**ISSN:** 2583-9888
**DOI:** doi.org/10.51219/JAIMLD/priyank-anurag/116

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 1

*Research Article*

# Hardware Fuzzer for the Type-C Host System

**Priyank Rathod* and Anurag**

Intel Corporation Folsom, CA, USA

**Citation:** Rathod P, Anurag. Hardware Fuzzer for the Type-C Host System. *J Artif Intell Mach Learn & Data Sci 2023*, 1(1), 418-420. DOI: doi.org/10.51219/JAIMLD/priyank-anurag/116

**Received:** 01 March, 2023; **Accepted:** 18 March, 2023; **Published:** 20 March, 2023

***Corresponding author:** Priyank Jayantilal Rathod, Intel Corporation Folsom, CA, USA, E-mail: rathodpriyank@gmail.com



## ABSTRACT

A hardware fuzzer, a unique testing tool, is designed to test hardware systems by introducing a variety of invalid, unexpected, or random inputs. It's an extension of software fuzzing, which targets software applications to find issues through unconventional input data. This type of fuzzer is particularly useful in developing embedded systems or crucial systems, including automotive, aerospace, and medical devices. However, it can test any hardware that enables system interfaces. Many systems enabled in their early stages can use this mechanism to see if the enablement has all the supported features enabled. There are many ways to test the system, and such methods can be used to find vulnerabilities and defects in software and hardware-based systems. In the current article, fuzzing Type-C devices involves applying unconventional inputs (test vectors) to test the device's behavior and identify vulnerabilities or defects. It would include sending invalid or unexpected input to the device and observing its behavior to identify potential vulnerabilities. Fuzzing a Type-C device can target various aspects such as the USB protocol, data transfer, power delivery negotiation, and other features.

**Keywords:** Fuzzer, Hardware Fuzzing, Type-C Fuzzing, Device Fuzzer, Systems Fuzzer, Device Under Test for Fuzzer

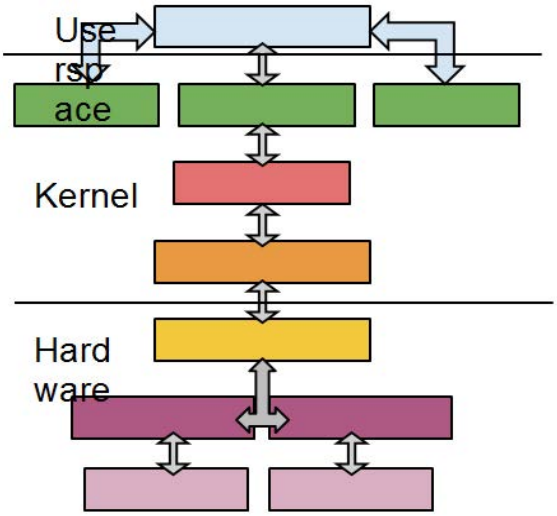
## 1. Introduction

Hardware has become increasingly complex in terms of IP blocks in the last decade. Earlier, Moore's law and Dennard scaling came to a standstill, and hardware engineers had to find novel ways to improve the performance gains of the hardware[4,5]. USB Type-C ports are one main interface to communicate with the system. It supports charging, audio, video, data, and networking over it. Any attacker can control the host machine utilizing devices where the device behavior is altered to exploit. USB software devices and host software stacks are not designed to withstand such exploits as they are uncommon and not widely performed. They require physical access to the system, but having a vulnerability can open many doors for attacks[6]. Many types of USB Type-C devices communicate using standard PD messages. These messages are transmitted using I2C lines between the SOC and PD Controller and then to the device. These messages are communicated to the Host Controller devices on the SoC about the device type connected to the other end, and device descriptors are set up so that the system can enumerate and use the devices. Firmware is loaded into the PD controller to decode these messages and perform certain operations to and from the SoC. These modern attacks on the USB can come from the USB stack to USB specifications. Fuzzing has become the most popular tool for performing runtime testing on software environments and is quickly becoming popular in the hardware space. In many instances, fuzzing has revealed more bugs than automated or manual testing in software stacks[7,8]. Some tools have detected over 50+ bugs in the USB Linux kernel subsystem[9]. All the current mechanisms to mitigate the threats from USB peripherals are focused on defending the host machines and checking for the USB host stack. However, many devices have multiple functionalities, such as devices and host functionalities combined in one device. It would make the stack

more prone to failures and vulnerabilities because two software stacks need to be verified. Most USB Type-C bugs are found around the enumeration rather than in the USB core logic; hence, it has some stability in terms of the working functionality but not much in enumeration, partly because the CC lines and PD messages are doing that work.

## 2. Understanding of USB Stack

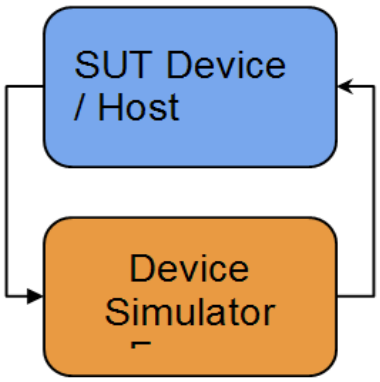
A typical USB Type-C stack has multiple layers. These layers represent the respective part of the OS from where it is operated. User space applications are covered using the HID device types over the Type-C connector. Multiple data types are getting transferred, such as audio, video, and files; hence, different applications use these spaces to make access and operation more accessible for the user.




The next layer in the stack is the kernel layer. Many software stacks are glued together in these layers to make the USB functional. Certain types of devices are classified into different classes, from where they are enumerated and operated further. Almost all USB devices have a core USB functionality where the core specifications of any USB devices are programmed and typically not altered or changed by any specific device type, as classes manage those. The kernel layer is responsible for communication with the hardware or the PHY layer, which uses the devices to be enumerated/detected by the host system.

## 3. USB Software Stack Fuzzing

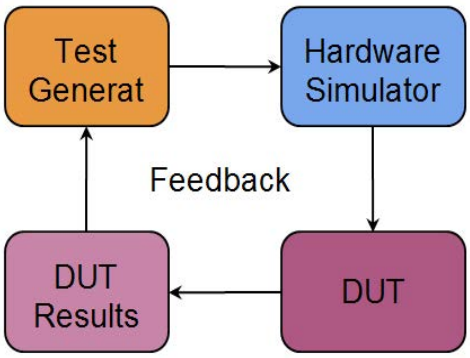
Most USB fuzzers follow the software stack on the host side. Plenty of fuzzers are available to fuzz devices, such as USB devices. Still, there would be scenarios where USB devices are simulated to perform the fuzzing over the stack. In this case, the device would be simulated randomly by a mutation performed in the device class.




Using the mutation on the device class, the host must enumerate or restart the enumeration process once the device resets the connection. In repetition, this process would expose the software stack to issues in the enumeration area where most vulnerabilities are found. Performing such tests in the loop would make the host device more secure regarding the USB core software stack.

### Type-C Fuzzing

Type-C is an extension of the USB stack where all the connectors follow the same USB spec in both cable orientations, making it a unique solution that accommodates different devices. It uses the CC line to connect the host and device with the power and alternate accessory modes as a part of the type-c device specification. Modifying the CC signal using a hardware simulator would change the device detection behavior using fuzzing. During the CC change, the device type messages would be mutated to simulate the virtual fuzzing device. Once the incorrect hardware behavior is recorded and stored, make sure that it will not be repeated in the next iteration of the steps to minimize the testing time. A hardware-based harness would trigger any user-desired actions based on the abnormal operation of the hardware detection. Using such a hardware fuzzer in a Type-C-based system would find PD firmware or other issues related to the power delivery negotiation sooner using the repetition in a continuous loop. Fortunately, Type-C PD communication uses I2C messages to negotiate or detect the devices and can be used to add the signal to the feedback loop.




This virtually mutated device would act as a device to the host system and start the enumeration process. The whole process takes place as a normal process from the host system's point of view. Similar tools such as FuzzUSB combine static analysis and symbolic execution to extract internal state machines from USB gadget drivers and use them to achieve state-guided fuzzing through multi-channel inputs[10]. A similar tool to the earlier one is FirmUSB, which exercises the USB domain knowledge and finds security bugs in the FW of the USB device. Still, it would not attend to the statefulness and suffers from issues such as path explosion[11].

## 4. Related Work

Many software fuzzers support the different types of fuzzers inside the USB stack or any software stack. These software stacks are very good at logging the incorrect behavior of the stack getting fuzzed. Most software-based fuzzers are generic enough with slight modifications to be used by any software stack. Adding a hardware fuzzer is not a new approach but rather unique for the Type-C. Mainly, hardware fuzzers are designed for the application in the mind and custom to the use case. However, having the hardware and software-based fuzzers with the feedback loop would create a closed-loop system.

## 5. Conclusion

Putting the Software and Hardware Fuzzer together is familiar. Having SW and HW fuzzers connected via a feedback loop would curate the following vector of inputs. These would enable most non-repetitive iterations and be productive in finding issues/bugs faster than before, as seen in the software-based fuzzers approach. Such solutions would be scalable but easy to re-configure when needed as they work on a template-based solution.

## 6. References


1. USB type-c ® cable and connector specification. 2019.
2. USB 3.2 specifications. 2017.
3. Class definitions for communication devices 1.2. 2007.
4. Moore GE. Cramming more components onto integrated circuits. Electronics 1998;86.
5. Dennard RH, Gaensslen FH, Rideout VL, Bassous E, LeBlanc AR. Design of ion-implanted mosfet's with very small physical dimensions. IEEE Journal of Solid-State Circuits 1974; 256-268.
6. Kierznowski D, Mayes K. Badusb 2.0: USB man in the middle attacks. Royal Holloway 2016.
7. Peng H, Payer M. USBfuzz: A framework for fuzzing USB drivers by device emulation. 29th USENIX Security Symposium 2020; 397-414.
8. Patrick-Evans J, Cavallaro L, Kinder J. Potus: Probing off-the-shelf usb drivers with symbolic fault injection. 11th USENIX Workshop on Offensive Technologies 2017.
9. Syzkaller. Linux kernel usb bugs found by syzkaller.
10. Kim K, Kim T, Warraich E, et al. FuzzUSB: Hybrid Stateful Fuzzing of USB Gadget Stacks. 2022 IEEE Symposium on Security and Privacy 2022; 2212-2229.
11. Hernandez G, Fowze F, Tian D, Yavuz T, Butler KR. Firmusb: Vetting usb device firmware using domain informed symbolic execution. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security 2017; 2245-2262.