Accelerating Simulation of Future Processors with Compiler and Microcode Assist

Evgeny A. Yulyugin*

*Intel Corporation
Stockholm, Sweden
evgeny.yulyugin@intel.com

Abstract— **Pre-silicon software development approach requires fast simulation of future hardware to run full software stacks far ahead of silicon availability. Wind River(R) Simics(R) virtual platform framework we use relies on Intel(R) VT-x technology to accelerate simulation of Intel targets using Intel host processors. In the case of Intel Control-flow Enforcement Technology (CET), standard Intel VT-x based direct execution technique turned out to be inapplicable due to the nature of the new extension. To achieve usable performance, we created a novel solution based on microcode patching. This approach provided the performance needed to successfully enable pre-silicon work on CET. Simics with microcode assists was used to enable CET support in several important software stacks, including operating systems, compilers and databases.**

Keywords— **Intel Control-flow Enforcement Technology, CET, Simics, software simulation, pre-silicon prototyping, validation.**

I. Introduction

Intel® Control-flow Enforcement Technology (CET) [1] is a new instruction set architecture extension intended to improve software security by making it harder to use Return-Oriented Programming (ROP) [2] and Jump-Oriented Programming (JOP) [3] – the most prevalent attack methodologies for exploit writers targeting vulnerabilities in programs. CET is a typical example of a new hardware feature that needs software support in order to be meaningful to users. To minimize the gap between hardware and software availability, software developers must be shifted left in the product life-cycle to the pre-silicon phase, meaning that a high quality pre-silicon software development environment is required.

Intel published the first version of CET specification in 2016 [1], and hardware support is not yet available at the time of writing. Still, using Wind River® Simics® [4], different teams have been working on software support for CET since 2015. When CET arrives in hardware, major software stacks will have support already in place, allowing users to benefit from new hardware features immediately on release.

Simics is a virtual platform framework commonly used for pre-silicon software development. Simics provides a software-based solution that runs on existing Intel platforms while providing access to features from future platforms. Simics can run real firmware, UEFI, and operating-system code and simulate mechanisms involving both instruction-set and platform-level changes.

In order to be useful for software development, the virtual platform has to be fast. In Simics, a preferred way to achieve high simulation speed is to use Intel VT-x [5] technology to execute instructions directly on the host – known as Simics VMP. In the case of CET, some aspects turned out to be impossible to accelerate with Simics VMP on existing hardware.

To resolve this, we developed a solution that uses microcode patches to existing processors plus minor changes in compilation framework to provide an effective and fast virtual platform solution that can use Simics VMP.

II. Intel Control-flow Enforcement Technology

Intel Control-flow Enforcement Technology (CET) provides the following capabilities to defend against ROP/JOP style control-flow subversion attacks:

- *Shadow Stack* is intended to defend against ROP. It is a second stack that is used exclusively for control transfer operations. This stack is separate from the data stack and can be enabled for operation in user or supervisor mode independently. The pointer to the head of shadow stack is held in a new processor register called SSP. When shadow stack is enabled, the CALL instruction pushes the return address on both the data and shadow stack. The RET instruction pops the addresses from both stacks and compares them. If they do not match, the processor signals a control protection exception. Parameters passed to the CALL instruction are stored only on the data stack as shown in Fig. 1.

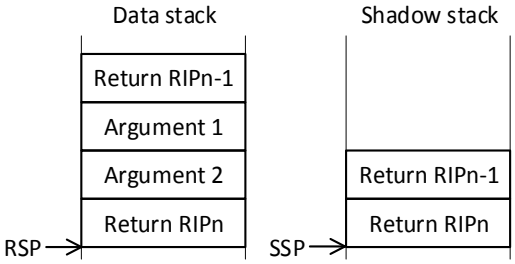

Fig. 1 Stack usage on near CALL instruction

- *Indirect branch tracking* is aimed to defend against Jump/Call Oriented Programming. It introduces special "ENDBRANCH" instructions to mark valid JMP/CALL targets in the program. The instructions reuse subset of NOP (no operation) instruction encodings to ensure that

978-1-5386-4547-5/17 $31.00 © 2017 IEEE
DOI 10.1109/ICEnT.2017.30

programs compiled with ENDBRANCH support will continue working on legacy hardware. The CPU implements a state machine that tracks JMP and CALL instructions so that if there is no matching ENDBRANCH instruction at the target address location, the processor raises a control protection exception (#CP) as show in Fig. 2.

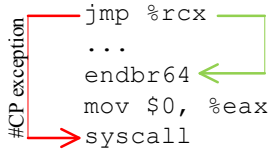

Fig. 2 ENDBRANCH usage

Intel CET involves changes to several parts of an Intel processor core as follows:

- Control register and model specific registers extensions for configuring shadow stack and indirect branch tracking mechanisms.
- XSAVES and XRSTORS instructions to support context-switch of new model specific registers.
- Memory-management unit (MMU) is extended to enforce write protection of the shadow stack such that it is writeable only by stores originating from control transfer instructions like CALL and interrupt vectoring, but not by stores originating from other instructions like MOV, XSAVES, etc. Likewise the MMU enforces loads from the shadow stack originating from control transfer instructions like RET, IRET to only happen from pages marked as shadow stack memory pages.
- Modifications to control transfer flows – CALL, RET, interrupt/exception delivery, SYSCALL, SYSRET, SYSENTER, SYSEXIT, VM entry, VM exit, SMI, RSM, task switch, etc. to support shadow stacks and indirect branch tracking.
- ENDBRANCH new instructions to mark valid CALL/JMP targets in a program.
- New shadow stack management instructions to manage shadow stacks e.g. RDSSP instruction to read the SSP into a general purpose register, INCSSP to increment the shadow stack pointer to perform unwinds, SAVEPREVSSP and RSTORSSP instructions to switch shadow stacks to support user mode threading, etc. RDSSP is encoded using a NOP just like ENDBRANCH, while the other instructions use currently unused instruction encodings.

III. Simulating CET with Simics

To build a virtual platform with CET support, we started with a model of 6th generation Intel Core™ processor as a basis. The developed model was delivered to a major database developer company and to a major operating system provider. It was also used by internal Intel teams for enabling of different types of software including compilers, C standard library, and Linux and Zephyr operating systems.

A. Simics VMP

Simics has three ways to run target instructions: a plain interpreter, a just-in-time (JIT) compiler, and Simics VMP. Simics VMP uses Intel VT-x technology to run Intel 64 code efficiently on host Intel processors. Since the goal of VMP is to accurately simulate a particular instruction set and machine, it mixes VT-x execution with the JIT and interpreter in order to handle instructions not found on the host but present in the target architecture. This implementation relies on getting VM exits for unknown instructions. It also limits VMP to updating the processor and memory state that the host instructions update.

B. Issues with CET and Simics VMP

As discussed above, CET introduces new behavior for existing instructions as well as several new instructions. These changes prevented use of Simics VMP mode and made it impossible to run control flow instructions using the Simics JIT. As a result the simulation ran quite slowly in interpreter mode.

The shadow stack does not exist on current hardware and the instructions that manipulate the stack do not trap in VT-x since they are innocuous instructions. The core problem is that the behavior of existing instructions is redefined, which is fundamentally beyond what can be emulated with Intel VT-x.

The protection of the shadow stack depends on MMU changes that affect all memory accessing instructions. The changes cannot be accurately simulated using existing hardware MMU implementations. The JIT engine could not improve the execution speed of control transfer instructions as well because it was developed with assumption that all memory accesses generated by a particular instruction have the same access rights (which is no longer true for the shadow stack accesses). Instruction that doesn't change control flow still can be simulated in JIT mode.

CET extends behavior of existing control transfer instructions: CALL, RET, SYSCALL, SYSRET, SYSENTER, SYSEXIT, JMP and IRET. These instructions should make additional checks and raise exceptions if operations do not meet criteria as defined in the new architecture, not to mention that these flows rely on and update shadow stack contents. Simics VMP cannot execute that without hardware support.

Encodings of the new instructions also create issues. The ENDBR32 and ENDBR64 instructions are encoded as multi-byte NOPS, as well as the new RDSSP instruction. Such instructions do not trap in VT-x, which means that we cannot run such code using virtualization technology. Other new CET instructions use previously undefined instruction encodings, which means they will raise invalid opcode exception on current hardware, cause a VM exits, and allow for emulation within the context of VMP.

C. Making VMP Support CET on Existing Hardware

In order to provide a fast solution for testing, prototyping, and developing software using CET, a better approach was needed than the slow interpreter. We needed Simics VMP to accelerate most of the execution. To achieve this, a holistic

solution involving Simics, hardware microcode, and the compilation toolchain was designed.

A microcode patch was developed to change the behavior of near CALL and RET instructions, taking additional actions related to CET. CALL now pushes return addresses to two stacks and RET reads addresses from both stacks, compares them, and results in a VM exit if the addresses do not match. The microcode patch was also able to supply the hypervisor with additional information to allow it to more efficiently process these VM exits. This allowed the most frequent and performance sensitive portions of the new CET behavior to be implemented on the old hardware as near-native functionality.

FAR CALL, SYSCALL and other control transfer instructions were modified to cause a VM exit, as they are less frequent and have more complicated behavior that is best to be handled in the regular software simulator versus attempting to emulate their changes in the microcode. Note that such VM exits are not available in VT-x without this patch.

The MMU problem was not actually resolved. The microcode-based setup does not check against attacks that try to modify shadow stack. But the solution is still sufficient to develop and test CET-based software stacks against ROP attacks. This shows that a virtual platform solution does not necessarily have to be complete to be useful. Moreover, anyone who is interested in checking robustness against attacks on the shadow stack can still use slow but precise interpreter.

To simulate RDSSP a different opcode was used in the target code since overriding of a NOP could not be achieved through microcode patch. This opcode replacement meant that a small change to the compilation toolchain supporting the CET was necessary - basically, temporarily emitting place-holder instructions during enablement work. The Simics interpreter mode supports official RDSSP encoding and can be used to test final code.

Together these changes unblock effective simulation (see table 1) of CET in Simics that can be used for enablement of full software stacks including databases and operating systems.

TABLE I
CET SIMULATION SUMMARY IN SIMICS

Instruction or mechanism	Simulation modes (standalone Simics)	Simulation modes (Simics + microcode)
RDSSP instruction	Interpreter, JIT	Interpreter, JIT, alternative opcode for VMP
Other shadow stack instructions	Interpreter, partially JIT	Interpreter, partially JIT, VMP
Near call/return	Interpreter	Interpreter, VMP
Other control transfer instructions	Interpreter	Interpreter, VMP
Shadow stack protection in MMU	Interpreter, JIT	Interpreter, JIT
Endbranch mechanism	Interpreter, JIT	Interpreter, JIT

IV. PERFORMANCE EVALUATION

The microcode patch developed for this work changes performance sensitive instructions – near CALL and near RET. These changes themselves slow down the host significantly, making interpreter-based CET simulation 7.5 times slower. But the patch allows to run simulated code directly on the host CPU using Simics VMP technology.

To estimate speed-up that can be achieved with new CET simulation approach a special test was developed. The test was designed with an idea that near CALL and RET instructions have the most significant performance impact. The speed-up achieved with the microcode assisted simulation is about 98 times compared to interpreter-based simulation running on unmodified host.

V. CONCLUSIONS

The Simics CET implementation has been used by different teams inside and outside of Intel for enabling, development, debugging and testing of different types of software including compiler toolchains, databases and operating systems. The speed achieved by the microcode patch made Simics fast enough to be useful to simulate large software stacks.

The prototype was helpful in proving CET architectural correctness and compatibility with existing software. Compatibility was validated by running existing unmodified applications (designed without knowledge of CET) on simulated systems with CET and the shadow stack enabled, and with an operating system and C runtime modified to support CET. Correctness was validated by testing software in situations where a memory safety bug can lead to control flow attacks, with CET catching those situations.

The technique described in the article extends the standard virtual platform approach used for pre-silicon software development with some minimal changes to existing hardware, in order to efficiently simulate new instruction set architecture extensions. In addition, instructions emitted by the compilation toolchain for certain operations ware temporarily changed so that efficient simulation was achieved.

ACKNOWLEDGMENT

Author would like to thank Barry Huntley and Vedvyas Shanbhogue for the assistance with microcode; Jakob Engblom, Magnus Christensson and Grigory Rechistov for the useful comments on drafts of this article.

REFERENCES

[1] *Control-flow Enforcement Technology Preview*, Intel Corporation, 2016.

[2] M. Prandini and M. Ramilli, "Return-Oriented Programming," in IEEE Security & Privacy, vol. 10, no. 6, pp. 84-87, Nov.-Dec. 2012.

[3] F. Yao, J. Chen and G. Venkataramani, "JOP-alarm: Detecting jump-oriented programming-based anomalies in applications," 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, 2013, pp. 467-470.

[4] D. Aarno and J. Engblom, *Software and System Development using Virtual Platforms – Full System Simulation with Wind River Simics*, Morgan Kaufmann Publishers, 2014.

[5] *Intel® 64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation, 2017.