Layer concepts involved in the micro-contract from Figure 6 are maximal number of instructions in short branches (`IC_short`), short branch itself and speculation. `IC_short` belongs to ISA. This practically means that ISA is being extended to include this information. It is defined as a number. Since this is primitive concept needed for more complex ones, it should always be enforced by the micro-contract. `Short_branch_eq` is a more complex concept present on the level of ISA. It is defined grammatically as a regular instruction similar to `BEQ` (branch if equal). For the purpose of this example, we use RISC-V-like instruction syntax. However, the concept is general and supports any concrete ISA for which the author can provide corresponding grammar definition. Finally, this concept has to be enforced by the micro-contract when instruction count of the branch is smaller than the maximal number of instructions in short branches.

The `speculation` part represents the processor's ability to speculatively execute instructions. It is clearly a property of the microarchitecture itself. The criteria under which the micro-contract enforces speculation to be turned off is `short_branch_eq`. When `short_branch_eq` is enforced on the ISA level then speculation is suspended. This particular part of the micro-contract illustrates the logical junction between two abstractions without intermingling their internals.

It is also important to note that `sb*` is not completely covered by the micro-contract in Figure 6. However, the criteria for all the branch instructions is considered to be the same. Thus, extension of this micro-contract to cover all the branches would just add simple grammatical definitions for the other five RISC-V branch counterparts (`BNE.s`, `BLT.s`, `BGE.s`, `BLTU.s` and `BGU.s`) while other parts would remain the same. For the sake of convenience, future versions of the micro-contracts framework will provide the support for concise expressions of repeating parts. One of the possible approaches is inheritance of the layer concepts. However, the ontological information contained in a more concise version of the micro-contract is exactly the same as in its extended version.

Additionally, we may need to construct a finely-grained mechanism for tackling security implications of Spectre1.1 further up the computing stack. That is, we may be interested in analyzing the security of our system through reasoning about the high level concept such as cryptographic keys given to the system as variables of a program. To do so, we need to design the chain of micro-contracts so as to cover upper layers such as e.g. assembler, linker, compiler and the programming language itself. The micro-contracts framework embraces writing multiple micro-contracts that chain to each other all the way down to the microarchitecture. Therefore, we eliminate a portion of the complexity of non-trivial computing systems that impedes our reasoning about information security.

5 CONCLUSION

In this paper we introduced the micro-contracts framework as a formal, minimal and modular methodology for enforcing security policies over multiple layers of computing systems. We discussed the building blocks of a micro-contract as well as the internal structure. Afterwards, we described multi-layer composition of multiple micro-contracts and introduced the concept of composability. Finally, we demonstrated the usability of the introduced concepts through the design of the micro-contracts among different layers.

```
information:
    - Any
involved layers:
    - ISA
    - microarchitecture
layer concepts:
    - IC_short:
        belongs to: ISA
        defined as: number
        criteria: Always
    - short_branch_eq:
        belongs to: ISA
        defined as: grammar("BEQ.s rs1, rs2, imm")
        criteria: instruction_count() > IC_short
    - speculation:
        belongs to: microarchitecture
        defined as: property
        criteria: if short_branch_eq then absent
                                      else present
```

Figure 6: An example of micro-contract for mitigating Spectre1.1 attack.

We focused on guarding against timing side channel attacks introduced by strength reduction compiler optimization and Spectre1.1. We envision micro-contracts as a sound methodology for studying cross-layer security implications of both future attacks and their mitigations.

REFERENCES

[1] David Chisnall, Colin Rothwell, Robert N.M. Watson, Jonathan Woodruff, Munraj Vadera, Simon W. Moore, Michael Roe, Brooks Davis, and Peter G. Neumann. 2015. Beyond the PDP-11. *ACM SIGARCH Computer Architecture News* 43, 1 (Mar 2015), 117–130.

[2] Vijay D'Silva, Mathias Payer, and Dawn Song. 2015. The Correctness-Security Gap in Compiler Optimization. *2015 IEEE Security and Privacy Workshops* (May 2015).

[3] Chris Hathhorn, Chucky Ellison, and Grigore Roşu. 2015. Defining the undefinedness of C. *ACM SIGPLAN Notices* 50, 6 (Jun 2015), 336–345.

[4] ISO/IEC JTC1/SC22/WG14. 2011. *Information technology - Programming languages - C.* ISO/IEC 9899:2011. International Organization for Standardization.

[5] M. A. Kinsy, S. Khadka, M. Isakov, and A. Farrukh. 2017. Hermes: Secure heterogeneous multicore architecture design. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 14–20.

[6] Vladimir Kiriansky and Carl Waldspurger. 2018. Speculative Buffer Overflows: Attacks and Defenses. arXiv:cs.CR/1807.03757

[7] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[8] V. Benjamin Livshits and Monica S. Lam. 2005. Finding Security Vulnerabilities in Java Applications with Static Analysis. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14 (SSYM'05)*. USENIX Association, Berkeley, CA, USA, 18–18.

[9] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L. Titzer, and Toon Verwaest. 2019. Spectre is here to stay: An analysis of side-channels and speculative execution. arXiv:cs.PL/1902.05178

[10] Kayvan Memarian, Justus Matthiesen, James Lingard, Kyndylan Nienhuis, David Chisnall, Robert N. M. Watson, and Peter Sewell. 2016. Into the depths of C: elaborating the de facto standards. *ACM SIGPLAN Notices* 51, 6 (Jun 2016), 1–15.

[11] MITRE. 2009. CVE-2009-1897.

[12] MITRE. 2018. CVE-2018-10933.

[13] Marco Patrignani and Deepak Garg. 2017. Secure Compilation and Hyperproperty Preservation. *2017 IEEE 30th Computer Security Foundations Symposium (CSF)* (Aug 2017).

[14] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based information-flow security. *IEEE Journal on selected areas in communications* 21, 1 (2003), 5–19.

[15] Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama. 2013. Towards optimization-safe systems. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13* (2013).

[16] Robert N.M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, and et al. 2015. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. *2015 IEEE Symposium on Security and Privacy* (May 2015).