Performance evaluation of the Linux kernel eBPF verifier

Maxime Derri
INRIA

Julia Lawall
INRIA

Kahina Lazri
Orange

January 31, 2025

Outline

1. Evolutions of the eBPF verifier
   a. What is the eBPF verifier?
   b. Performance evaluation
2. Comparion of the eBPF verifier and PREVAIL
   a. Design comparison
   b. Performance evaluation
3. Conclusion
4. Appendix

The eBPF verifier

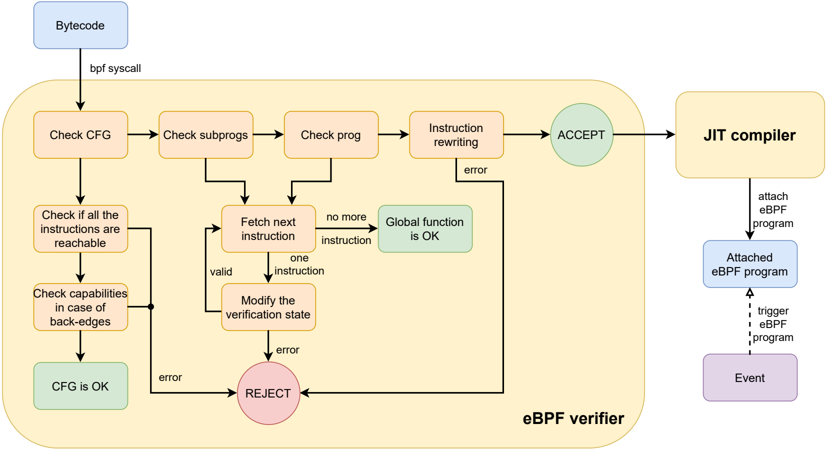


Figure: Overview of the Linux eBPF verifier

The eBPF program verification is complex

The verifier is getting complex

- New features
- Possible bugs
- Hard to assure the safety
- False positives

The eBPF program verification is complex

The verifier is getting complex

- New features
- Possible bugs
- Hard to assure the safety
- False positives

eBPF is an active research field

- Finding bugs dynamically (e.g. Syzkaller, Buzzer, SEV, BVF)
- Finding bugs statically (e.g. Agni)
- Isolation (e.g. BeeBox, MOAT, SafeBPF)
- Other verifiers (e.g. PREVAIL)

Observe the eBPF verifier evolution

- Hard to understand how the eBPF verifier works
- Hard to track all the modifications made on the verifier
- Observe how the eBPF verifier behave:
  - Verification time
  - Memory footprint
  - Program rejection

Does the eBPF verifier evolution impact performance?

Setup

- Ubuntu 18.04 (kernel 4.19) to 24.04 (kernel 6.10)
- QEMU-KVM VMs
- The PLDI paper version of PREVAIL is used as a loader for the eBPF verifier
- Memory footprint is obtained using mm_page_alloc and mm_page_free tracepoints

Samples

- 192 pre-compiled eBPF programs from the PREVAIL repository
- Only 144 of the 192 programs are used (Linux kernel, prototype-kernel, cilium_test, Cilium, Open vSwitch and Suricata)
- The remaining 48 programs are not used as program types can't be deduced from custom sections by general loaders

Verification time

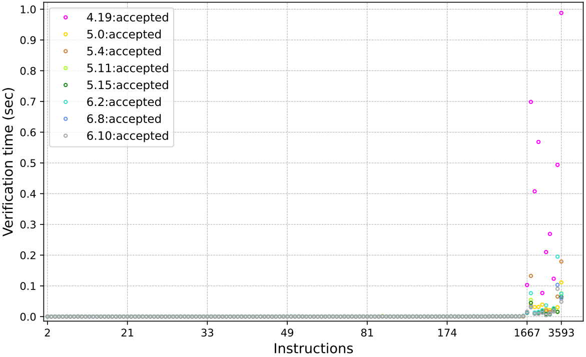


Figure: Average time (sec) vs instructions

- Since the 5.0 kernel version, verifiers are faster due to branch management improvements

Memory footprint

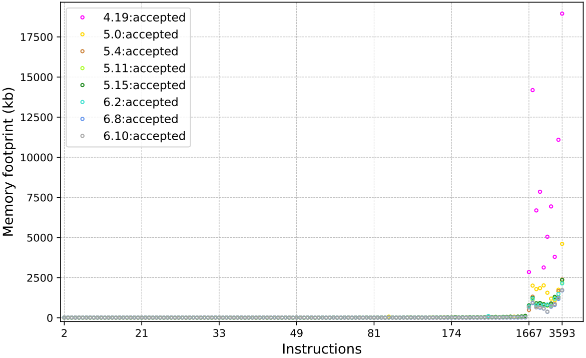


Figure: Maximal memory footprint (kb) vs instructions

- Since the 5.0 kernel version, verifiers consume less memory due to branch management improvements

Programs accepted

	4.19	5.0	5.4	5.11	5.15	6.2	6.8	6.10
Accepted programs (/144)	139	141	142	141	141	143	143	143
Complexity limit reached	X							
Helper not yet available	X	X						
Bad kernel configuration	X							
Bad kernel version argument	X							
Map pre-allocation disabled (perf_event program type)	X	X	X	X	X			
Lockdown mode				X	X	X	X	X

Table: Programs accepted and causes of rejection per eBPF verifier version

Note

– Some Linux distributions disable the bpf syscall (e.g. Ubuntu 19.04) or add lockdown checks (e.g. Ubuntu 21.04)

Programs accepted

4.19	5.0	5.4	5.11	5.15	6.2	6.8	6.10
143/144	144/144	144/144	144/144	144/144	144/144	144/144	144/144

Table: Accepted programs per eBPF verifier version

The verifier is not always the culprit!

- Solving the issues results in programs being accepted (except one program where the complexity limit is reached)

Outline

Design principles of the eBPF verifier and PREVAIL

	eBPF verifier	PREVAIL
Location	Kernel-space	User-space
Abstract domains	Interval: [a,b] tnum: (value, mask)	Zone: (X - Y) $\leq$ c
Paths	State pruning	Join operator
Loops	Loop according to the condition State pruning	Fixpoint computation Join operator Weak topological ordering Widening and narrowing operators
Termination	Max number of instructions Restrictions on loop usage Infinite loop detection	Max number of instructions Widening and narrowing operators
Soundness	Bugs were found Some bugs might stay Hard to prove the verifier entirely	Should be sound by design Not a lot of work on PREVAIL

Table: Differences between the eBPF verifier and PREVAIL

eBPF verifier vs PREVAIL

- Observe the evolution of the verification performance of two verifier versions since 2019
- Observe how they perform with the set of eBPF objects provided in the PREVAIL paper

Performance comparison

Setup

- Ubuntu 18.04 (kernel 4.19) and 24.04 (kernel 6.10)
- QEMU-KVM VMs
- The PLDI paper version of PREVAIL is used as a loader for the eBPF verifier
- The PLDI paper version of PREVAIL is used with the kernel 4.19
- The latest version of PREVAIL is used with the kernel 6.10
- Memory footprint is obtained from /proc/$pid/status, VmHWM

Verification time of the eBPF verifier vs PREVAIL

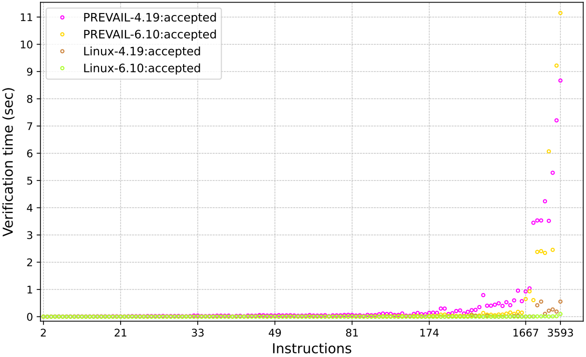


Figure: Average time (sec) vs instructions

- The PREVAIL verification time increases with the number of instructions

Memory footprint of the eBPF verifier vs PREVAIL

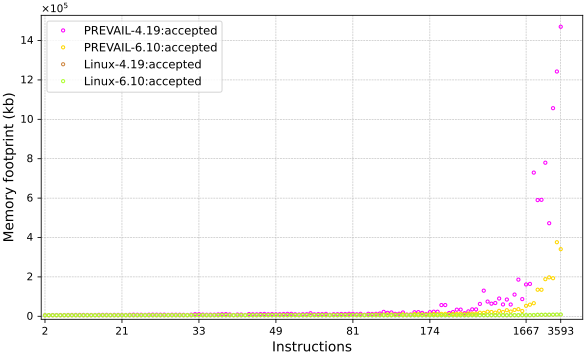


Figure: Maximal memory footprint (kb) vs instructions

- The PREVAIL memory footprint increases with the number of instructions

Programs accepted by the eBPF verifier vs PREVAIL

	4.19	6.10
Linux	143/144	144/144
PREVAIL	143/144	140/144

Table: Accepted programs per verifier[1]

⟶	PREVAIL 4.19	PREVAIL 6.10	Linux 4.19	Linux 6.10
PREVAIL 4.19		1	1	1
PREVAIL 6.10	4		3	4
Linux 4.19	1	1		1
Linux 6.10	0	0	0	

Table: Programs rejected from a verifier[1] which are accepted by another verifier[1]

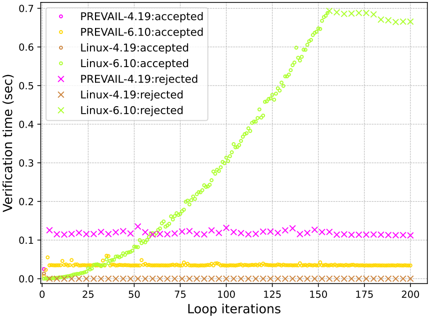
[1] PREVAIL+4.19 = PLDI version and PREVAIL+6.10 = Latest version

Performance comparison with many paths

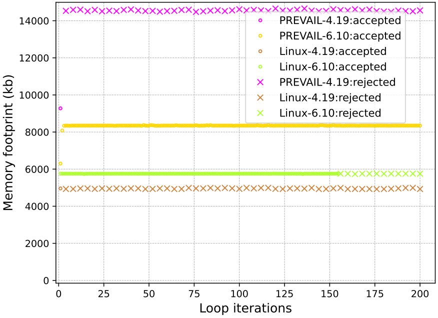
Samples

- double_strcmp template from the PREVAIL repository
- Performs two manual strcmp (two non-unrolled loops with if/else)
- The number of loop iterations is configurable
- Samples are compiled with Clang-8

Comparison on loops



(a) Average time (sec) vs iterations



(b) Maximal memory footprint (kb) vs iterations

Figure: Verification of programs built from the double_strcmp template with an increasing number of loop iterations

- PREVAIL is faster as it reached a fixpoint

Outline

Conclusion

- The performance of the eBPF verifier has improved
- No regression has been found for the eBPF verifier
- The program rejection is not limited to the eBPF verifier
- PREVAIL proposes an interesting approach to bounded loop verification
- The memory consumption of PREVAIL has decreased
- It seems that there is a regression with the latest version of PREVAIL (requires further investigations)

Outline

A bit more on branches and pruning

Setup

- Ubuntu 18.04 (kernel 4.19), 19.04 (kernel 5.0), 20.04 (kernel 5.4)
- QEMU-KVM VMs
- The PLDI paper version of PREVAIL is used as a loader for the eBPF verifier

Samples 1

- 144/192 pre-compiled eBPF programs from the PREVAIL repository

Samples 2

- double_strcmp template
- Loops are unrolled
- Samples are compiled with Clang-8

Changes in branches and pruning (1)

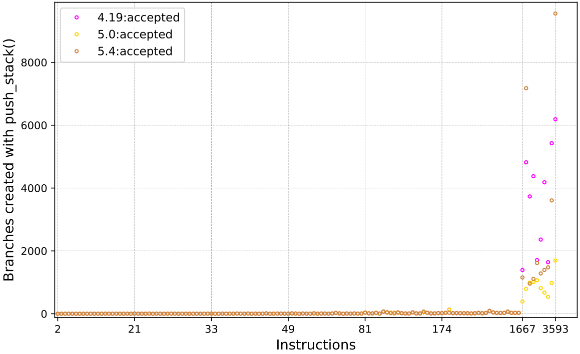


Figure: Number of branches created with push_stack() vs instructions

- The eBPF verifier in kernel 4.19 creates more branches than the one in 5.0

Changes in branches and pruning (2)

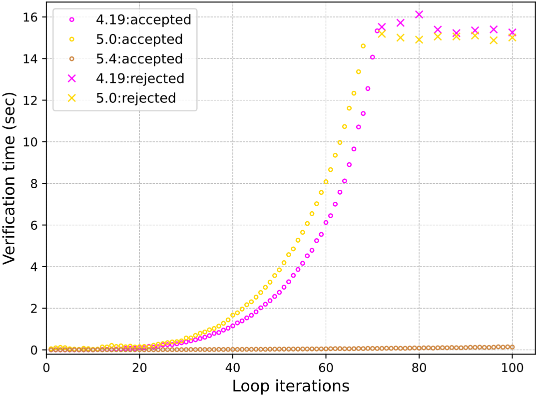


Figure: Average time (sec) vs iterations

- There are major changes between kernels 5.0 and 5.4 (precisely, in the kernel 5.3)

Impact of compilers and instruction sets

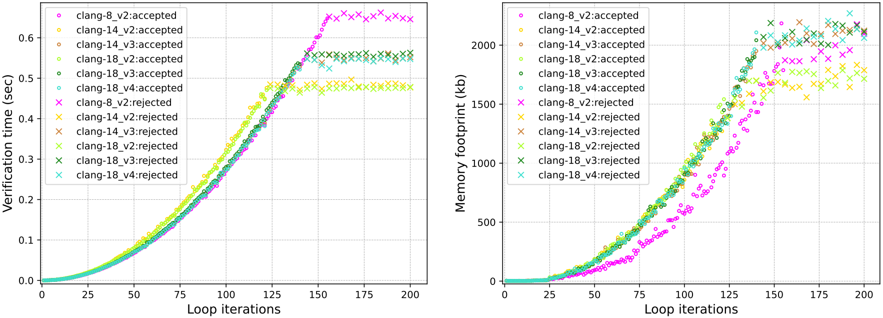
Setup

- Ubuntu 22.04.4 (kernel 6.8)
- Physical machine
- The PLDI paper version of PREVAIL is used as a loader for the eBPF verifier
- Memory footprint is obtained using mm_page_alloc and mm_page_free tracepoints

Samples

- double_strcmp template
- Samples are compiled with 3 versions of Clang (8, 14, 18) and 3 instruction sets (v2, v3, v4)

Impact on loops



(a) Average time (sec) vs iterations

(b) Maximal memory footprint (kb) vs iterations

Figure: Verification of programs built from the double_strcmp template with an increasing number of loop iterations

- More loop iterations are verified when programs are compiled with Clang-8

Impact on loops

	for $i = 1; i \leq 200; ++i$		
	$i = 1$	$i = 2$	$i \geq 3$
Clang-8 + v2	31	55	55
Clang-14 + v2	21 (-32.26%)	68 (+23.64%)	60 (+9.09%)
Clang-14 + v3	21 (-32.26%)	64 (+16.36%)	56 (+1.82%)
Clang-18 + v2	20 (-35.48%)	68 (+23.64%)	60 (+9.09%)
Clang-18 + v3	20 (-35.48%)	64 (+16.36%)	56 (+1.82%)
Clang-18 + v4	20 (-35.48%)	64 (+16.36%)	56 (+1.82%)

Table: Number of instructions of programs built from the double_strcmp template