Google Summer of Code 2019 Proposal:

Fuzzing QEMU's Implementation of VirtIO Devices

Alexander Oleinik
a1xndr on irc.oftc.net
alxndr@bu.edu

Summary

I propose to integrate **QEMU** with the **OSS-Fuzz** continuous fuzzing service. In particular, I aim to develop functionality to fuzz QEMU's implementation of devices adhering to the **VirtIO** standard. Fuzzing is a powerful technique for bug finding. The popular American fuzzy lop fuzzer, lists 371 bugs and vulnerabilities in its (incomplete) trophy case, alone. Bugs in VirtIO devices have potential to enable attackers to perform virtual machine escape attacks. QEMU's implementation of VirtIO devices is a particularly appealing target for fuzzing due to their widespread use and VirtIO's clear specification of the driver-device interface. Integration with OSS-Fuzz will improve the chances that dangerous bugs are located and patched, prior to exploitation in the wild.

Problem Description and Proposal

VirtIO is a standardized interface between virtual machine guest drivers and virtual devices on the host. Prior to VirtIO, virtualization developers implemented "emulated" devices, mimicking the functionality of real hardware and relying on existing drivers installed within the guest. Since the limitations facing hardware developers do not always translate into the emulated software models, the guest-host interfaces inherit inefficiencies and overhead which could be avoided. VirtIO solves this issue by defining a specification for straightforward interfaces between virtual devices and corresponding drivers on the guest.

I propose to integrate QEMU's VirtIO implementation with a coverage-guided fuzzing mechanism, such as libfuzzer. Since fuzzing performance depends on the rate of execution, existing tools such as `QTest` may not be sufficient. A successful solution, may depend on a lightweight guest, or rely on a one-time initialization procedure along with compact libfuzzer/AFL targets exercising the `virt-queue` handlers. I will clearly document the system to open pathways for support of additional fuzzing targets. Once fuzzing is integrated into QEMU, the solution can ideally be deployed on Google's OSS-Fuzz service.

Goals

In summary, my goals for this project are:

- Fuzz inputs to at least one QEMU VirtIO Device.

- Optimize the fuzzing setup to achieve an adequate execution rate (>1,000 executions/second)
- Integrate the fuzzing with ClusterFuzz. Ideally, get QEMU accepted into Google's OSS-Fuzz service, and run on their ClusterFuzz instance.
- Find previously unreported bugs, or reproduce a known bug, such as `CVE-2017-5931`.
- Document the overall structure of the fuzzing system, and any code written as part of the project.

Timeline

Here is a twelve week implementation schedule for the project. It does not include specific mention of GSoC obligations, such as phase evaluations:

May 6 - May 27 (Pre-coding period): Grow accustomed to the QEMU code base. Review the VirtIO 1.0 Specification and VirtIO device implementations in QEMU, as well as related standards such as PCI/PCIe and IEEE 802. Familiarize with and observe QEMU's contribution procedure, in action, including the QEMU-devel mailing list, especially patch submissions.

May 27 - June 10: Based on review of VirtIO device implementation and discussions with mentor, pick a VirtIO device(virtio-net) for initial a empts at fuzzing. Fuzz the device using the `QTest` framework, libfuzzer targets, or a guest with a custom driver for con guring and fuzzing across VirtIO communication channels. Gain insight about issues related to execution speed, availability of coverage information, pros/cons of fuzzing with KVM enabled, etc.

June 10 - June 24: Based on the insights gained from the initial proof of concept, integrate guided fuzzing support into QEMU to support fuzzing of VirtIO devices. Ensure that the implementation is compatible with OSS-Fuzz requirements.

June 24 - July 1: Document and thoroughly test the code. Submit a pull request to OSS-Fuzz, requesting that QEMU be added to the service. At this point, we are capable of fuzzing QEMU with our own instance of ClusterFuzz.

July 1 - July 15: Upon a positive response from the OSS-Fuzz team, add configuration for QEMU to OSS-Fuzz (including metadata, Docker setup for the fuzz environment, and a build script). Submit this configuration as a pull-request to OSS-Fuzz. Regardless the response, continue improving the precision and execution rate of the fuzzing procedure.

July 15 - July 29: Ensure that the project is building properly on OSS-Fuzz' clusterfuzz infrastructure, and fuzzing performance matches expectations. Document the integration in QEMU resources such as `docs/` and the wiki.

July 29 - August 26: Explore adding additional fuzzing targets (e.g. libfuzzer harnesses/shims) and update the OSS-Fuzz configs, if necessary. Analyze whether past vulnerabilities, such as `CVE-2017-5931`, could be caught through fuzzing, by fuzzing locally. Deal with crashes reported by ClusterFuzz on a case-by-case basis, with guidance from the mentor. Ensure that documentation is up-to-date.

About Me

I am a first year PhD student doing research in the field of systems security. I use QEMU on a daily basis for kernel debugging, security analysis of embedded devices, and virtual hosting. I have written harnesses for applications, such as proprietary IoT web-servers, to perform input fuzzing. Throughout my research, I have reported multiple vulnerabilities to embedded device vendors, including one which has been assigned `CVE-2018-11106`. I understand the significance of QEMU in today's computing world, and I would be delighted to work towards improving QEMU's security through automated fuzzing.