# Poster: VIDEZZO: Dependency-aware Virtual Device Fuzzing

Qiang Liu*† <cyruscyliu@gmail.com>, Flavio Toffalini† <flavio.toffalini@epfl.ch>,
Yajin Zhou* <yajin_zhou@zju.edu.cn>, Mathias Payer† <mathias.payer@nebelwelt.net>
* *Zhejiang University,* † *EPFL*

Hypervisors (virtual machine monitors—VMMs) are widely deployed in cloud infrastructure. They transfer data and instructions from guest operating systems to the host environment through virtual devices that are driven by I/O operations (Port I/O—PIO or Memory-Mapped I/O—MMIO). These I/O operations follow specific protocols and thus are called *virtual device messages*.

Virtual devices are the most prominent attack surface in hypervisors. Hypervisors isolate an untrusted guest from the hypervisor and all other virtual machines. A key security property is that a guest cannot escape from its virtual machine. However, hypervisors are complex pieces of software and researchers have discovered ways to escape them (e.g., QEMU, VirtualBox, and VMWare), with 41.5% (22/53) of these escapes due to bugs in virtual devices [1]. According to our CVE survey [2], 57.4% (252/439) of the vulnerabilities in QEMU were found in virtual devices.

The security of virtual devices has been under heavy scrutiny [3], [4], [5], [6], [7], [8], [9], [10]. Since VFD [7] in 2017, fuzzing has become the dominant approach as it implicitly abstracts device complexity through concrete executions, outperforming symbolic approaches. Later, a platform-independent black-box hypervisor fuzzer [8] discovered multiple bugs due to its high throughput and multi-dimension inputs. Fuzzing of virtual devices advanced further when started considering coverage feedback [9] and guest-provided data through DMA channels [9], [10], [11].

Despite hundreds of bugs in virtual devices, existing solutions are limited due to two, so far, overlooked challenges.

**Intra-Message Dependency: a field in a virtual device message may be dependent on another field**. Guests communicate with virtual devices through virtual device messages. Each virtual device message follows a given *message structure* and encodes *message fields* that have different semantics, e.g., a four-byte scalar or a pointer. Particularly, a field may be dependent on another field. For example, a bit in a data field may tell a virtual device the type of pointer field. Virtual device fuzzers unaware of the dependencies are slower in reaching certain code or may even miss critical functionalities.

**Inter-Message Dependency: a message may depend on a previously issued message**. A virtual device message may modify the internal state of a virtual device and can be chained to form a sequence of complex interactions. In a virtual device, two messages might go through different paths but are entangled by the device-internal state, which implies an ordered sequence of messages. Mutators unaware of these dependencies may violate order constraints, which wastes time and hardware resources.

Our goal is to overcome the two challenges and achieve both *scalability and efficiency* in fuzzing virtual devices based on two observations. First, we notice that source code encodes message semantics, serving as a reference for message structures. Widely-used hypervisors (QEMU and VirtualBox) are open-source, encoding abundant information about how to interact with virtual devices. Moreover, source code is amenable to automatic analysis and inherently less labor-intensive to validate than complex specifications. Second, well-formed messages exercise more coverage and provide better feedback to the fuzzer for future mutations.

**Our Approach.** We introduce a new dependency-aware virtual device fuzzing framework VIDEZZO (Virtual Device Fuzzer), which considers both intra-message and inter-message dependencies.

*Lightweight Intra-Message Annotation.* To support intra-message dependencies, we design a novel and lightweight descriptive grammar. When reviewing the source code, a security analyst of a virtual device may record intra-message annotation with our descriptive grammar to allow a fuzzer to know how to handle intra-message dependencies. We argue that our lightweight grammar is a good trade-off between the full grammar implementation from the hardware specification used in NYX-SPEC and the heuristic-based approach used in V-SHUTTLE and MORPHUZZ. We semi-automate the annotation extraction.

*Novel Inter-Message Mutators.* To handle inter-message dependencies, we design three new categories of mutators based on a virtual device message as a mutation atom. These mutators create a single message or form message sequences leveraging the genetic nature of fuzzers to provide consistency (message-level), diversity (sequence-level), and semantics (group-level). These message-aware mutators not only self-learn the inter-message dependencies but also keep the advantages of different mutation granularity.

Based on the above two techniques, we present the design of VIDEZZO in the following. VIDEZZO has two parts: VIDEZZO-CORE and VIDEZZO-VMM bindings. The former manages fuzzing input, parses it into virtual device messages, and processes these messages according to our design. The latter, VIDEZZO-VMM, registers targeted virtual devices, initializes the guest VMM without running any operating system, and dispatches VMM-specific messages.

VIDEZZO-CORE is VMM-agnostic, while VIDEZZO-VMM requires customization for each new VMM. The flexible system design enables the scalability of VIDEZZO.

Importantly, VIDEZZO-CORE enables persistent mode, avoiding a heavy fork server to improve performance. We leverage reflective delta-debugging to address side effects due to the accumulated internal state. Specifically, VIDEZZO stores all intermediate test cases and supports delta debugging [12] to reduce the collected seeds to a minimal stable Proof of Concept (PoC).

Compared to previous work, VIDEZZO is both scalable and efficient. VIDEZZO currently supports two hypervisors, i.e., QEMU and VirtualBox, four architectures, i.e., i386, x86_64, AArch32, AArch64, 28 virtual devices in five device categories, i.e., USB, net, display, audio, and storage, and reaches competitive coverage faster. VIDEZZO is also effective in finding bugs. We successfully *reproduced 24 existing bugs and found 28 new bugs* across diverse bug types with 1 CVE assigned so far. We have been actively engaging with the QEMU and VirtualBox communities and provided 7 accepted patches.

Please scan the following two QR codes to fetch the paper and the source code of VIDEZZO.

(a) Paper (b) Source

# References


[1] W. Contributors, "Virtual machine escape," 2021, https://en.wikipedia.org/wiki/Virtual_machine_escape. Accessed: 2021/10/06.

[2] T. M. Corporation, "CVE search results matched "qemu"," 2021, https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=qemu. Accessed: 2021/10/06.

[3] T. Ormandy, "An empirical study into the security exposure to hosts of hostile virtualized environments," in *CanSecWest*, 2007.

[4] K. Cong, F. Xie, and L. Lei, "Symbolic execution of virtual devices," in *International Conference on Quality Software (ICQS)*, 2013.

[5] T. Yu, X. Qu, and M. B. Cohen, "VDTEST: An automated framework to support testing for virtual devices," in *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2016.

[6] J. Tang and M. Li, "When virtualization encounter AFL," in *Black Hat Europe*, 2016.

[7] A. Henderson, H. Yin, G. Jin, H. Han, and H. Deng, "VDF: Targeted evolutionary fuzz testing of virtual devices," in *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, 2017.

[8] S. Schumilo, C. Aschermann, A. Abbasi, S. Wörner, and T. Holz, "HYPER-CUBE: High-dimensional hypervisor fuzzing," in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[9] ——, "NYX: Greybox hypervisor fuzzing using fast snapshots and affine types," in *USENIX Security Symposium (USENIX Security)*, 2021.

[10] G. Pan, X. Lin, X. Zhang, Y. Jia, S. Ji, C. Wu, X. Ying, J. Wang, and Y. We, "V-SHUTTLE: Scalable and semantics-aware hypervisor virtual device fuzzing," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[11] A. Bulekov, B. Das, S. Hajnoczi, and M. Egele, "MORPHUZZ: Bending (input) space to fuzz virtual devices," in *USENIX Security Symposium (USENIX Security)*, 2022.

[12] renatahodovan, "renatahodovan/picire: Parallel delta debugging framework," 2022, https://github.com/renatahodovan/picire. Accessed: 2022/07/01.