

Mitigating Bugs in the Linux eBPF Verifier using Rust- or PREVAIL-based Layered Verification

FOSDEM'25 — Feb. 1, 2025

Luis Gerhorst¹, Timo Hönig²

¹ Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

² Ruhr-Universität Bochum, Germany



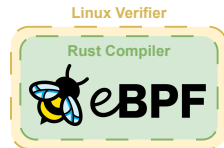
Friedrich-Alexander-Universität
Faculty of Engineering

RUHR
UNIVERSITÄT
BOCHUM

RUB

Motivation and Overview

- Kernel verifier is bug-prone
- Existing solutions introduce runtime overheads or are not exhaustive
- Layered-security approach
 - Verify program-safety twice
 - Bugs in one algorithm are “*caught by the other verifier?*”
- Case study with Rust Compiler and PREVAIL
 - Exploitable? *Yes, but sometimes tedious*



Layered Verification



■ Two verifiers

- Implementations should be as different as possible
- Must be unlikely to share the same bug

■ Potential benefits

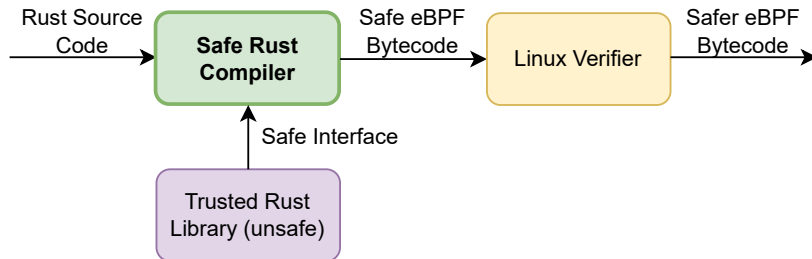
- No runtime overheads
- Reuse existing tools

- Safe Rust Compiler:
 - All unsafe features disabled
 - Generates BPF object files
- PREVAIL:



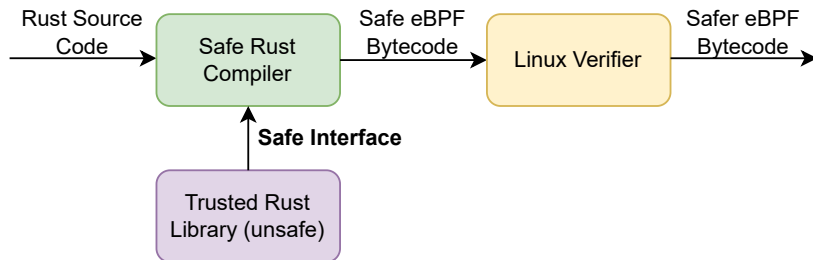
- *Changes Required? Bugs? Exploitable?*

Adaptions for a Safe Rust Compiler



- Disable unsafe and bug-prone features (e.g., `proc_macro`)
- Termination checks

Adaptions for a Safe Rust Library Interface



- Extend `aya-rs`
- Guarantee non-static destructors

Scope of Rust Compiler Bugs

- All compiler bugs: 3979 to 9906
- Arbitrary-code-execution bugs: 0 to 104
 - None *reported so far*
 - 104 compiler crashes reported (SIGSEGV etc.)
 - Potential for further hardening exists
- Relevant unsoundness bugs: 34 to 94
 - Exploitable if there are **only unsoundness bugs?**

Exploit for Rust-based Layered Verification

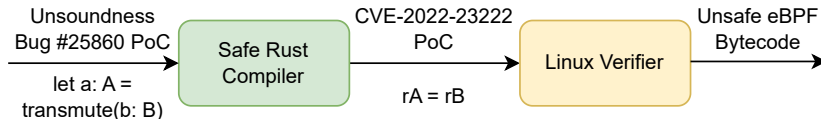
- Rust Unsoundness Bug #25860¹
 - *“Implied bounds on nested references + variance = soundness hole”*
 - Open since 2015, to be fixed by work-in-progress design changes
- Linux Verifier CVE-2022-23222
 - Pointer-arithmetic tracked incorrectly

¹<https://github.com/rust-lang/rust/issues/25860>

Rust Unsoundness Bug #25860: Unsafe Transmute Primitive

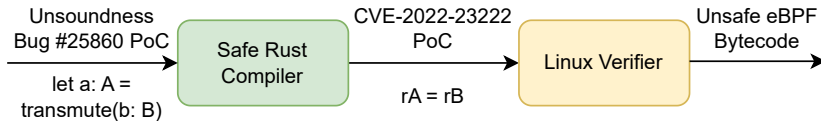
```
pub fn transmute<A, B>(obj: A)
    -> B {
    enum DummyEnum<A, B> {
        A(Result<
            A, Option<Blank<A, B>>>),
        ...
    }
    ...
    core::mem::replace(
        ref_to_b, Err(None)
    ).ok().unwrap()
}
```

Exploit for Rust-based Layered Verification: Approach



1. Use Rust unsoundness bug for type-unsafety
2. Create OOBs pointer
3. Find Rust program that generates unsafe bytecode missed by Linux Verifier

Exploit for Rust-based Layered Verification: Techniques



- Use `transmute()` to construct OOB pointer
 - Optimizations remove complexity
 - Compiles to register assignment `rA = rB`
- Prevent unwanted optimizations using data-dependencies
- **Tedios**: Get register-usage as needed
 - e.g., `ptr += scalar` vs. `scalar += ptr`

PREVAIL-based Layered Verification



- Does not reuse kernel code → Bugs likely different
- 37 issues open (1 potentially relevant bug)
- Assume there exists bytecode $F()$ which is **incorrectly abstracted**
- Exploitable?

Exploit for PREVAIL-based Layered Verification: Idea

- Incorrectly abstracted functions
 - F1(): Linux eBPF Verifier (e.g., CVE-2017-16995)
 - F2(): PREVAIL (assumed)
- Approach for layered-verification exploit
 - **Combine** result of F1() and F2()
 - “Hide” OOB read/write

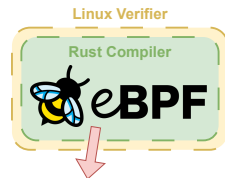
Exploit for PREVAIL-based Layered Verification: Code

```
r1 = F1()      # actual: 1  Linux: 0  PREVAIL: 1
r2 = F2()      # actual: 1  Linux: 1  PREVAIL: 0
r3 = r1 & r2    # actual: 1  Linux: 0  PREVAIL: 0

r3 *= OOB_OFF
r4 = PTR_TO_MAP_ELEM( )
r4 += r3        # actual: r4+OOB_OFF  Linux/PREVAIL: r4
*r4 = ...       # OOB write
```

Summary

- Layered verification has limited potential
 - Exploits can be **combined**
 - Roughly **doubles exploitation effort**
 - Bugs in the runtime (e.g., helpers) not covered
- Effort for exploit
 - Rust** Tedious (because of indirection)
 - PREVAIL** Easy
- <https://github.com/luisgerhorst/sust>



The following slides were not presented during the talk.

- 34 unsoundness bugs relevant to eBPF:

[https://github.com/rust-lang/rust/issues?q=is%3Aissue%20state%3Aopen%20label%3AI-unsound%20AND%20\(%20label%3AT-compiler%20OR%20label%3AT-lang%20\)%20AND%20\(-label%3AO-Arm%20-label%3AO-wasm%20-label%3AO-PowerPC%20-label%3AO-windows%20-label%3AO-AArch64%20-label%3AO-SPARC%20-label%3AO-x86_64%20-label%3AO-MIPS%20-label%3AO-windows-msvc%20-label%3AO-AVR%20-label%3AO-x86_32%20-label%3AA-inline-assembly%20-label%3AF-extern_types%20-label%3AF-type_alias_impl_trait%20-label%3Afixed-by-next-solver%20-label%3AF-thread_local%20-label%3AF-simd_ffi\)%20&page=1](https://github.com/rust-lang/rust/issues?q=is%3Aissue%20state%3Aopen%20label%3AI-unsound%20AND%20(%20label%3AT-compiler%20OR%20label%3AT-lang%20)%20AND%20(-label%3AO-Arm%20-label%3AO-wasm%20-label%3AO-PowerPC%20-label%3AO-windows%20-label%3AO-AArch64%20-label%3AO-SPARC%20-label%3AO-x86_64%20-label%3AO-MIPS%20-label%3AO-windows-msvc%20-label%3AO-AVR%20-label%3AO-x86_32%20-label%3AA-inline-assembly%20-label%3AF-extern_types%20-label%3AF-type_alias_impl_trait%20-label%3Afixed-by-next-solver%20-label%3AF-thread_local%20-label%3AF-simd_ffi)%20&page=1)

- 94 unsoundness bugs: <https://github.com/rust-lang/rust/issues?q=is%3Aissue%20state%3Aopen%20label%3AI-unsound%20>
- 104 compiler crashes:
<https://github.com/rust-lang/rust/issues?q=is%3Aissue%20state%3Aopen%20label%3AT-compiler%20label%3AI-crash>
- Rust unsoundness bug exploited: <https://github.com/luisgerhorst/sust/blob/10a103f8dad718c54813bf715dad69da3399fba5/ebpf/cgroup-skb-egress-ringbuf/cgroup-skb-egress-ebpf/src/main.rs#L42>

- Other Rust compiler features that might be bug-prone or unsafe: feature(specialization), panics, unions, trait objects, projections, async/await, self-referential types