Toward Efficient Fuzzing of Nested Virtualization

Reima Ishii
The University of Tokyo
Tokyo, Japan

Takaaki Fukai
National Institute of
Advanced Industrial Science and Technology
Tokyo, Japan

Takahiro Shinagawa
The University of Tokyo
Tokyo, Japan

Abstract

Infrastructure-as-a-Service (IaaS), such as Amazon EC2 and Google Compute Engine, provides virtual machines (VMs) to users while protecting their hardware resources by using hypervisors. Since a vulnerability in hypervisors can cause serious problems in resource protection, improving hypervisor security is critical. In addition, recent IaaS providers support nested virtualization, which allows users to run their own *L1 hypervisors* on top of the provider's *L0 hypervisors*, and then run *L2 guest OSs* on the L1 hypervisors. Since nested virtualization is realized by emulating hardware-assisted virtualization functions, such as Intel VT-x and AMD-V, in the L0 hypervisors, the security of the emulation functions is equally important. However, hardware-assisted virtualization functions are complex in specification, and vulnerabilities are often discovered in their emulation implementations.

Fuzzing is generally an effective approach to finding software vulnerabilities, in which random data is repeatedly input into the target software and its execution is monitored to find defects. However, applying fuzzing to nested virtualization poses the following challenges; (1) Appropriate initialization: randomly generated instruction sequences do not complete the initialization of hardware-assisted virtualization functions easily, while executing existing initialization code does not fuzz emulation code sufficiently. (2) Huge VM state space: trying all possible states is impractical, while generating near-valid VM states from random data without prior knowledge is difficult. (3) Various CPU configurations: the CPU functions to be emulated are determined at L0 hypervisor startup, making it difficult to fuzz various emulation codes from VMs.

Several studies have applied fuzzing to hypervisors. However, most of them target the fuzzing of virtual CPUs and virtual devices, and few address the issues of nested virtualization. Syzkaller, a state-of-the-art Linux system call fuzzer, can fuzz the KVM's nested virtualization using system call sequences. However, Syzkaller does not directly address the challenges of nested virtualization; it uses fixed initialization code, randomly generated data as VM state, and a fixed CPU configuration. Thus, it is difficult to improve code coverage.

We propose a novel scheme for efficient fuzzing of nested virtualization. First, to address the challenge of (1) appropriate initialization, we create a *dedicated small harness* that executes a instruction sequence mutated from a template of the correct initialization code, enabling the execution of a variety of nearly correct initialization codes. Second, for the challenge of (2) huge VM state space, we introduce the *VM state validator* that can round randomly generated VM states to valid states, thus efficiently producing VM states on the boundary of valid and invalid states. Finally, for (3) various CPU configurations, we introduce a system that automatically restarts the L0 hypervisor using the fuzzer-generated values as startup parameters.

We implemented the proposed scheme using a standard fuzzer AFL++. The fuzzing target was Intel VT-x nested virtualization in KVM. The dedicated small harness was implemented as a single binary with the L1 hypervisor and L2 guest OS integrated, based on VMXBench. The VM state validator was implemented by porting and bug fixing Bochs' VMCS checker code. Hypervisor parameter fuzzing was implemented by interfacing AFL++ with KVM's module loading scripts. We used kcov to obtain coverage and enabled the sanitizers KASAN, KCSAN, and UBSAN.

We used this implementation to measure code coverage of KVM's nested virtualization and compared it to KVM's selftest and Syzkaller. Experimental results of measuring the coverage of nested.c in KVM showed that the proposed method reached 81.3% in 54 hours, compared to 55.2% for selftest and 69.0% at most for Syzkaller after 564 hours of fuzzing.

As a result of actual fuzzing, we discovered a previously unknown vulnerability in the nested virtualization function of KVM. We reported this vulnerability to the Linux Kernel developers, and a patch was immediately created. The vulnerability was also assigned CVE-2023-30456 and determined to have a Base Score of 6.5 Medium.

In the future, we plan to expand the coverage evaluation to codes other than nested.c. We also aim to apply the proposed scheme to different architectures, such as AMD CPUs, and to hypervisors other than KVM.