An Optimized Mitigation Approach for Load Value Injection

The Load Value Injection (LVI) Deep Dive describes how LVI methods against Intel® Software Guard Extensions (Intel® SGX) enclaves can be mitigated by inserting an `LFENCE` instruction after each load to serialize its execution. Rather than individually mitigating all load instructions, it is possible to instead consider all Load+Transmit gadgets, which are pairs of instructions that consist of a load and a subsequent instruction that may transmit the loaded value via a covert channel. All Load+Transmit gadgets are mitigated against LVI if the following property is satisfied:

For all Load+Transmit gadgets in each procedure/function, every path in the control flow graph (CFG) from Load to Transmit is "cut" by at least one `LFENCE` instruction.

Intel has collaborated with industry partners to develop a patch to LLVM/clang that optimally satisfies this property. The patch analyzes the CFG to find all Load+Transmit gadgets, and then uses this information to build a condensed CFG that characterizes the control-flow relationships between the gadgets. An example of a condensed CFG is depicted in Figure 1.

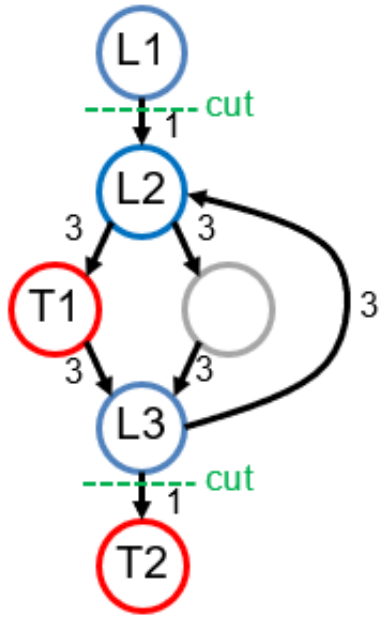

Figure 1. Condensed Control Flow Graph (CFG)

The value loaded by instruction L1 can be transmitted by the instruction denoted as T1, and likewise for L2 and T2. Load L3 has no corresponding transmitter. The edges (black arrows) indicate control flow, and the annotation for each edge is a positive integer that estimates the cost of inserting an `LFENCE` instruction along that edge. Intuitively, inserting an `LFENCE` into a loop is more expensive than inserting it outside of a loop, because any `LFENCE` inside of a loop is likely to be executed more than once.

The cost of inserting `LFENCE` instructions can then be estimated as:

$$\text{Cost} := \sum_{e \in CFG} c_e x_e,$$

where for each edge e, the constant c_e represents the cost of inserting an `LFENCE` along e, and x_e is 1 if the edge receives an `LFENCE` instruction, and 0 otherwise. Recall that the desired security property requires at least one `LFENCE` instruction along every CFG path from each load to each of its corresponding transmits. Hence the next step for the analysis is to find, for each Load+Transmit gadget, all paths in the condensed CFG from the load instruction to the transmit instruction. Once all paths have been found, the requirement that each path must be "cut" by at least one `LFENCE` instruction can be formalized as:

$$\sum_{e \in P} x_e \geq 1$$

for all condensed CFG paths P from L_i to T_i, and for all k gadgets (i.e., $\forall i$, $1 \leq i \leq k$). The cost function together with the constraints on the variables x_e constitute an integer programming optimization problem (specifically, the *minimum multi-cut problem*), the objective of which is to minimize the cost for the LFENCE insertion. For example, this algorithm would make two cuts on the condensed CFG in Figure 1, depicted as green dashed lines. These cuts represent the cheapest solution that suffices to cut all CFG paths from L1 to T1 and from L2 to T2; the cuts indicate the optimal locations to insert LFENCE instructions.

The LLVM tool uses the SYMPHONY solver[1] to obtain the optimal solution, and then inserts LFENCE instructions accordingly. A similar approach has been employed for optimal insertion of fences in the context of high-performance concurrent programming.[2] At the time of this writing, the patch to LLVM can be found here. By default, the patch uses a greedy heuristic that approximates the optimal minimum multi-cut. The SYMPHONY multi-cut optimization is performed by a separate plugin provided by Intel.

Footnotes

1. T. Ralphs, A. Mahajan, S. Vigerske, mgalati13, LouHafer, jpfasano, A. Bulut and anhhz, "coin-or/SYMPHONY: Version 5.6.17," February 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2576603.
2. J. Bender, M. Lesani and J. Palsberg, "Declarative Fence Insertion," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015)*, Pittsburgh, PA, USA, 2015.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel provides these materials as-is, with no express or implied warranties.

No product or component can be absolutely secure.

Intel, the Intel logo, Intel Core, Intel Atom, Intel Xeon, Intel Xeon Phi, Intel® C Compiler, Intel Software Guard Extensions, and Intel® Trusted Execution Engine are trademarks of Intel Corporation in the U.S. and/or other countries.

**Other names and brands may be claimed as the property of others.*