Fuzz The Power: Dual-role State Guided Black-box Fuzzing for USB Power Delivery

Paper #XXX

Abstract

1 Introduction

Since the emergence of USB Type-C (a.k.a., USB-C), numerous peripheral devices have been adopting this new type of USB. Due to its advantage in usability and scalability, USB-C is quickly becoming an alternative to the traditional USB types, such as Type-A and B. USB Power Delivery (USB PD) is an essential functionality available through a USB-C interface. USB PD is capable of supplying more increased power, whose voltage is high enough to be an alternative to traditional power adaptors. In recent years, a newer revision of USB PD comes with much richer functionalities regarding power delivery, such as Extended Power Range (EPR), which can supply up to 240W [].

Unfortunately, such rich functionalities of USB PD essentially increases attack surfaces, security

We observe several unique challenges regarding this. First, USB protocol defines a communication beteween two entities, *host* and *device*. In recent years, peripheral devices that can act as either for their needs.

Although existing USB fuzzers [**?**] are present in the literature, none of them are designed to test USB PD software stack.

In this paper, we present a new USB fuzzer, FUZZPD, which is the first fuzzing technique for USB Power Delivery. FUZZPD operates in a *blackbox* and *plug-and-fuzz* manner, rendering the fuzzing system compatible with all USB PD-capable devices, including proprietary peripheral devices. FUZZPD provides an isolated fuzzing framework that is free of hardware emulation and expensive program analysis techniques (e.g., binary analysis and instrumentation) that are often tedious and error-prone. Furthermore, FUZZPD achieves broad fuzzing coverage, so its execution reaches the *both* ends of USB protocol by swapping the roles on the fly. This is unlike other communication based fuzzers that simply examine either of the two entities solely [1**?**]. We achieve this by leveraging a dual role state machine.

Specifically, FUZZPD takes advantage of dual role state machine from specification. We extract and build a dula-role state machine that allows a fuzzer to fully control both port partners and reach all presented PD functionalities in DUT. Also, we leverage temporal logics to represent contraints and perform checking, which can facilitate

We evaluate FUZZPD from various aspects. We use 5 mobile devices featured with USB PD from multiple vendors. Meanwhile, we have found 15 bugs so far.

The key contributions of this paper are as follows.

- We design USB PD fuzzing system in blackbox mode, which tackles dual-role capable of power and data features.
- We also realize constraint-guided mutation, which allows .
- We evaluate FUZZPD, and discover 15.

2 Background

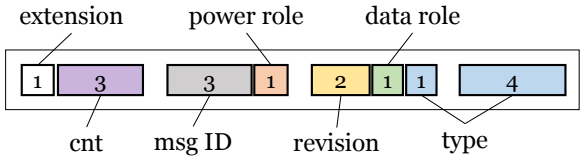

Figure 1: USB Power Delivery header.

2.1 USB Power Delivery

As a new type of USB port connector, USB-C (a.k.a. USB Type-C) [2] is becoming mainstream for a variety of peripherals and smart devices. Unlike previous USB types (e.g., USB-A), USB-C provides enhanced usability — *flip-able* plugs, and *symmetric* cables. Moreover, USB-C is capable of conveying multiple types of data. For example, USB-C delivers not only traditional USB data, but non-USB data, such as HDMI video data.

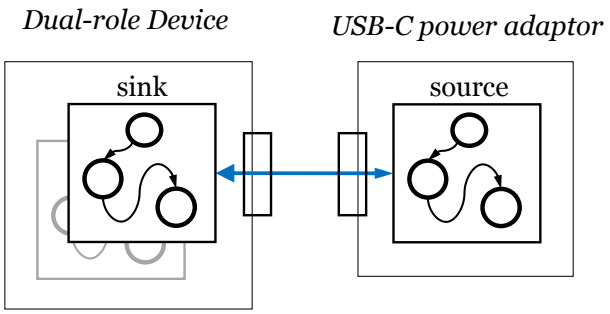

Figure 2: Dual-role USB PD communication.

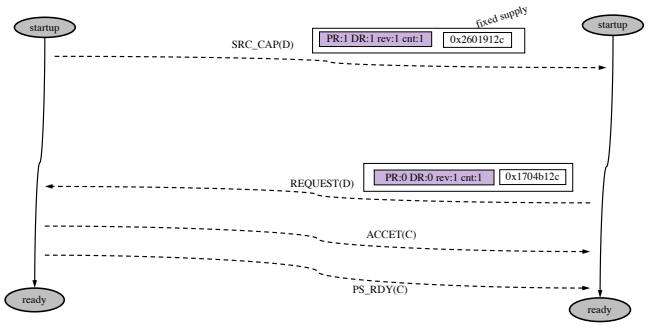

Figure 3: Power contract procedure.

Such enhanced functionalities with USB-C are in fact accomplished through USB Power Delivery (USB PD). USB PD is a recent technology designed for advanced power supply over USB-C. As a new power delivery mechanism, USB PD can provide more charging power (up to 240W[3]) to the connected device via a USB PD-compatible cable. Unlike typical USB protocol, the main tasks of USB PD are categorized into: 1) *power management*, 2) *power/data role control* and 3) *vendor specific data management*. In addition to the power control, USB PD is designed to control multi-typed data communications (e.g., DisplayPort and Thunderbolt) over USB channel, which is delivered as a payload wrapped by *PD messages* [3].

Figure 1 illustrates USB PD header. There are three types of PD messages: *control*, *data*, and *extended* messages.

2.2 Dual Role Devices

Power delivery via USB involves two parties with different roles — *source* discharging the power, and *sink* charging the power. Both power roles operate based on their own state machines that are outlined in the USB PD specification [3]. While USB PD devices can support either role only by nature e.g., USB power adaptors must be source-only, dual-role capable devices are able to serve as both roles (e.g., USB hubs). In particular, they can swap the current role as requested at any time during the communication. For example, a USB source port may become a sink after AC power supply is removed from the device (Figure 2).

Beside the power roles, data roles determine host and device roles for USB data communication. In particular, the data roles have *Downstream Facing Port* (DFP) and *Upstream Facing Port* (UFP). Similar to the power roles, the data roles can be swapped at runtime if the both connected devices support the dual-role data capability.

Technique	**Fuzz Target**	**Fuzz Scope**	**Device Dep**	**HW Emul**
FaceDancer [4]	host	*E* only	HW	✓
POTUS [5]	host	*C* only	SW	Y
vUSBf [6]	host	*E* only	SW	Y
umap2 [7]	host	*E* only	Hybrid	Y
syzkaller [8]	host	*E*	SW	Y
USBFuzz [1]	host	*E*	SW	Y
FuzzUSB [**?**]	device	*E*	SW	Y
FUZZPD	dual	*E*	SW	N

Table 1: State-of-the-art USB fuzzers.

3 Security Model

We assume through a USB-C cable, target machines can be connected to pontential malicious USB-C ports, which are possible

"juice jacking"

We assume attack surface is limited to PD message only because Some other potential attack surfaces, such as host command, are out of scope.

We also note that any damage caused by defective hardware of USB cables or power adaptors are out of scope. This could be a future research direction.

We do not fuzz the payload of USB PD.

We do not consider hardware part or physical layer e.g., signaling.

4 Motivation

As USB PD is growing rapidly in its functionality, there is an urgent need for secure USB PD communications [**?**]. Thus, the USB PD protocol has the potential to increase the device attack surface. Note that although there has been several researches for USB bug finding [1, 4, 6–8], all of them focus on identifying bugs in tranditional USB stacks, which is ineffective against vulneable USB PD stacks. In this work, we aim to design generic USB PD driver fuzzing, finding security faults to secure USB PD stacks. To maximize coverage of USB PD stacks, we address several challenges as follows.

Challenge 1: Lack of runtime information of DUT. Following the standard USB PD protocol, multiple platforms support USB PD capability with their own PD stacks. To

realize generic USB PD fuzzing scheme, we try to fuzz various USB PD stacks across different platforms. Unfortunately, fuzzing proprietary PD stacks (e.g., Windows surfaces and Macbooks) are more challenging. Without source code instrumentation, it is very hard to obtain immediate coverage results for each execution with mutated input. Due to the lack of code coverage information, coverage-guided fuzzing does not work well for proprietary USB PD fuzzing. Some of blackbox fuzzers [**?**] often use binary analysis (e.g., binary instrumentation) to trace executed paths, but such analysis is often inaccurate and error-prune due to the lack of high-level information. Some other work, such as kAFL [9], take benefits from hardware assistance (e.g., Intel PT) for coverage traces. However, it is platform dependent, thus we cannot rely on such features for all different platforms.

Challenge 2: Dual-role USB PD stacks. The target PD drivers can act differently according to the current role and state machine. Clearly, covering both PD stacks can achieve a wide range of path exploration and improve overall coverage, but fuzzing both PD stacks is more challenging because mutation must be different depending on the current role and should support state switching on-the-fly during fuzzing. Due to such complexity, all existing stateful fuzzers [10–14] limit their fuzzing scope to one end, not both ends, to reduce the complexity of analysis and make their solution easily manageable.

Challenge 3: Lack of seed inputs??.

These observations above motivate FUZZPD, to design a tailored testing environment for complex USB PD drivers. We describe our approach in the following.

1: Ineffective random fuzzing.

2: Insufficient stateful fuzzing.

Motivating example. Suppose we want to twist REQUEST msg. If host is src -> need PRSWAP -> (host becomes snk) then we can send REQUEST mutated

5 Design

In this work, we discover security flaws presented in USB PD drivers (e.g., TCPM, Type-C Port Manager) to secure USB PD software stacks. We propose a platform-agnostic and on-the-fly fuzzing framwork, FUZZPD, to find bugs with maximized coverage in USB PD drivers.

5.1 Approach Overview

Figure 4 presents an architecture of FUZZPD.

Solution 1: Feedback-based Blackbox USB PD fuzzing. Apart from open-source USB PD systems, we design our framwork to work for proprietary USB PD stacks, such as Windows surfaces USB PD. We achieve this by leveraging state-guided mutations. Despite closed-source (blackbox) USB PD stacks, we fully leverage statefulness of USB PD communication to generate stateful inputs, rather than naive random input

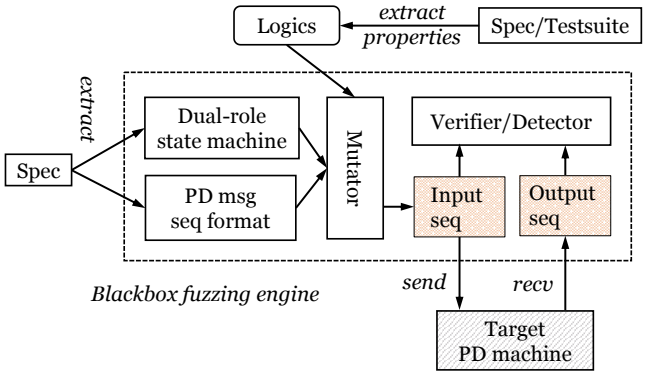

Figure 4: Overview of FUZZPD

generation. Without additional binary (or firmware) analysis, our input mutation works for blackbox USB PD in a platform-agnostic way, which is perfectly driven by state machines retrieved from specifications [3]. Instead of (unavailable) code coverage, we utilize state coverage, which facilitates stateful mutations to reach unexplored states first. The runtime feedbacks from the USB PD driver under test can help keep track of the current state in the working state machine, guiding next stateful input generation to trigger further state transitions.

Solution 2: Dual-role state awareness. In addition to proprietary USB PD fuzzing, we tackle dual-role PD stacks. As each role has its own and independent state machine, we fuzz both of the roles by maintaining dual state machines. Moreover, to handle any role switching during communication, we swap the current state machine to another on-the-fly, depending on the currently working role. Along with dual state machines, we also maintain dual state coverage to guide state transitions in its state machine. By visiting all presented states in both state machines, we extend the state coverage, reaching (and testing) more functional logics in USB PD drivers.

Solution 3: Constraint guidance. In addition to mutation enhancement by stateful guidance, we further improve the ability of mutation guided by logic constraints. We fully leverage

5.2 Dual-role State Machine

Due to operations in a stateful manner, USB PD To realize constraint-guided mutation (see §5.3), we should manage and take controls of state machine.

USB PD specification outlines the state machines that operate throughout the lifetime on each end of the communication. However, such state machines are hardly suited for blackbox fuzzing because most of their states and trasitions are neither under control nor even useful. For example, some of the states rely on interal in a non-deterministic manner. Instead, we abstract away parts of states irrelevant and construct a custom state machine such that we not only have full control over the state transitions, but get straight to the point that fuzzer

Algorithm 1 State Transition Sequence.

Input: G - target gadget code
Input: TF - transition function

```
1:  function StateMerge (S_p)
2:      VP ← {}
3:      S ← {} // initialize a state machine
4:      for each {P_start, P_end, V} in S_p do
5:          for each P in {P_start, P_end} do
6:              if P ∉ VP then
7:                  VP = VP ∪ P
8:                  AddState(P, S)
9:          AddTransition(V, P_start, P_end, S)
10:     return S
11: // entry point: global function calls
12: return S
```

Output: S - a gadget state machine

wants to reach. In particular, our state machine deals with both power roles, and facilitates roles swaps on the fly. This allows us to examine and test all different functions from different roles in the test device, by resolving the challenge discussed in **??**. Thus we fully orchestrate state transitions on demand.

5.2.1 Message Sequence Transitions

We observe that Since message sequences that are controllable, all transitions across states are achieved through a sequence of PD message exchange. We find out all these per-function message sequences from the specification. For example, a power contract procedure obtained is as follows, [SRC_CAP]->[REQUEST]-[ACCEPT]-[PS_RDY].

5.2.2 State Machine Construction

We perform manual extraction on the specification.

5.2.3 State Machine Driving

One essential feature of our state machine is to drive transitions and states towards

5.3 Constraint-guided Fuzzing

5.3.1 Constraint Collection

As discussed **??**, we exercise constraints to better guide the mutation of FuzzPD. Constraints to evaluate are from USB PD specification. We faithfully extract safety and security properties from the specification. We exclude

Power role.

Data role.

Message sequence.

Device	OS	Dual-role	PD rev.
Apple MacBook Pro 13	MacOS Big Sur	✓	2.0
Apple iPad Pro	iPadOS 15	✓	3.0
Microsoft Surface Pro 8	Windows 11	✓	3.0
Samsung Galaxy S21	Android 11	✓	3.0
Xiaomi Mi 11 Lite 5G	Android 11	✓	3.0
Google Pixel 5a	Android 12	✓	3.0
Acer Chromebook Spin 713	ChromeOS	✓	3.0

Table 2: Specification of testing machines used in the experiment.

5.3.2 Runtime Pre-processing

Note that the capability of USB PD functions vary from device to device. For example, USB PD power adaptors are capable of advanced power supply, such as PPS (Programmable Power Supply), whereas PD-compatible smartphones are not. Thus, it seems more reasonable to extensively examine PPS functionality for the USB power adaptor, rather than the smartphones. To maximize the performance of the mutation in this regard, we prioritize the message sequences that are particularly supported by the device under test. To this end, we identify all capable PD functions from the DUT beforehand, which must be completed at runtime due to black box approach. We collect and record all the functions available to the DUT, including power-related functions as well as non-power functions.

5.3.3 Mutation Guided by Constraints

Although stateful mutation approach would be essential in We use for not only for muaation, but for as seed inputs.

5.4 Checkers

Differential checker.

Safety (non-functional) checker.

Liveness (functional) checker.

6 Implementation

Our prototype of FuzzPD is built upon Chromebook Spin 713 []

7 Evaluation

In this section, we evaluate FuzzPD from different angles.

Experimental setup. We assume all the devices under test have at least USB PD 2.0. Table 2 presents USB PD devices.

7.1 Findings

So far, we have found 15 bugs in total.

Fuzzer	Constraint -guided	State -guided
B-fuzzer	✓	✓
FUZZPD	✓	✓

Table 3: Specification of baseline fuzzers.

Out of test smart devices, We observed MS Surface product is resilient to all our fuzz testing and has shown stronger security in USB PD system. For example, against increased voltage... with appropriate a reject message.

Semantic bugs.

Memory bugs.

7.1.1 Security Analysis

Root cause analysis and impact.

Overcharging. In normal procedure of power contract, the power source provides a few available power options to the sink at first. For safety reasons, according to the specification, the first power option out of them must be set to 5V and 1.5A (i.e., 15W, called *vSafe5V*), which is considered as safe power source for any connected devices. However, when mutating this to significantly increased value, some testing devices accept it all without validating its real value. If a malicious connected partner is capable of sending high voltage, then it can damage the target system by overcharging.

7.1.2 Case Study

7.2 Efficiency

7.3 Comparison with Compliance Testsuite

8 Discussion

Fuzzing throughput. However, USB PD communication is more

Additional PD functions. With the fast release of USB PD, more and more functions become available. As mentioned, although our evaluation is based on USB devices as much latest as possible to examine new features, we are unable to test all defined functionalities due to lack of the devices supporting them all, such as Extended Power Range (EPR). We believe we continute to test with richer functional devices in the near future. At the time of submission, we observe USB PD revision 3.1 has been released,

9 Related Work

USB fuzzing technique. In recent years, fuzzing has been enhanced toward USB domain, and it has shown meaningful bug finding results. FuzzUSB [15] is the most recent USB fuzzer. Unlike existing USB host fuzzers, FuzzUSB particularly discovers vulnerabilities from the USB gadget stack under the assumption of malicious USB hosts. Despite massive discovery of security bugs from the test gadgets, FuzzUSB limits its fuzzing scope, lacking USB PD stacks, and is unable to deal with dual roles during the fuzzing campaign. USBFuzz [1] designs a USB fuzzing system by emulating USB hardware.

Stateful fuzzing.

10 Conclusion

References

[1] H. Peng and M. Payer, "Usbfuzz: A framework for fuzzing usb drivers by device emulation," in *25th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 397–414.

[2] "Usb type-c® cable and connector specification," USB Implementers Forum https://www.usb.org/usb-type-cr-cable-and-connector-specification, 2019.

[3] "Usb pd specifications," USB Implementers Forum https://www.usb.org/document-library/usb-power-delivery, 2021.

[4] T. Goodspeed and S. Bratus, "Facedancer usb: Exploiting the magic school bus," in *Proceedings of the REcon 2012 Conference*, 2012.

[5] J. Patrick-Evans, L. Cavallaro, and J. Kinder, "Potus: Probing off-the-shelf usb drivers with symbolic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[6] S. Schumilo, R. Spenneberg, and H. Schwartke, "Don't trust your usb! how to find bugs in usb device drivers," *Blackhat Europe*, 2014.

[7] NCCGroup, "Umap2," https://github.com/nccgroup/umap2.

[8] D. Vyukov, "Syzkaller," https://github.com/google/syzkaller, 2015.

[9] S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, "kafl: Hardware-assisted feedback fuzzing for os kernels," in *Proceedings of the 26th USENIX Security Symposium (Security)*, Vancouver, BC, Canada, Aug. 2017.

[10] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Pulsar: Stateful black-box fuzzing of proprietary network protocols," in *Proceedings of the International Conference on Security and Privacy in Communication Systems (SecureComm)*. Springer, 2015, pp. 330–347.

#	Device	Bug type	Desc.	Msg seq	Status
1	Galaxy S21	Spec violation	Rev. violation	get status	Confirmed
2	Galaxy S21	Spec violation	Rev. violation	get battery cap	Confirmed
3	Galaxy S21	Spec violation	Rev. violation	get battery status	Confirmed
4	Galaxy S21	Spec violation	Rev. violation	get manifacturer info	Confirmed
5	Galaxy S21	Spec violation	Rev. violation	alert	Confirmed
6	Galaxy S21	Spec violation	Increased voltage	PW contract	In preparation
7	Pixel 5a	Spec violation	Rev. violation	alert	In preparation
8	Pixel 5a	Spec violation	Rev. violation	get status (STATUS)	In preparation
9	Macbook Pro	Spec violation	Max and min voltage inversion	PW contract	Reported
10	Macbook Pro	Spec violation	Increased voltage	PW contract	Reported
11	Mi 11 lite	Spec violation	Rev. violation	get src cap extension	In preparation
12	Mi 11 lite	Spec violation	Rev. violation	get battery status	In preparation
13	Mi 11 lite	Spec violation	Increased voltage	PW contract	In preparation
14	iPad	Spec violation	Increased voltage	PW contract	Reported
15	iPad	Spec violation	Max and min voltage inversion	PW contract	Reported

Table 4: List of previously unknown bugs discovered by FUZZPD.

[11] C. Aschermann, S. Schumilo, A. Abbasi, and T. Holz, "Ijon: Exploring deep state spaces via fuzzing," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[12] J. De Ruiter and E. Poll, "Protocol state fuzzing of tls implementations," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 193–206.

[13] H. Han and S. K. Cha, "Imf: Inferred model-based fuzzer," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, Oct.–Nov. 2017.

[14] E. B. Yi, H. Zhang, K. Xu, A. Maji, and S. Bagchi, "Vulcan: Lessons in reliability of wear os ecosystem through state-aware fuzzing," in *Proceedings of the 18th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2020.

[15] K. Kim, T. Kim, E. Warraich, B. Lee, K. R. Butler, A. Bianchi, and D. J. Tian, "Fuzzusb: Hybrid stateful fuzzing of usb gadget stacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.