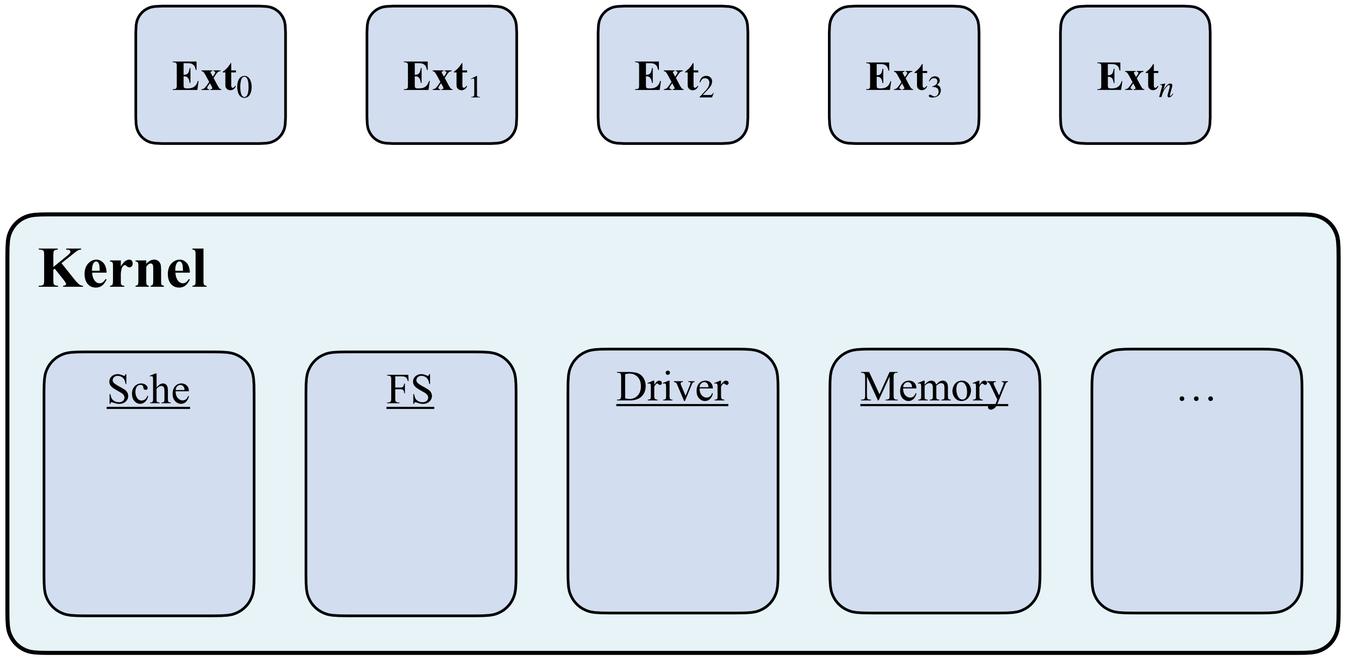
Prove It to the Kernel: Precise Extension Analysis via Proof-Guided Abstraction Refinement

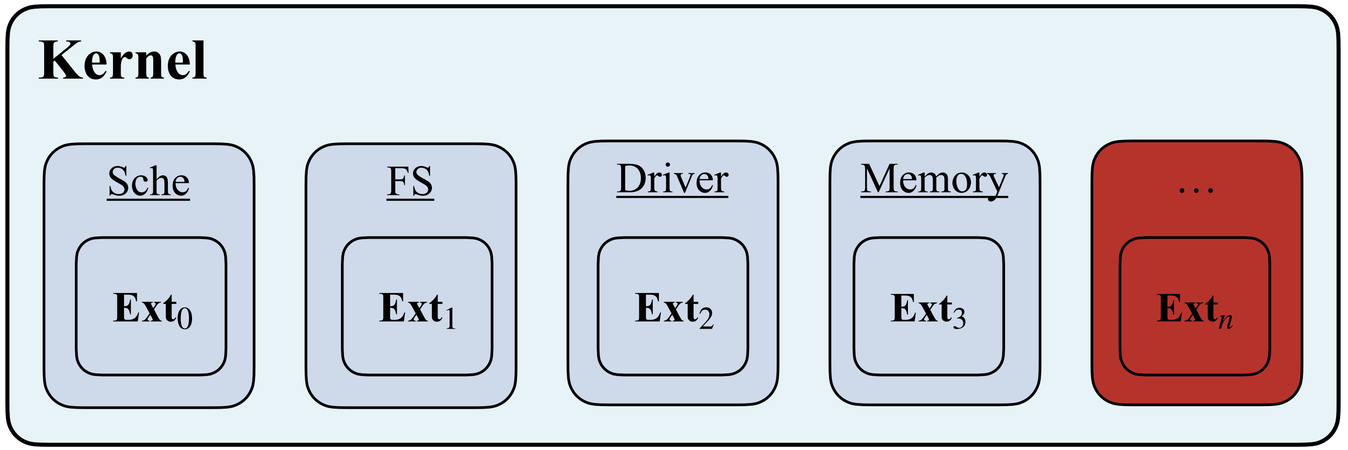
Hao Sun, Zhendong Su
ETH Zurich

ETH zürich

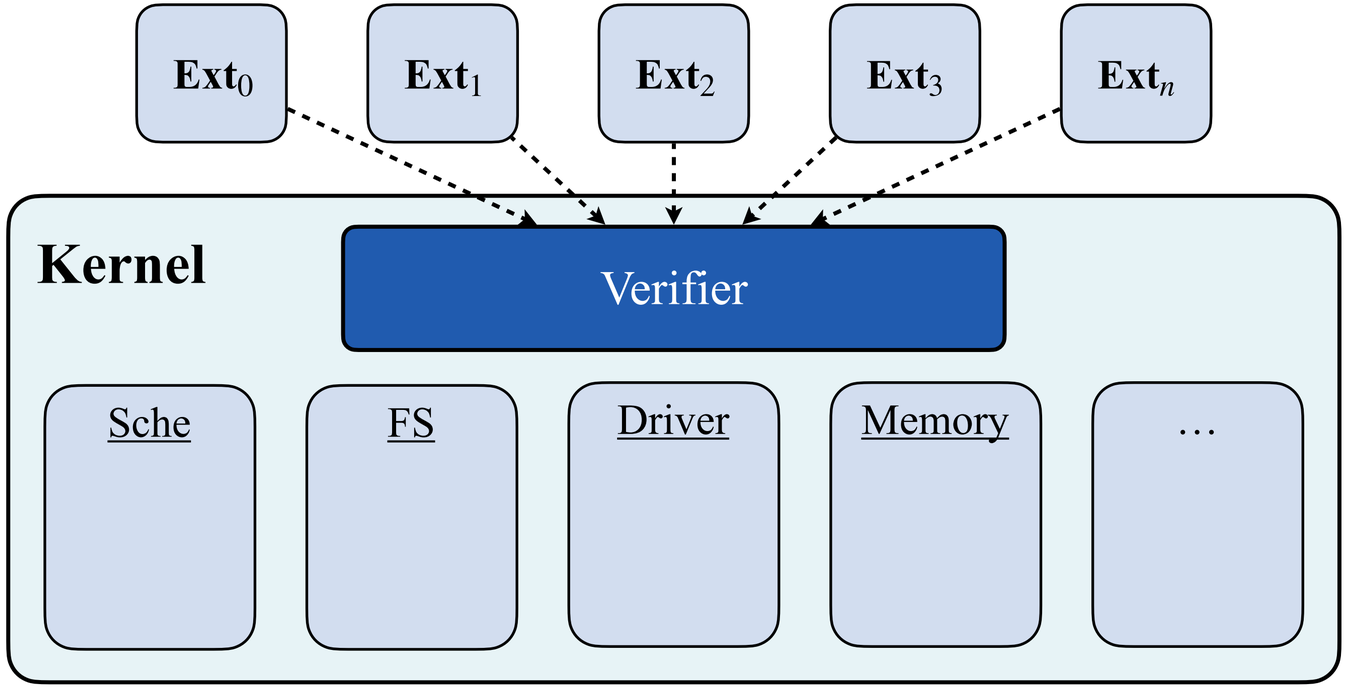
Extensible Kernel



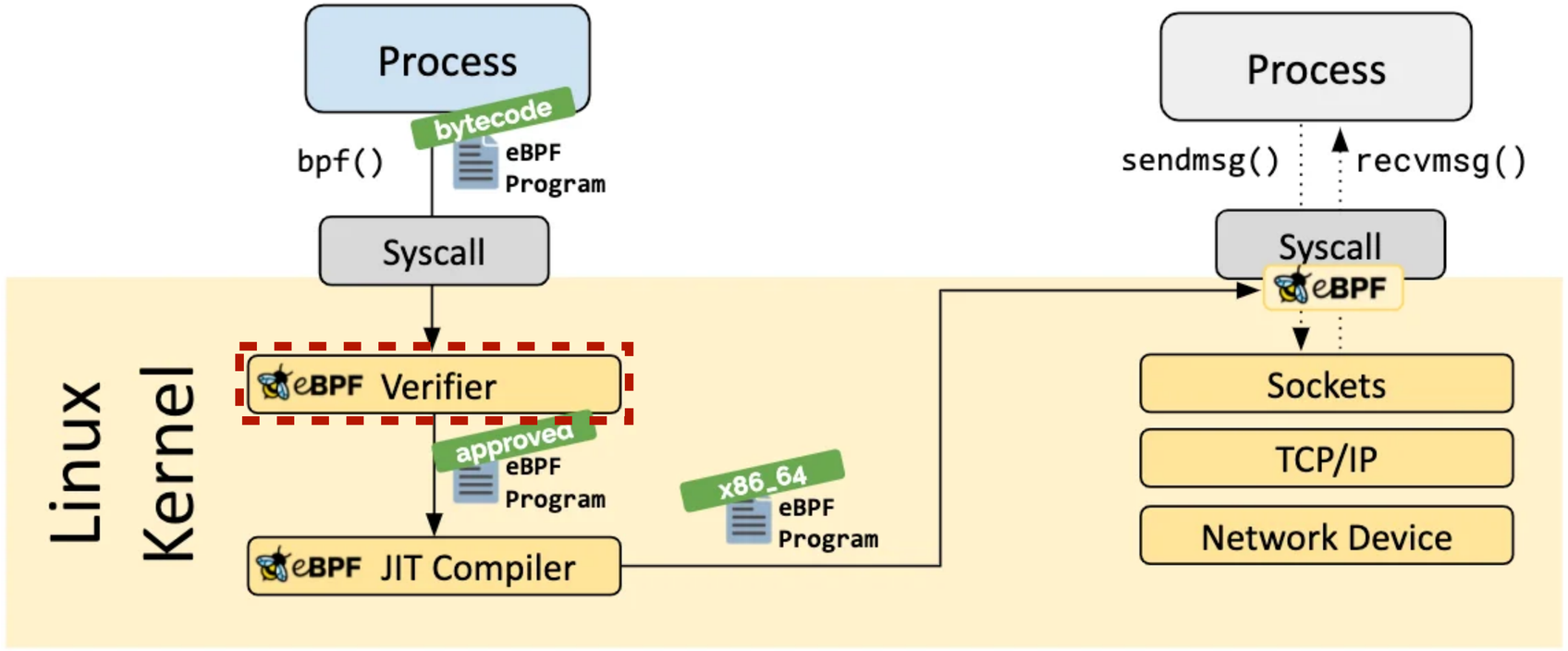
Extension Safety



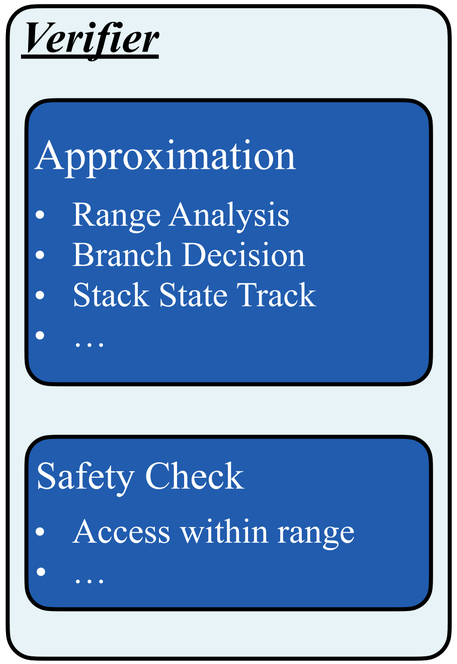
In-Kernel Extension Analysis



eBPF


Process
bytecode
bpf()
eBPF
Program
Syscall
Linux
Kernel
eBPF Verifier
approved
eBPF
Program
eBPF JIT Compiler
x86_64
eBPF
Program
Process
sendmsg()
recvmsg()
Syscall
eBPF
Sockets
TCP/IP
Network Device

eBPF Verifier

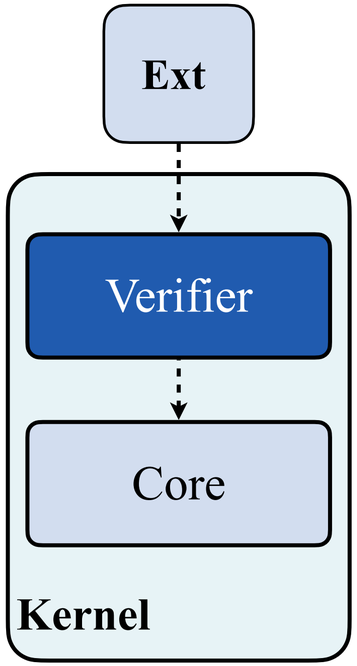


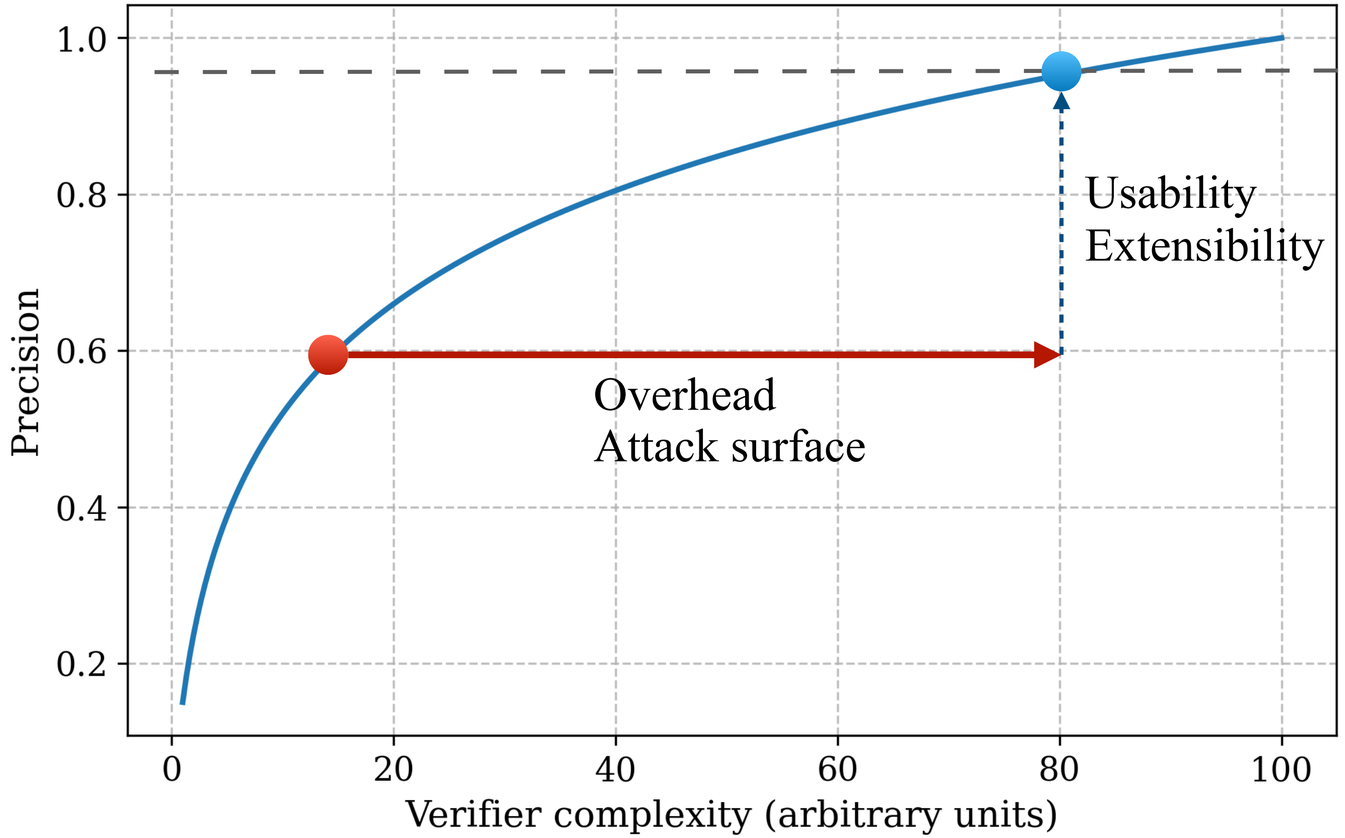

```
0: r1 = map_lookup(...) // r1 = ptr(size=16)
1: r2 &= 0xf            // r2 ∈ [0,15]
2: r2 <<= 1             // r2 ∈ [0,30]
3: r1 += r2             // r1.off ∈ [0,30]
4: r0 = *(u8*)r1        // unsafe: off+access_sz > mem_sz
```

Listing 1. Correct rejection of unsafe extension. The comments illustrate the verifier's analysis.

- Approximate the possible value set of each variable
- Check operation safety against the approximation
- Load the extension only if it passes the verifier

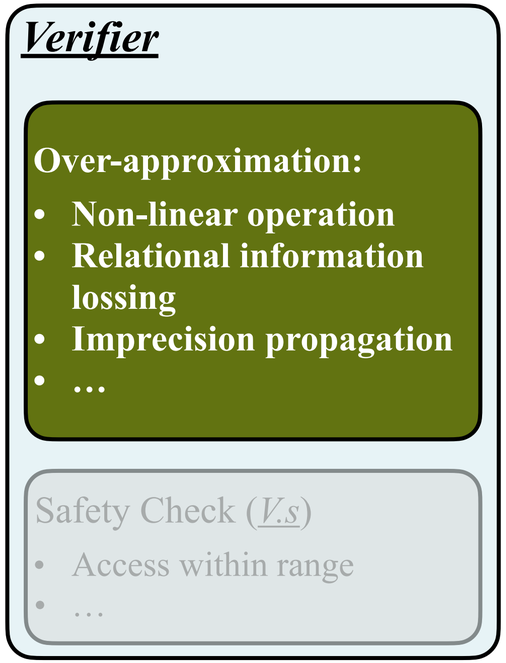
Precision vs. Complexity





Key challenge: achieving **higher precision** while maintaining **low complexity**

Imprecisions

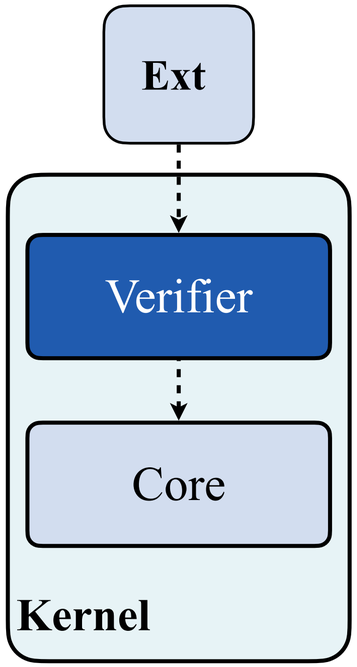


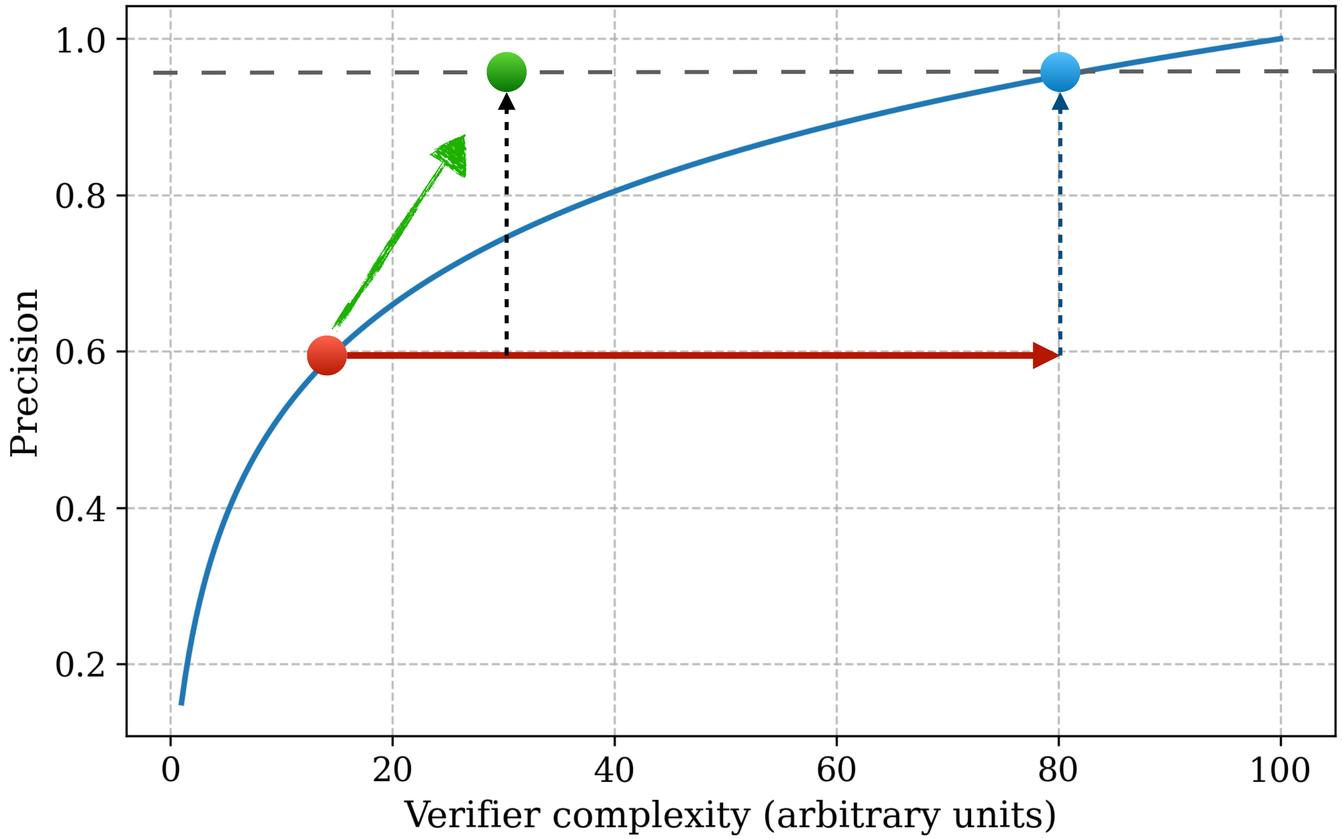
Instruction	Abstraction
1: r1 = map_lookup(…)	*r1 = ptr(**size=16**)*
2: r2 &= 0xf	*r2 ∈ [0,15]*
3: r1 += r2	*r1.off ∈ [0,15]*
4: r3 = 0xf - r2	*r3 ∈ **[0,15]***
5: r1 += r3	*r1.off ∈ **[0,30]***
6: r0 = *(**u8***)r1	Unsafe, rejected…

```
// 256 KiB per cpu core, of which 128 KiB is usable as
// we have to bound each new variable-length field to
// start at no more than half the size of the buffer to
// make the verifier happy.
#define EVENT_BUFFER_SIZE (1 << 18)
#define EVENT_BUFFER_SIZE_HALF (EVENT_BUFFER_SIZE >> 1)
```

Listing 3. Workaround to mitigate the verifier's imprecision.

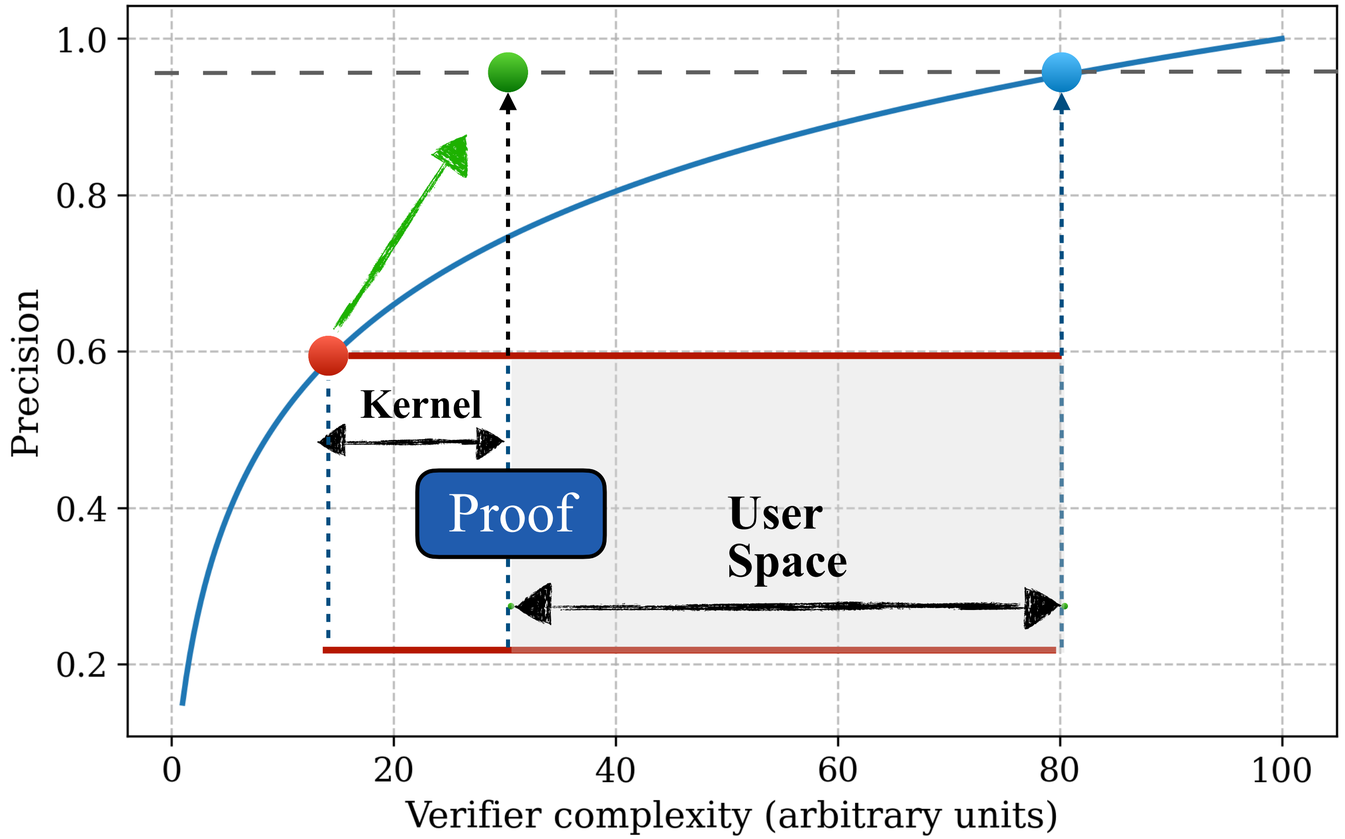
Our Goal





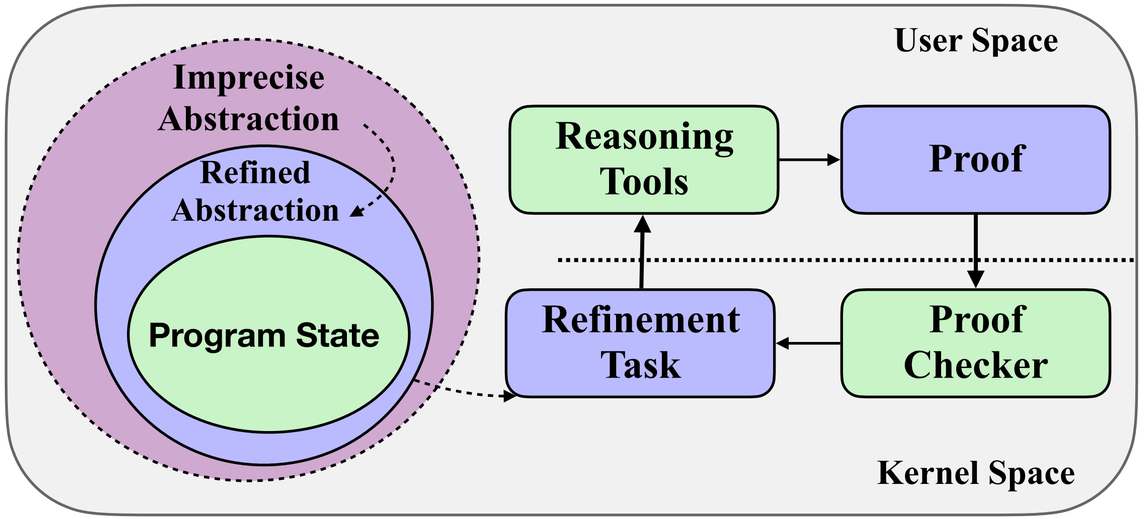
Goal: significantly enhance the precision with **linear-time** kernel space complexity.

Key Idea



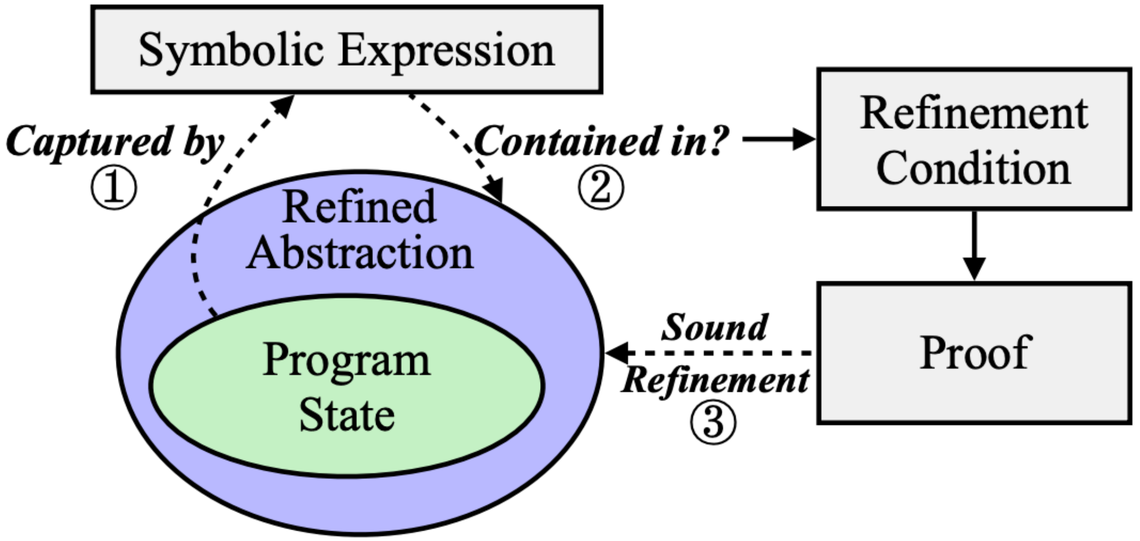
Keep the verifier simple, delegate nontrivial reasoning, and bridge the gap with proofs.

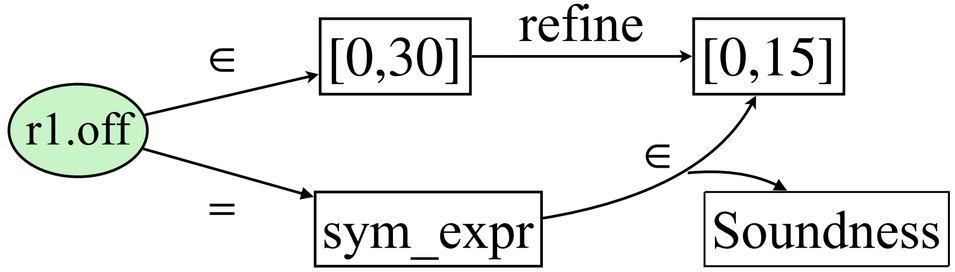
Proof-Guided Abstraction Refinement



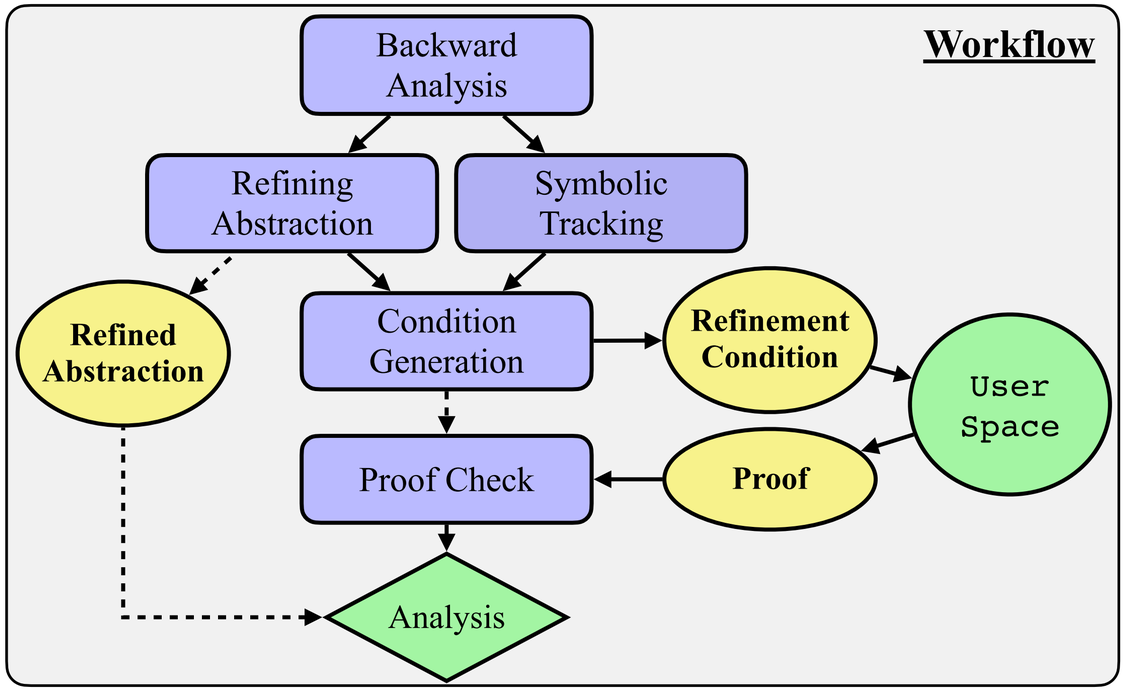
- On demand **abstraction refinement**:
  - The kernel verifier runs cheap analysis
  - Stops only when precision limits are reached
- Refinement soundness as **proof obligation:**
  - Encode refinement soundness as a formal formula
  - *User space* solver produces a formal proof
- Kernel space **proof checking**:
  - *Kernel proof checker* validates the proof
  - Verifier continues with the refined abstraction

Proof-Guided Abstraction Refinement





Workflow




```
...                     // instructions before are irrelevant
                        // (analysis ends, the suffix found)
r1 = map_lookup(...)    // {} (r1 defined)
r2 = load_ctx(...)      // {r1} (r2 defined)
r2 &= 0xf               // {r1, r2}
r1 += r2                // {r1, r2}
r3 = 0xf - r2           // {r1, r2} (r3 defined, r2 added)
r1 += r3                // {r1, r3} (r3 added to the set)
r0 = *(u8*)r1           // {r1} (backward analysis starts)
```

Listing 4. Backward analysis to pinpoint the start location.

Refinement

Instruction	**Abstraction**	**Refinement**
`1: r1 = map_lookup(…)`	*r1 = ptr(**size=16**)*	*r2 = sym*
`2: r2 &= 0xf`	*r2 ∈ [0,15]*	*r2 = sym&0xf*
`3: r1 += r2`	*r1.off ∈ [0,15]*	*r1.off = sym&0xf*
`4: r3 = 0xf - r2`	*r3 ∈ **[0,15]***	*r3 = **0xf - sym&0xf***
`5: r1 += r3`	*r1.off ∈ **[0,30]***	*r1.off = **(sym&0xf) + (0xf - sym&0xf)***
`6: r0 = *(u8*)r1`	Unsafe, refine…	*Condition: r1.off <= 15*
`6: …`	*r1.off ∈ **[0,15]***	*Proved and r1.off **refined***

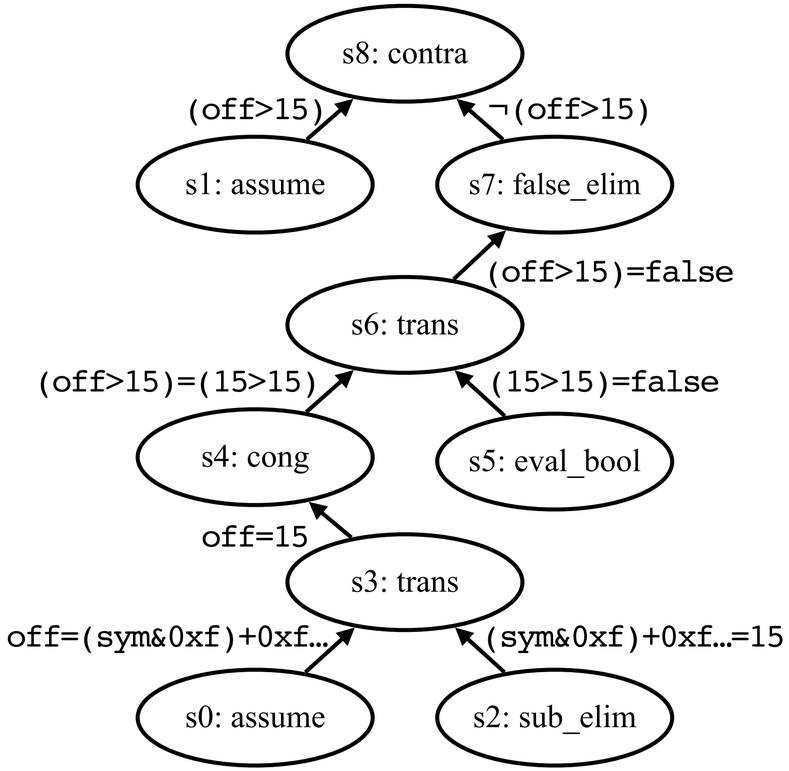
Proof Check

Step	Rule	Premise	Conclusion
s0	*assume*	–	`off = (sym&0xf) + 0xf - (sym&0xf)`
s1	*assume*	–	`off > 15`
s2	*sub_elim*	–	`(sym&0xf) + 0xf - (sym&0xf) = 15`
s3	*trans*	*s0,s2*	`off = 15`
s4	*cong*	*s3*	`(off > 15) = (15 > 15)`
s5	*eval_bool*	–	`(15 > 15) = false`
s6	*trans*	*s4,s5*	`(off > 15) = false`
s7	*false_elim*	*s6*	`¬(off > 15)`
s8	*contra*	*s1,s7*	**FALSE**

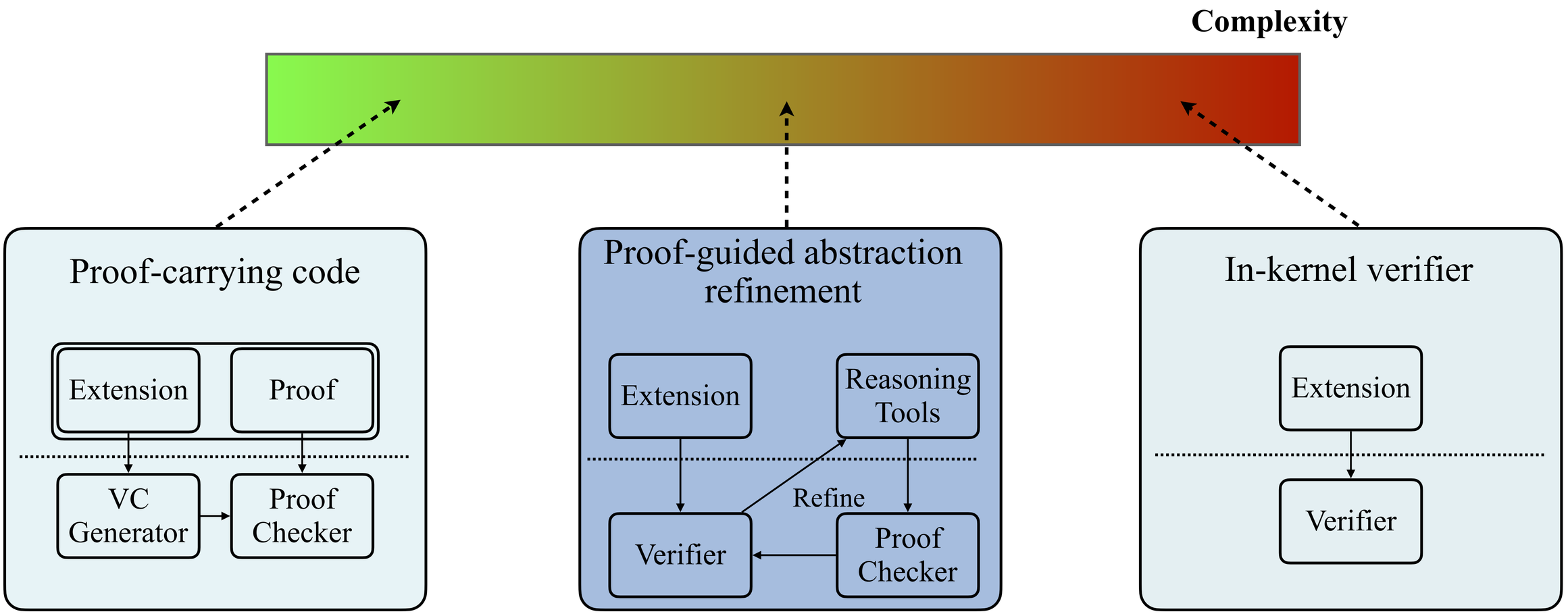
Rules:

$$\frac{t_1 = t_2, \ldots, t_{n-1} = t_n}{t_1 = t_n}\ \textit{trans} \qquad \frac{t_1 = s_1, \ldots, t_n = s_n}{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)}\ \textit{cong}$$

$$\frac{}{a + b - a = b}\ \textit{sub\_elim} \qquad \frac{F = \text{false}}{\neg F}\ \textit{false\_elim} \qquad \frac{F,\ \neg F}{\text{false}}\ \textit{contra}$$

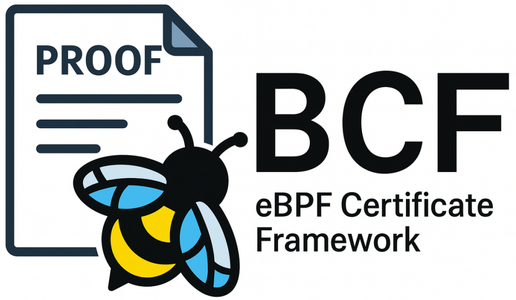


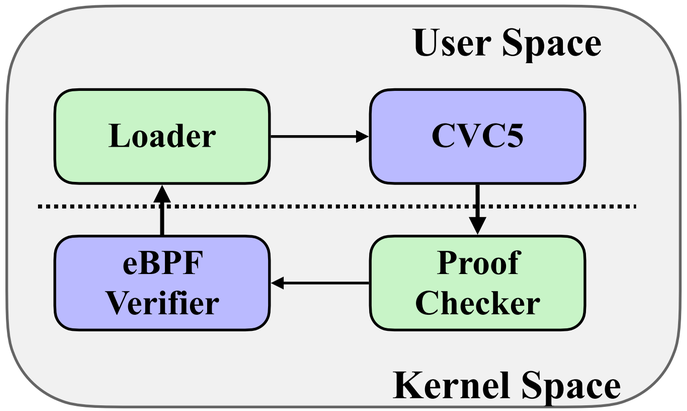
Complexity


Complexity
Proof-carrying code
Extension
Proof
VC Generator
Proof Checker
Proof-guided abstraction refinement
Extension
Reasoning Tools
Refine
Verifier
Proof Checker
In-kernel verifier
Extension
Verifier

BCF: eBPF Certificate Framework

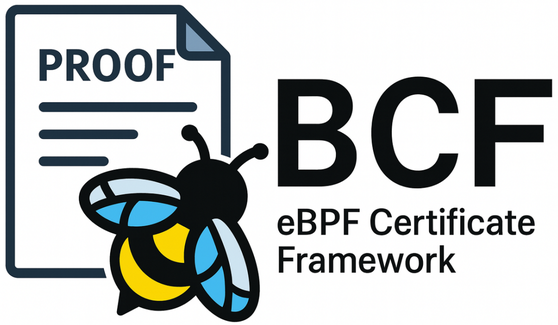
- Refinement procedure in the verifier (1.7K LOC)
  - Triggered when analysis stalls
  - Produces a refinement condition
- Loader program
  - Receive the condition, translates it for the solver
- CVC5 SMT solver
  - Performs reasoning and generates a formal proof
- In-kernel proof checker (5K LOC)
  - Performs linear-time proof validation





BCF: eBPF Certificate Framework

- Initial prototype open-sourced:
  - https://github.com/SunHao-0/BCF/tree/artifact-evaluation
- Continuous improvement:
  - https://github.com/SunHao-0/BCF/tree/main
  - Proof checker completely rewritten
  - Supports 50 proof rules:
    - 14 core, 33 boolean, 3 bitvector rules
  - 151 rewrite rules automatically converted from CVC5 RARE rewrites
- Ultimate goal: make this happen in the Linux kernel.



Evaluation

- Compiler-driven approach to derive a dataset
- For the same program, compile it with different configurations
- For the same source, rejection in an alternative configuration indicates a false rejection
- 512 programs collected in total
  - from real-world project
  - compiled with widely-adopted compilers
- Sizes range from 0.5 to 376 KiB

Object	Project	Size	Loc	Description
bpf_lxc.o [30]	Cilium	269 KiB	2,450	Container identity and policy
bpf_host.o [29]	Cilium	376 KiB	2,086	Host level policy and route
pping_kern.o [91]	xdp-project	19 KiB	1,546	XDP packet timestamping
xdp_synproxy.o [92]	xdp-project	9.9 KiB	821	XDP-based SYN proxy
felix_bin_bpf.o [26]	Calico	188 KiB	2,162	Pod network policy
ksnoop.bpf.o [53]	BCC	6.3 KiB	461	Kernel function tracing

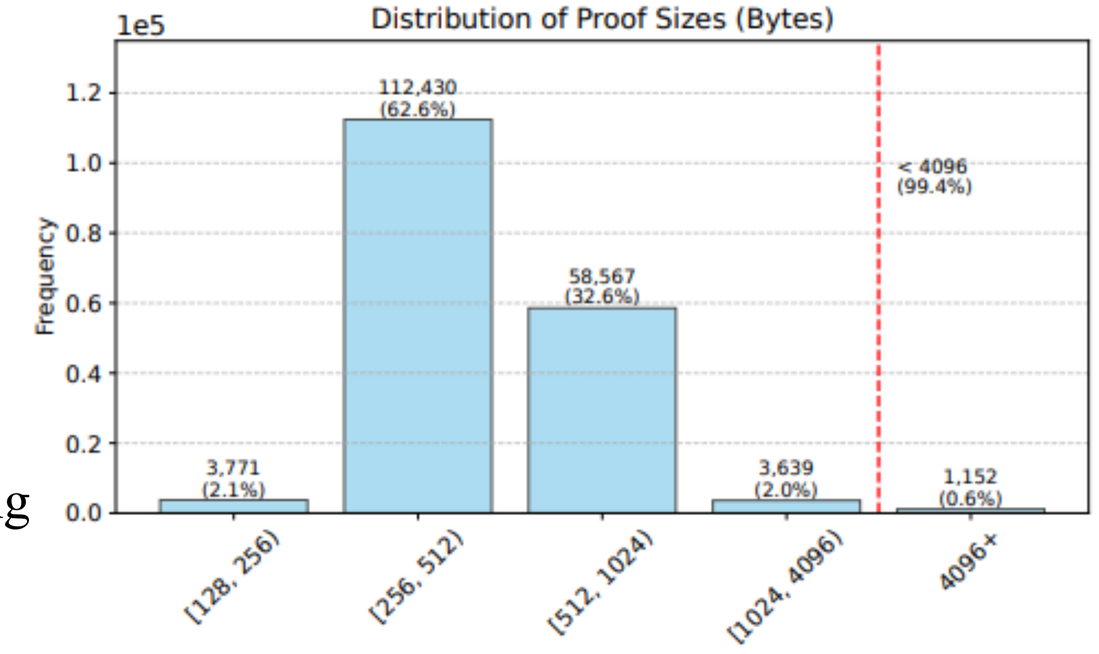
Key Results

- 403 out of 512 programs automatically loaded
- 78.7% accepted, 21.3% still rejected
- Reasons for remaining rejections:
  - Refinement not triggered for four programs
  - 82 were due to condition not satisfied
  - 23 were due to reaching the one million instruction limit
- The first two cases can be solved with further engineering efforts
- The last case requires better loop handling

Metric	Min	Avg	Max
Refinement Frequency	1	446	16,048
Symbolic Track Length	7	102	373
Condition Size (bytes)	88	836	2,128
Proof Check Time (μs)	31	49	1,845
Proof Size (bytes)	136	541	46,296

Key Results

- 403 out of 512 programs automatically loaded
- 78.7% accepted, 21.3% still rejected
- Reasons for remaining rejections:
  - Refinement not triggered for four programs
  - 82 were due to condition not satisfied
  - 23 were due to reaching the instruction limit
- The first two cases can be solved with further engineering
- The last case requires better loop handling



Thank you!

ETH zürich