Investigating the Security of PCIe Passthrough Connections with Fuzzing

Soumil Desai[1,2], Chathura Rajapaksha[2], Ajay Joshi[2]
Syosset High School, 70 Southwoods Road, Syosset, NY 11791[1],
Department of Electrical and Computer Engineering, Boston University, 8 St. Mary's Street, Boston, MA 022152[2]

Introduction

PCIe Passthrough

- PCIe connects a computer's peripheral components, such as network interface cards (NICs), graphics cards, and storage controllers to the host

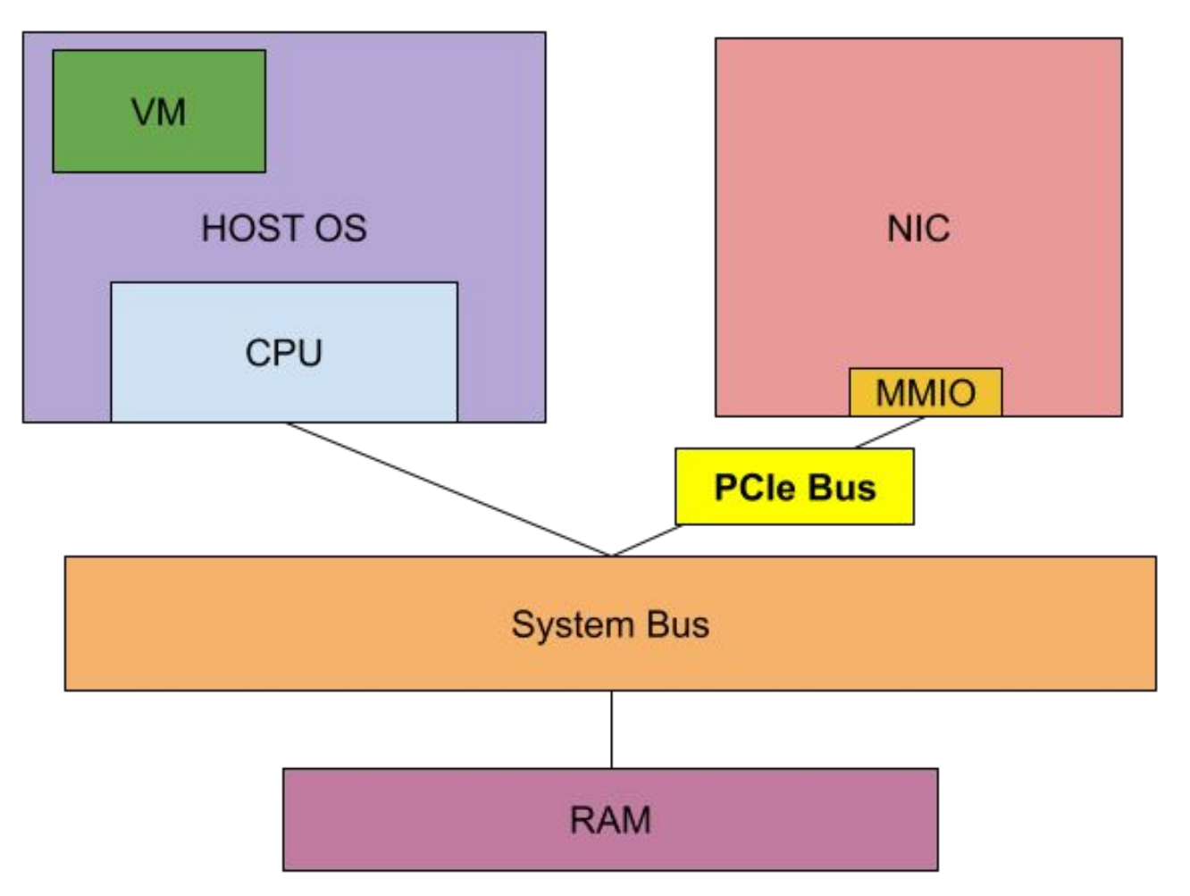

Figure 1: Typical representation of how a peripheral device (NIC) is connected to a host via a regular PCIe connection.

- PCIe passthrough gives VMs direct access to peripheral devices, enabling them to interact as though they are physically attached to the guest

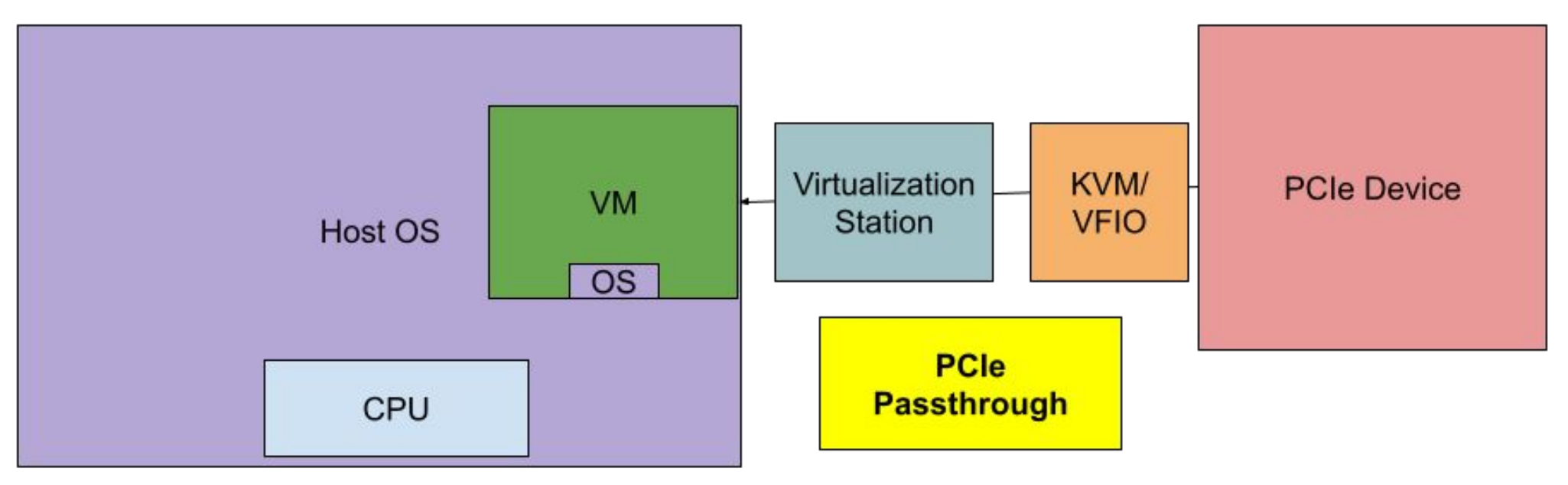

Figure 2: Representation of components in a PCIe passthrough connection. PCIe device is connected to guest directly.

Fuzzing

- Fuzzing is a popular method for identifying bugs in software, but has recently been applied more in hardware security
- Fuzzing involves repetitively feeding random inputs to a program or system in an attempt to find bugs or vulnerabilities through crashes or unexpected outputs
- Coverage-guided fuzzing involves mutating inputs based on the coverage of the previous executions

Problem/Task:

- Although the functionality of the PCIe interface has been verified, there is limited research on the security risks associated with exposing a PCIe device to a VM using PCIe passthrough
- We are testing the isolation of these passthrough connections, and the ability of a malicious VM to affect a host system

Methods

Setup

- Use a QEMU/KVM virtual setup on a machine with the following components (NIC and SSD passed through:
  - CPU: Intel Xeon Silver 4314 16-core CPU (x86)
  - SmartNIC: Mellanox ConnectX-6
  - SSD: 960 GB Samsung PCIe4 x4 NVMe
- We fuzz by writing values to the configuration registers of the PCIe device
  - We are specifically writing to the extended space, which contains 4096 bytes as compared to the 64-byte basic space

Threat Model

- We assume PCIe devices are passed through from host to VM, and the IOMMU is turned on in passthrough mode
- The attacker has root access in VM, and has no control over the host system

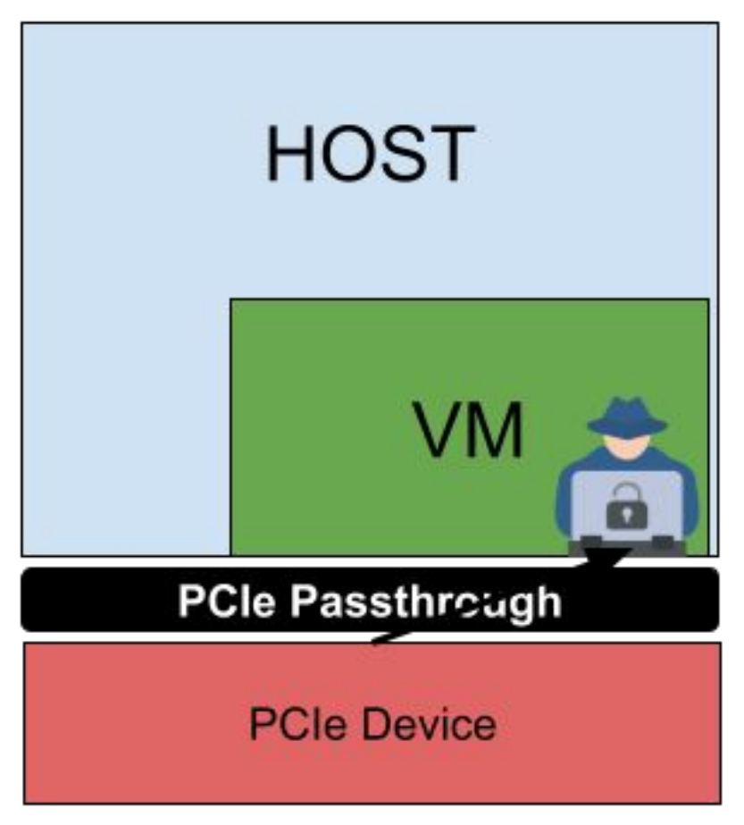

Figure 3: Threat model

Program Enhancements

- Created an exhaustive script tested each possible one-byte value and wrote to each register
  - 4096 extended space registers * 256 possible one byte values = 1,048,576 total writes
- Enhanced random fuzzing script to remove redundancy
  - Kept track of previously written register-value combinations to improve efficiency
  - Created script that automatically detected crashes in the guest system → wrote these values to a file such that these registers can be skipped in later iterations

Fuzzing Process

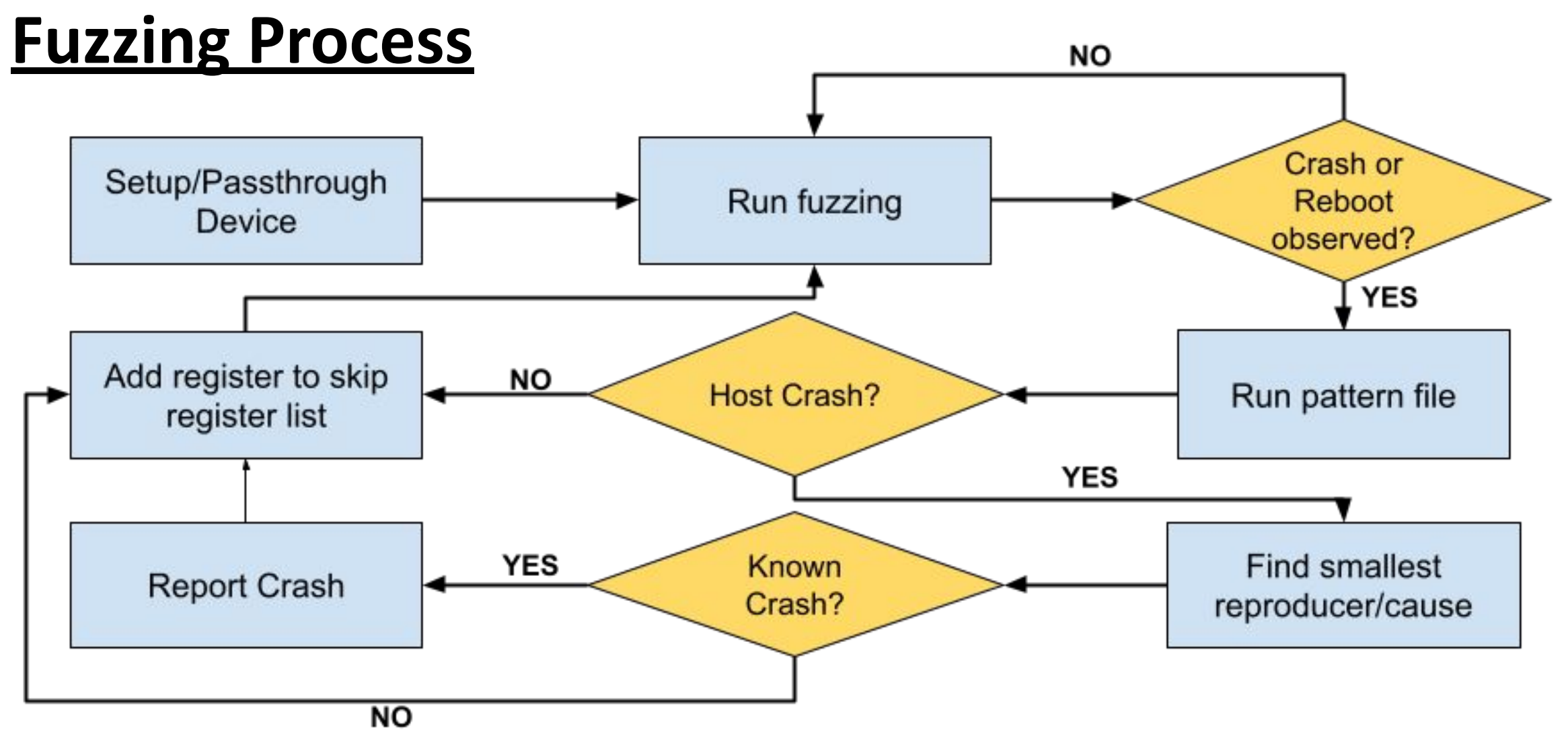

Figure 4: Fuzzing process

All crashes are skipped in future iterations. We find the cause and minimal reproducer for host crashes, and report unknown bugs that cause them.

Results

Fuzzing Results

- The results of fuzzing the NVME storage device are summarized below:

NVME SSD	Random Script	Exhaustive Script
Writes Performed	864,000	170,231
Crashes/Reboots	0	2

- Two crashes were detected while running the exhaustive script on the NVME SSD, on register offset 0x297 and 0x299

SmartNIC	Random Script	Exhaustive Script
Writes Performed	433,871	1,048,576
Crashes/Reboots	0	1 (unreplicated)

- An additional crash was found when fuzzing the NIC, but was not replicated yet

Script Coverage

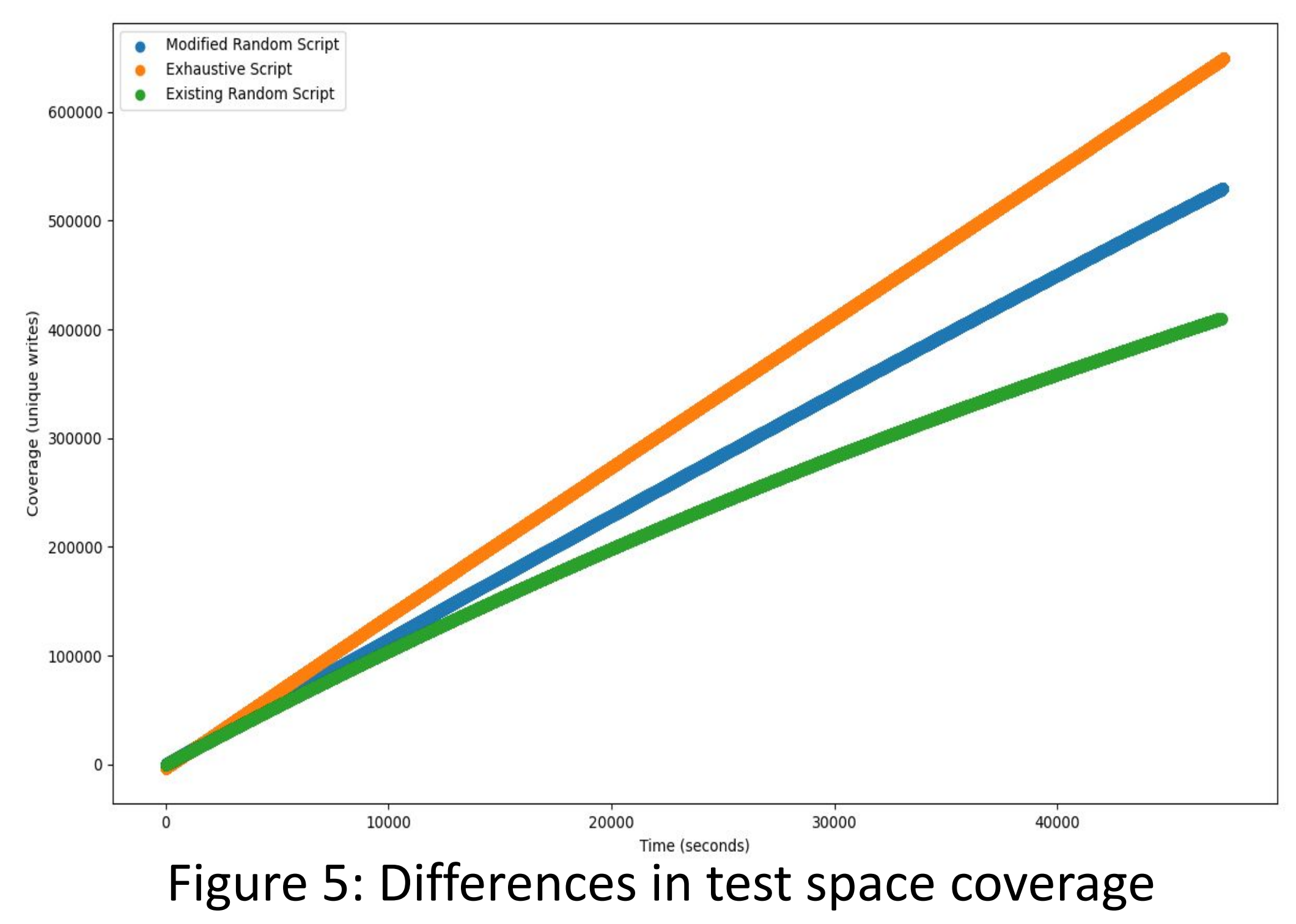

Figure 5: Differences in test space coverage

- Modified Random Script (blue) represents our script, which has removed duplicate values for better efficiency compared to the Existing Random Script (green)
- Exhaustive Script has a perfectly linear trend, as there are no duplicates because numbers are generated serially

Discussion/Conclusions

- After covering just 16.2% of the extended configuration space with the exhaustive script on the NVMe SSD device, 2 replicable reboots were detected
  - As such, the script will be ran to perform the remaining writes in the case that more bugs are detected
- Our programs perform efficiently as shown by the graph
  - As the trends continue over time, our graphs become even more efficient due to the higher frequency of duplicates as runtime increases
- Register writes and test space calculations were performed considering only cases where crashes were caused by one individual write of one byte to a single offset
  - The values of other registers will be taken into account for the full test space in the future
  - Since the PCIe extended configuration space is simply a continuous memory address space, more than one byte can be written at a time, and one register can be larger than one byte
  - There will be no way to exhaustively cover this test space in reasonable time

References

[1] Koteshwara, S., Kariuki, G., Ohmacht, M., & Crumley, P. (n.d.). Fuzzing the PCIe interface of smart devices [Unpublished manuscript], IBM T J Watson Research Center.
[2] Wu, Y., Zhang, T., Jung, C., & Lee, D. (n.d.). DevFuzz: Automatic device model-guided device driver fuzzing, https://www3.cs.stonybrook.edu/~dongyoon/papers/SP-23-DevFuzz.pdf
[3] Hansbrough, J. (2017, February 9). Fuzzing PCI Express: Security in plaintext. Google. https://cloud.google.com/blog/products/gcp/fuzzing-pci-express-security-in-plaintext

Acknowledgements

I would like to thank Dr. Ajay Joshi for the opportunity to conduct this project as part of the Integrated Circuits and Systems Group. In addition, I would like to thank Chathura Rajapaksha for his mentorship throughout the project and constant advice and support. Finally, I would like to thank the Boston University RISE program for the unique opportunity to work at BU.