Towards Modular Specification and Verification of Concurrent Hypervisor-based Isolation

Hoang-Hai Dang
BedRock Systems, Inc
hai@bedrocksystems.com

David Swasey
BedRock Systems, Inc
david@bedrocksystems.com

Gregory Malecha
BedRock Systems, Inc
gregory@bedrocksystems.com

Keywords NOVA, concurrent separation logic, robust safety, hypervisor

1 Introduction

NOVA [10] is a microhypervisor that executes in a privileged processor mode and provides basic services for virtualization, isolation, scheduling, and management of physical system resources. NOVA's design goal is to reduce the critical code base, and to leave richer virtualization features to user applications, such as a Virtual Machine Monitor (VMM), that run in a less privileged mode (e.g. user mode). The size (about 17K lines of code and 15 hypercalls [9]) makes it a suitable target for *formal verification*; however, its complexity poses several challenges:

- NOVA claims that it behaves *robustly* when running arbitrary user code.
- NOVA provides a rich capability-based API to interact with *kernel objects* representing system resources.
- The implementation (in C++ and assembly) is actively developed and supports both ARMv8 and x86-64. It is also highly optimized and concurrent, leveraging both atomics[1] as well as systems-level features.

In this talk, we focus on our approach to specify and verify NOVA using *concurrent separation logic*. In addition, we consider how this approach is compatible with reasoning about untrusted code.

2 A Hypercall's Path through NOVA

NOVA extends the semantics of the physical machine with a set of hypercalls. Figure 1 shows the basic steps of NOVA handling a hypercall from a user application (e.g., a VMM).

1. The user application (C++) calls a function implemented in the architecture-specific assembly (ISA), to set up the contents of the required CPU registers with valid arguments, following NOVA's calling convention (ABI).

2. From the ISA user mode, control is passed on to NOVA in ISA privileged mode, where (in assembly) NOVA parses registers, sets up the C++ call stack, and re-enters C++.

[1]While NOVA uses weak-memory atomics, our current verification focuses on a variant of NOVA that uses SC atomics.

PriSC'24, January 20, 2024, London, UK

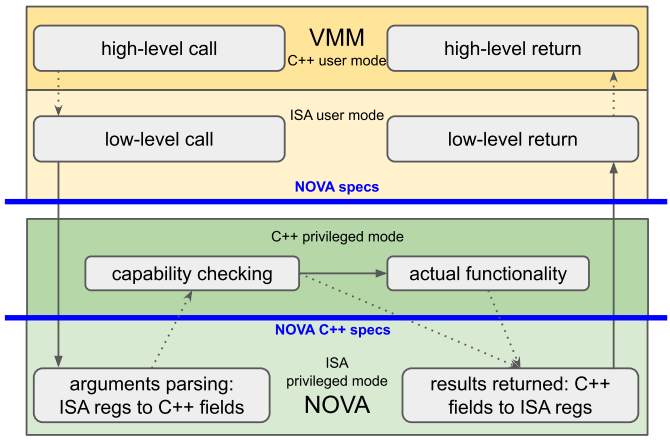

Figure 1. A path of a hypercall into NOVA

3. NOVA's core logic is implemented in C++ but must still drive system components such as interrupt controllers and page tables.
4. NOVA's C++ code returns its results through assembly by resetting its stack, updating user registers with the results of the hypercall, and returning to user mode.

3 Specify and Verify NOVA

Our goal is to achieve scalability in verifying NOVA and its clients. In addition to *functional correctness* properties, we want to prove the following properties:

- *robust safety*: user mode code cannot break NOVA or critical components like the VMM;
- *isolation*: any code running atop NOVA in user mode can only access physical resources assigned to it; and
- *refinement*: the NOVA-extended ISA semantics allows multiple VMs to run on the same machine *as if* they were running on separate machines.

Towards that end, we exploit concurrent separation logics (CSLs) and their state-of-the-art features to construct small-footprint, highly-expressive and precise specifications.

As the first step, we have developed resources and logically atomic specifications [3, 6, 11] for most of NOVA's kernel objects and hypercalls. Using BRiCk [7]'s axiomatized CSL for C++ and its proof automation infrastructure, we have verified a sequentially-consistent version of NOVA's C++ code for the ctrl_sm ("control semaphore") hypercall as well as several lower-level modules used in NOVA.

Local and precise logically-atomic specifications Our NOVA specifications are designed to *precisely* capture NOVA's state at every visible *linearization point*. For example, the following specifies steps in the ctrl_sm hypercall to wake up (UP) a semaphore *sm*.

$$\left\langle \begin{array}{l} c.\ sm \overset{\text{cap}}{\longmapsto} c \mid \\ \qquad\qquad \text{if UP} \notin c \text{ then BAD\_CAP} \\ ().\ sm \overset{\text{cap}}{\longmapsto} c \Rrightarrow \text{else} \left\langle \begin{array}{l} n.\ sm \overset{\text{val}}{\longmapsto} n \mid \\ n'.\ sm \overset{\text{val}}{\longmapsto} n' \Rrightarrow \Phi(n') * \ldots \end{array} \right\rangle \end{array} \right\rangle$$

Each pair of angle brackets ($\langle x.\ P \mid y.\ Q \Rrightarrow \Phi \rangle$) specifies the logically-atomic effects of a linearization point, where P is the atomic precondition right before, and Q the atomic postcondition right after the linearization point. In this example (to UP an *sm*), there are two visible linearization points, one to check the capability (which requires the resource $sm \overset{\text{cap}}{\longmapsto} c$ as the precondition), and one to update the *sm*'s counter (which requires the resource $sm \overset{\text{val}}{\longmapsto} n$). We have found that the small-footprint specifications enabled by separation logic are quite modular from other hypercalls specifications and even the NOVA state that backs unrelated kernel objects. Consequently, we have been able to adapt the specifications, and even the proof, of logically separated pieces while other portions of the NOVA implementation have changed.

Most importantly, the specifications aim to capture precisely *all possible visible interferences* that can happen during a hypercall, including both those from trusted code and those from untrusted code. While this complicates the specifications, it is easy to derive simpler specifications that can be used by trusted-code applications on top of these. The benefit of these precise specifications is that they should enable us to prove properties that concern untrusted user code.

Specifying the Behavior of User Code To describe the behavior of (potentially untrusted) user code running concurrently atop NOVA, we use a small step, operational model based on the processor models developed in SAIL [1]. This operational model of a core interacts with the rest of the system, e.g. memory, hypercalls, and priviledged state, through events. To embed this into separation logic, we describe the semantics of a NOVA execution context (similar to a thread) as a weakest precondition where the user events are "handled" in separation logic. For example, when the user model emits a syscall event, we extract the register state in separation logic and dispatch to the appropriate hypercall specification such as the one for ctrl_sm presented above. Similar handling occurs for memory accesses where we describe the 2nd stage address translation using the abstract state of the page tables that are exposed as NOVA state.

We conjecture that, if we can prove a wp_nova_cpu for every user thread in our whole machine configuration,[2] we should be able to extract strong properties (such as robust safety) for the whole machine. Note that since the specification are all written in the same separation logic, we can combine the machine and NOVA resources with invariants and custom, higher-order ghost state for these proofs.

An intensional approach to Robust Safety The usual approach to verifying robust safety [2, 4, 8, 12] is to classify security-relevant state into low- vs. high-integrity state, and to attach a protocol to the low-integrity state; then, one proves that all machine steps are compatible with those protocols for low-integrity state (i.e, given only low-integrity state, untrusted code can take any step available to it).

By exposing NOVA's states as aforementioned specifications without any security-awareness but with precise functional correctness, we leave the client of NOVA interface the choice to define custom protocols/abstractions for low-integrity state and to derive simple specifications for working with these abstractions. Then, by proving wp_nova_cpu for all possible user-code programs while maintaining the low-integrity protocols, the client should eventually be able to show a robust safety metatheorem.

4 Open Challenges and Next Steps

Verifying NOVA's C++ code against precise, modular specifications is only part of a much bigger effort. By refining the challenges unique to a microkernel (as provided in §1), we identify the following more concrete challenges:

(A) Develop concurrent separation logics for the languages used by NOVA and its applications. These include logics for C++ as well as assembly languages running in both user and privileged modes.

(B) Specify and verify the interfaces between these logics, so that we can move back and forth between C++ and assembly, in both user and privileged modes, as needed in the dotted arrows in Figure 1. Promising work exists in this direction in Melocoton [5].

(C) Develop the separation logic interfaces for NOVA, which include small-footprint resources for NOVA's kernel objects and states as well as logically atomic specifications for accessing and updating those resources. These logically atomic specifications encode the functional correctness of NOVA, including the capability checking and each hypercall's function.

(D) Verify NOVA implementation, modularly by each hypercall, against the specifications in (C). The verification should be done using C++ privileged-mode logic from (A), and then connect to the ISA specifications (C) using the cross-language interfaces in (B).

(E) Derive security properties, such as robust safety and isolation, from the ISA semantics and NOVA's functional correctness specifications. Verifications of user applications contribute to this task.

In this talk, we report our on-going efforts and results in (C) and (D).

[2] Our model also exposes a weakest precondition for devices which must also be proven.

References

[1] Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Shaked Flur, Kathryn E. Gray, Prashanth Mundkur, Robert M. Norton, Christopher Pulte, Alastair Reid, Peter Sewell, Ian Stark, and Mark Wassell. 2018. Detailed Models of Instruction Set Architectures: From Pseudocode to Formal Semantics. In *Proc. Automated Reasoning Workshop*. 23–24. Two-page abstract.

[2] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. 2011. Refinement Types for Secure Implementations. *ACM Trans. Program. Lang. Syst.* 33, 2, Article 8 (feb 2011), 45 pages. https://doi.org/10.1145/1890028.1890031

[3] Pedro da Rocha Pinto, Thomas Dinsdale-Young, and Philippa Gardner. 2014. TaDA: A Logic for Time and Data Abstraction. In *ECOOP 2014 - Object-Oriented Programming - 28th European Conference, Uppsala, Sweden, July 28 - August 1, 2014. Proceedings (Lecture Notes in Computer Science)*, Vol. 8586. Springer, 207–231. https://doi.org/10.1007/978-3-662-44202-9_9

[4] Andrew D. Gordon and Alan Jeffrey. 2001. Authenticity by Typing for Security Protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 145–159. https://doi.org/10.1109/CSFW.2001.930143

[5] Armaël Guéneau, Johannes Hostert, Simon Spies, Michael Sammler, Lars Birkedal, and Derek Dreyer. 2023. Melocoton: A Program Logic for Verified Interoperability Between OCaml and C. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 247 (oct 2023), 29 pages. https://doi.org/10.1145/3622823

[6] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. ACM, 637–650. https://doi.org/10.1145/2676726.2676980

[7] Gregory Malecha, Abhishek Anand, and Gordon Stewart. 2020. Towards an Axomatic Basis for C++. In *The 11th Coq Workshop*.

[8] Michael Sammler, Deepak Garg, Derek Dreyer, and Tadeusz Litak. 2020. The high-level benefits of low-level sandboxing. *Proc. ACM Program. Lang.* 4, POPL (2020), 32:1–32:32. https://doi.org/10.1145/3371100

[9] Udo Steinberg. 2023. NOVA Microhypervisor: Feature Update, Talk at FOSDEM 2023.

[10] Udo Steinberg and Bernhard Kauer. 2010. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10)*. Association for Computing Machinery, New York, NY, USA, 209–222. https://doi.org/10.1145/1755913.1755935

[11] Kasper Svendsen and Lars Birkedal. 2014. Impredicative Concurrent Abstract Predicates. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings (Lecture Notes in Computer Science)*, Vol. 8410. Springer, 149–168. https://doi.org/10.1007/978-3-642-54833-8_9

[12] David Swasey, Deepak Garg, and Derek Dreyer. 2017. Robust and compositional verification of object capability patterns. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 89:1–89:26. https://doi.org/10.1145/3133913