real-world programs are accepted
- Overhead
  - 0% to 62% overhead for eBPF execution
  - Lightweight invocation
- `https://sys.cs.fau.de/verifence`
- *Questions?*

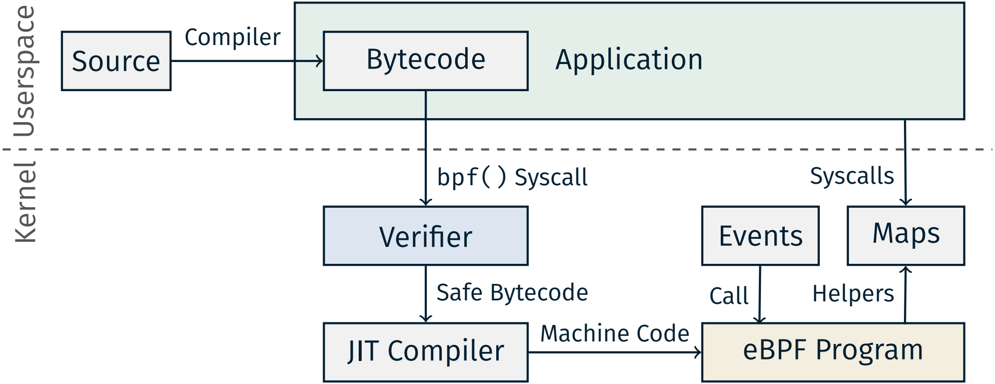
SPECTRE

DFG Deutsche Forschungsgemeinschaft

Appendix

1. eBPF Overview
2. Rejections per Project
3. Performance Data
4. Percentage of Speculation Barriers for 844 Programs
5. References
6. Attributions

eBPF Overview



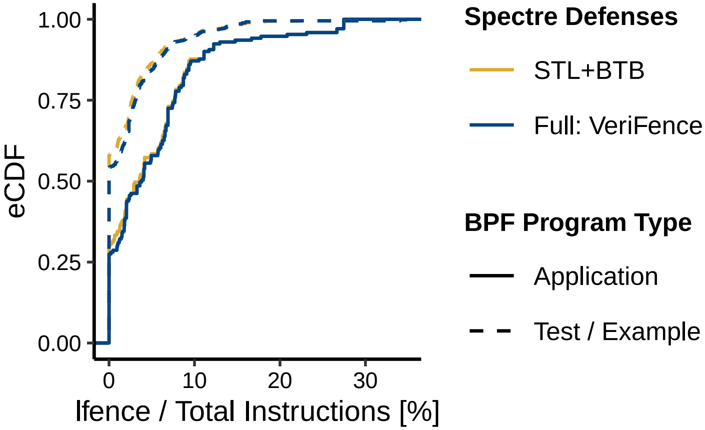
Rejections per Project

Project	# Programs	# Files	# Files Rejected
Linux Selftests	592	275	80
BCC	133	39	19
Linux Samples	71	32	5
Loxilb	19	4	3
Cilium	10	1	0
libbpf Examples	10	7	1
Parca	7	4	3
Prevail	2	2	1

Performance Data

- 43 BCC Tracers
- Parca Stack Sampling Profiler (20Hz - 2kHz)
- Loxilb Network Load Balancer
    - TCP and SCTP Throughput (iperf3)
    - TCP CRR and RR (netperf)
    - HTTP Tail Latency (wrk2)

Percentage of Speculation Barriers for 844 Programs



References 1/3

- VeriFence: Lightweight and Precise Spectre Defenses for Untrusted Linux Kernel Extensions — `https://arxiv.org/abs/2405.00078`
- `io_uring`: BPF controlled I/O — `https://lpc.events/event/11/contributions/901/`
- Programmable System Call Security with eBPF — `https://arxiv.org/abs/2302.10366`

References 2/3

- Techniques to poison speculation instead of using a barrier (similar to SLH):
  - bpf: prevent out of bounds speculation on pointer arithmetic — `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=979d63d5`
  - Secure automatic bounds checking: prevention is simpler than cure — `https://dl.acm.org/doi/10.1145/3368826.3377921`
  - You Shall Not Bypass: Employing data dependencies to prevent Bounds Check Bypass — `https://arxiv.org/abs/1805.08506v3`

References 3/3

- Sandboxes for eBPF
  - BeeBox: Hardening BPF against Transient Execution Attacks — `https://cs.brown.edu/~vpk/papers/beebox.sec24.pdf`
  - MOAT: Towards Safe BPF Kernel Extension — `https://www.usenix.org/conference/usenixsecurity24/presentation/lu-hongyi`
  - Unleashing Unprivileged eBPF Potential with Dynamic Sandboxing — `https://dl.acm.org/doi/10.1145/3609021.3609301`
  - Improving eBPF Complexity with a Hardware-backed Isolation Environment – `https://lpc.events/event/18/contributions/1947/`

Attributions

- “a yellow construction vehicle” by Jon Sailer — `https://unsplash.com/photos/a-yellow-construction-vehicle-4YjxxjiLKag`