barkhausen institut

Applying Modern Verification Techniques to a Root-of-Trust Bootloader

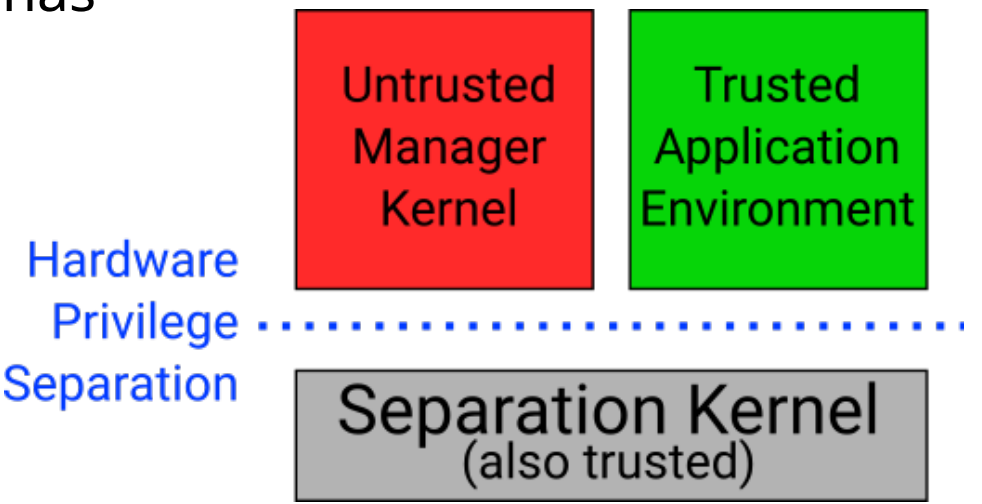
Nicholas Gordon
Barkhausen Institut

Carsten Weinhold
Barkhausen Institut

Introduction
Formal Verification: The Next Frontier in Trusted Computing?

- Trusted Computing paradigm has "settled"

- Systems still insecure!
  - *Software has bugs.*

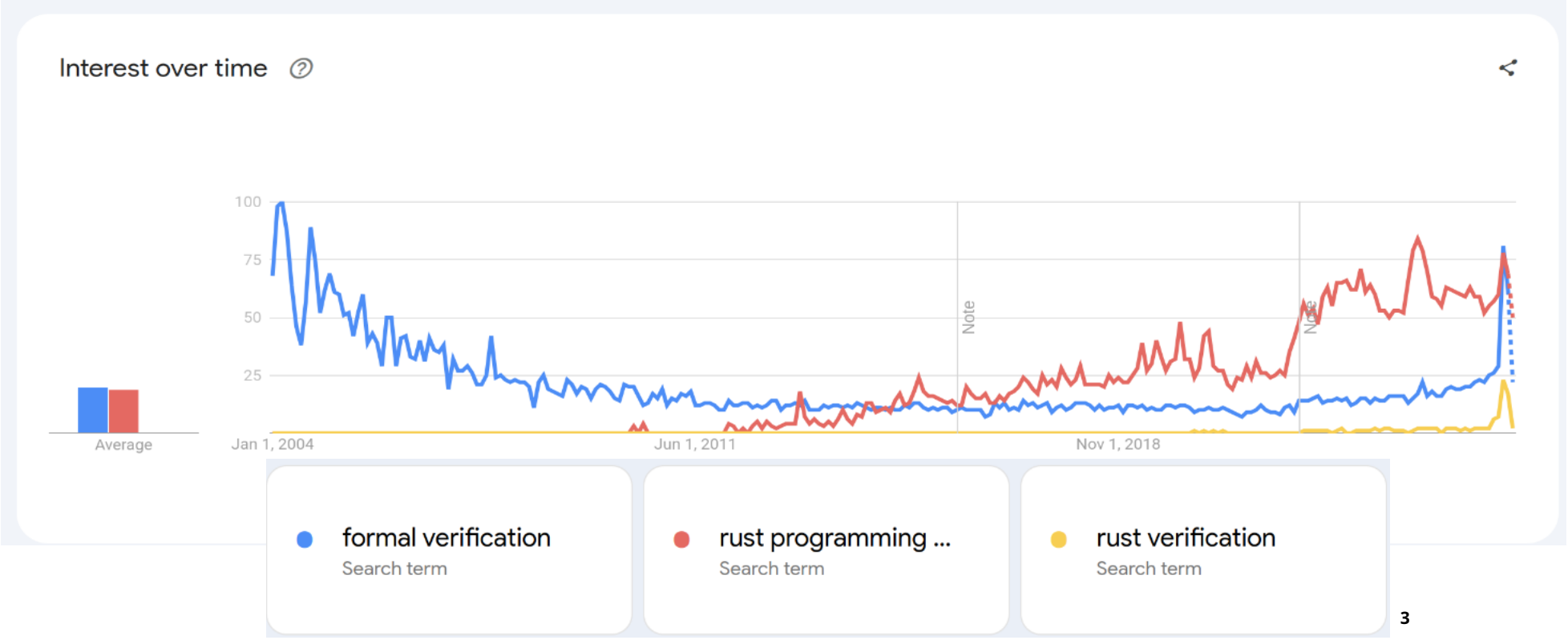


- *Solution*: Verification for security, robustness

- This talk: half **experience report**, half **call-to-action**

Recent Advances and Hype

- Software language changes give new opportunities: Rust



Recent Advances and Hype

- Lots of energy in Rust in particular
  - Kani, Prusti, Creusot, Verus, …
- Elsewhere, too:
  - Rocq/Isabelle/Lean + automation
  - Prover-language embeddings (e.g. RefinedRust)
- *Opportunity? Maybe have another look*

Motivation/Problem
System Software and Verification: A Poor Match?

- Hurdles:
  - „Heroic“ efforts: SeL4, CertiKOS
    - Design limitations
  - Tool use difficulties
  - Applicable to existing software?

Idea/Approach
Test-Drive a Verification Tool

- There have been developments since the milestone works
  - SeL4: **2009**
  - CertiKOS: **2016**

- Modern, Rust-based framework (follow the hype)
  - Which one to pick?

Idea/Approach
Tool Use Difficulties

- What kind of tool to use?
  - SMT Solvers (Verus, Kani)
  - Foundational tools (Rocq, Isabelle)
    - Other Barkhausen talk to learn more

Idea/Approach
Verus

- Verus: expressiveness, accessibility, syntax
- SMT-based solver
- Encodes properties as Hoare triples (pre- and post-conditions)

```
#[inline(always)]
#[cfg(target_arch = "riscv64")]
unsafe fn prepare_switch<Data: CtxData>(
    ctx: LayerCtx<Data>,
    where_to: *mut LayerCtx<Data>,
    Tracked(where_to_perm):
        Tracked<&mut PointsTo<LayerCtx<Data>>>,
    eclear: *const u8,
) -> (entry: usize)
    requires
        where_to@.addr == crate::MEM_OFFSET,
        eclear@.addr != 0,
        eclear@.addr >= where_to@.addr,
        old(where_to_perm).ptr() == where_to,
        old(where_to_perm).is_uninit(),
    ensures
        entry != 0,
        entry >= eclear as usize,
        where_to_perm.opt_value() ==
            MemContents::Init(ctx),
        crate::range_cleared(crate::MEM_OFFSET as int,
                             eclear@.addr as int),
{ ... }
```

Idea/Approach
Why re-write? Retrofit!

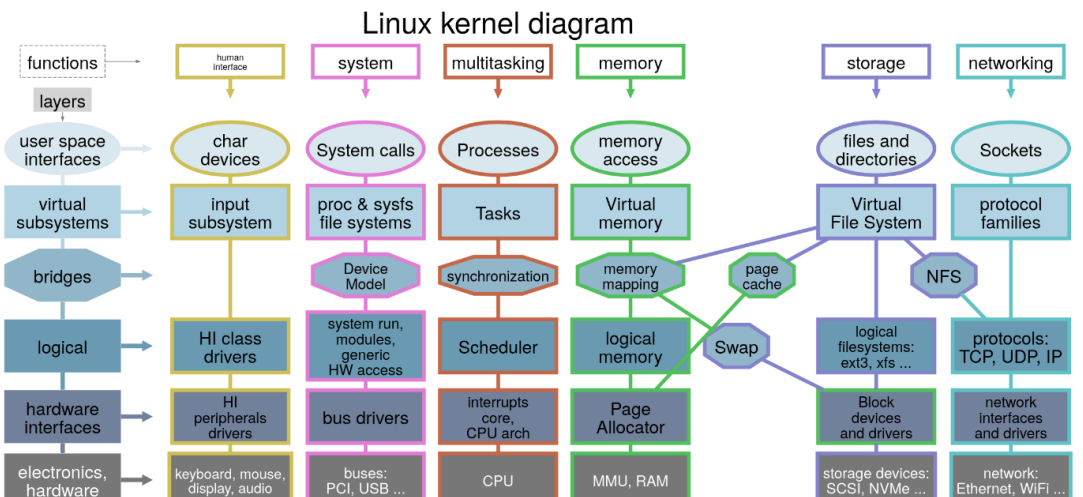
- Efforts have been (mostly) *from-scratch*

- *Why is this a problem?*
  - Re-writing “with verification in mind” not very practical.

- *Retrofit* something instead?

Idea/Approach
What to retrofit?

- Conventional kernels *way too big*
  - Too much complexity
  - Tool suitability unclear (sunk cost fallacy)
  - What to prove?
  - Novelty? (vs. e.g. seL4)



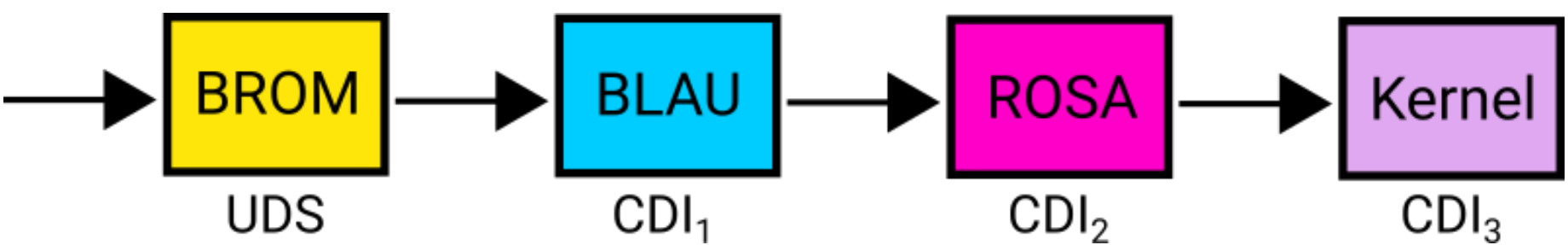
[1]

Idea/Approach
Ideal target: Bootloader ?

- Bootloader
  - Low-level, low-abstraction
  - Good exercise of tool's ability to handle *unsafe* code, interact with hardware

Implementation
Verus + M³ Bootloader

- M³ bootloader
  - Provides a root of trust (DICE protocol)
  - In use for other research
  - Four boot stages



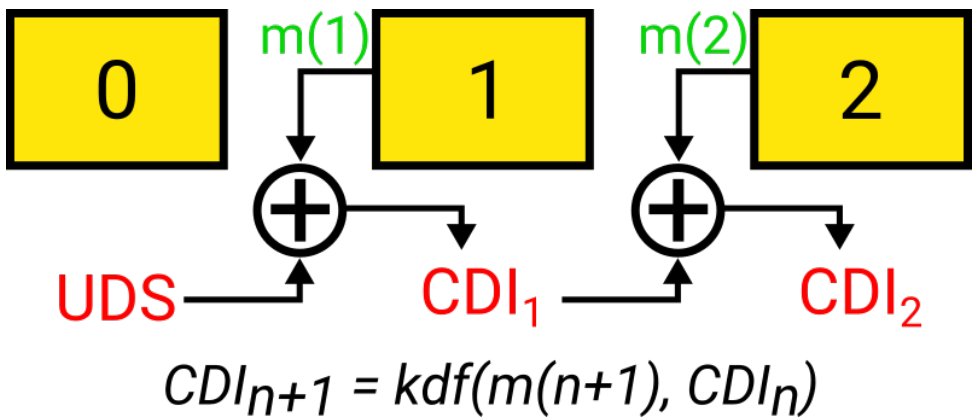
Implementation
Verus + M³ Bootloader

What to prove?

- DICE: measure next stage, combine with your secret, hand bundle to next stage

- Secure inter-stage erasure property:
  - Data ownership, raw memory, context switching, assembly

  - Straightforward to express



Implementation
What's the spec?

- How does the spec look?

```
pub uninterp spec fn addr_cleared(a: int) -> bool;

pub open spec fn range_cleared(start: int, end: int) -> bool {
  forall|i: int| start <= i < end ==> #[trigger] addr_cleared(i)
}

pub open spec fn wf(layer : Bootstage) -> bool {
  range_cleared(layer.begin, layer.end)
}
```

Implementation
Verus + M³ Bootloader

- Implemented directly into existing Rust codebase
  - Integrated with M³'s semi-custom build system (!)

- Stubbed out „non-critical" code
  - Unrelated application code
  - Library code

- Reorganization was necessary to cooperate with Verus
  - Still research software, after all

Evaluation
Proof-To-Code Ratio, Performance, Developer Experience

- *What did we prove?* Assumptions/limitations:
  - Inherit from Verus (Rust semantic assumptions)
  - Trust M³, Verus, and then Rust libraries (~90k LOC)
  - Trust hardware definitions, trust linker symbols

```
fn main() → ! {
    ...
    <bootloader stuff>
    ...
    assert(rot::range_cleared(rot::MEM_OFFSET as int,
                              rot::get_eclear() as int));
    loop { }
}
```

Evaluation
Proof-to-Code Ratio

- Proof to code ratio was very low: 0.56 (proof LOC to source LOC)
  - Bootloader is simple
  - Selected property was also very simple
  - Matichuk shows quadratic growth of proof with spec

	Application	Library	Total
LoC Without Verus	*725*	607	1,332
LoC With Verus	899	1,175	*2,074*
Verus LoC	*174*	*568*	*742*
Ratio	*0.24 (=174/725)*	*0.94*	*0.56*

Evaluation
Performance

- Performance impact negligible
  - Strength of ownership types/"ghost code" concept

Stage	Orig.	With Verus	Diff.	Percent
Brom	1.8	1.5	-0.3	-16.78%
Blau	52,977	52,943	-34	-0.06%
Rosa	72,205	76,958	4,753	6.58%

- Binary size similarly unaffected
  - Size *decreased* in some cases due to compiler optimization

Stage	Orig.	With Verus
Brom	30K	30K
Blau	58K	42K
Rosa	109K	110K

Evaluation
Developer Experience

- Though very popular, Verus still research software
  - “Quality of life”, usability improvements
  - More complete low-level memory support
  - Broader/more complete support for Rust constructs

- How long did it take to get comfortable?
  - Total effort: ~3 months
  - “Acclimitization time” of ~2 months

  - *Pretty reasonable*

Discussion
Conclusion, Future Directions, Lessons Learned

- New avenues opening up for verified system software
  - Practical step in the development cycle?
  - I'm a kernel guy, not a verification guy

- Limitations: small scale, simple property

- Future questions:
  - Applicability at scale?
  - Hardware model? Register contents, interruptibility, etc.
  - Other tools?

Thank you
Questions and Feedback

References:
[1] https://makelinux.github.io/kernel/diagram/
[2] RefinedRust: https://plv.mpi-sws.org/refinedrust/
[3] Verus code: https://github.com/verus-lang/verus

Backup Slides

- Why not Kani?
  - Bounded model checkers are useful, too!
  - Also accessible to systems programmers
  - Different workflow compared to contracts (test-based)

Backup Slides

- How big is the bootloader?
- ~750 LOC in application
- ~600 LOC in specific libraries
- Depends on some fraction of M^3 library
  - ~29k LOC

Backup Slides

- What were those Verus hangups?
  - Before:

```
impl CtxData for BromCtx {
    const MAGIC: Magic = encode_magic(b"BromCtx", 1);
}
```

  - After:

```
impl CtxData for BromCtx {
    fn magic() -> Magic {
        0x01000001
    }
    //const MAGIC: Magic = encode_magic(b"BromCtx", 1);
}
```