IOTFUZZER: Discovering Memory Corruptions in IoT Through App-based Fuzzing

Jiongyi Chen , Wenrui Diao , Qingchuan Zhao , Chaoshun Zuo , Zhiqiang Lin,
XiaoFeng Wang , Wing Cheong Lau , Menghan Sun , Ronghai Yang, Kehuan Zhang

Presented by Sezana Fahmida

Outline

- Introduction
- Background
- Challenges
- Scope & Assumptions
- Design
- Implementation & Evaluation
- Discussion
- Conclusion

Introduction

- Internet of Things (IoT) dominating the global market
- IoT devices is projected to reach 20.4 billion in 2020, forming a global market valued $3 trillion
- smart plugs, smart door locks, smart bulbs etc
- 2014 to 2016, 90+ independent IoT attack incidents
- Targets implementation flaws within a device’s firmware

Background

Typical IoT architecture

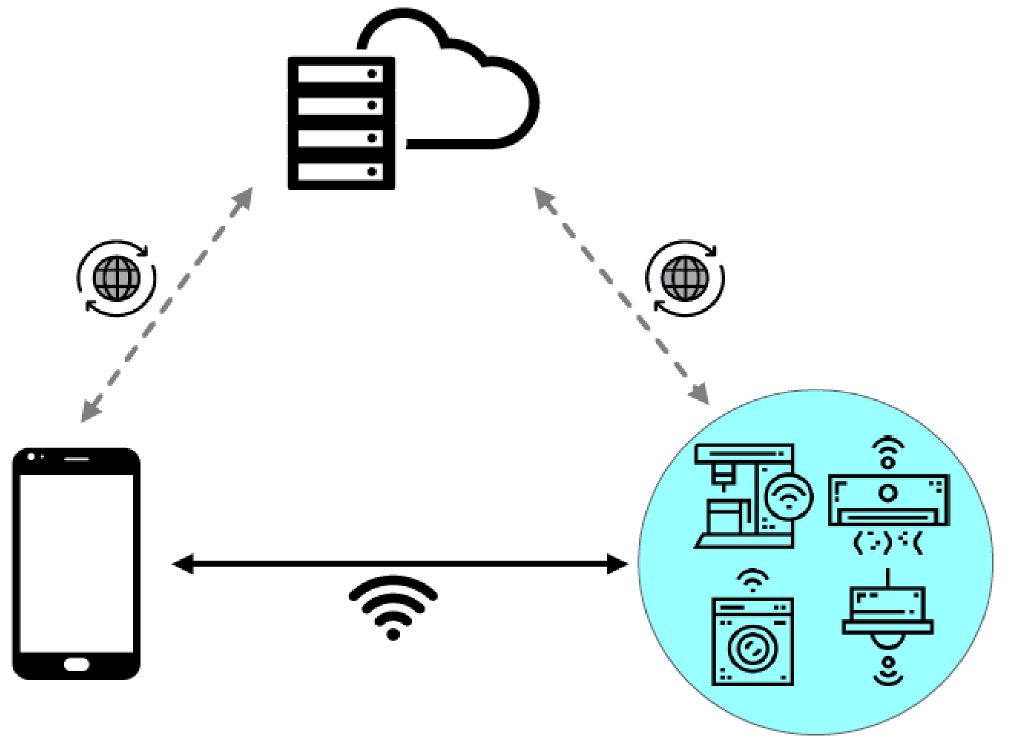


Fig. 1: Typical smart home communication architecture

Typical IoT architecture

- Devices equipped with sensors
- Wireless Connection
- IoT app to control devices provided by vendors
- Communication mode between app and device can be
  - Direct (wifi/Bluetooth)
  - Delegated (via a cloud server)

Obstacles in Firmware Analysis

- Firmware: Special software providing
  - System control
  - Status monitoring
  - Data collection
- Highly customized to fit device architecture
- Main Challenges
  - Firmware Acquisition
  - Firmware Unpacking
  - Executable Analysis

Motivation

- Skip direct firmware analysis by alternative approach
- Intuition: Leverage IoT apps to find vulnerabilities
- Advantages:
  - No need for firmware analysis
  - Avoids reverse engineering binary executables
  - Feasable: Most IoT devices use app
- Design goal: generate protocol-guided and cryptographic consistent fuzzing messages from IoT apps to find memory corruption

Challenges in IoTFuzzer Design

- Mutating fields in networking messages
  - Device specific protocols are used
- Handling encrypted messages
  - Communication between app and device encrypted
  - Code obfuscation
  - Increases complexities
- Monitoring crashes
  - Cannot locally monitor the running process in the system

Solutions

- Mutating fields in networking messages
  - Mutate data at the source
- Handling encrypted messages
  - Reusing cryptographic functions at runtime
- Monitoring crashes
  - Use heartbeat mechanism

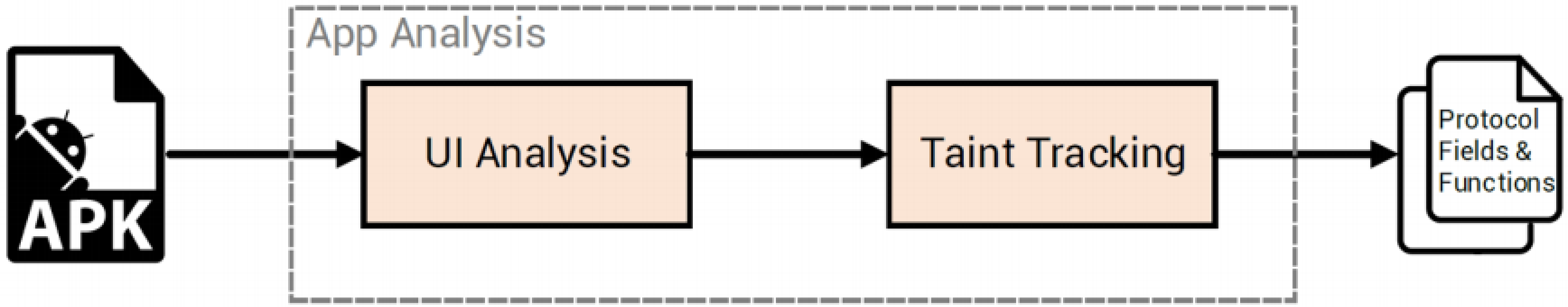
Scope & Assumption

- IoT devices with apps
- Communication channel: Wifi
- Direct Connection , No cloud server
- Android platform

IoTFuzzer Design

- Two phases
- App analysis
  - UI analysis
  - Data Flow analysis
- Fuzzing
  - Runtime mutation
  - Response Monitoring

App Analysis



Picture taken from author's slides

App analysis

- UI analysis
  - Static analysis of apk
  - determine the UI elements that eventually lead to the message delivery
  - from the target network communication APIs construct the backward code paths to UI event handlers
  - Activity transition graphs: To find the order of events

App analysis

- Data flow analysis
  - to recognize the protocol fields and record the functions that take these arguments
  - Dynamic taint tracking
  - Taint source: string, system API, user input
  - Taint sink: networking API and encryption functions

Fuzzing

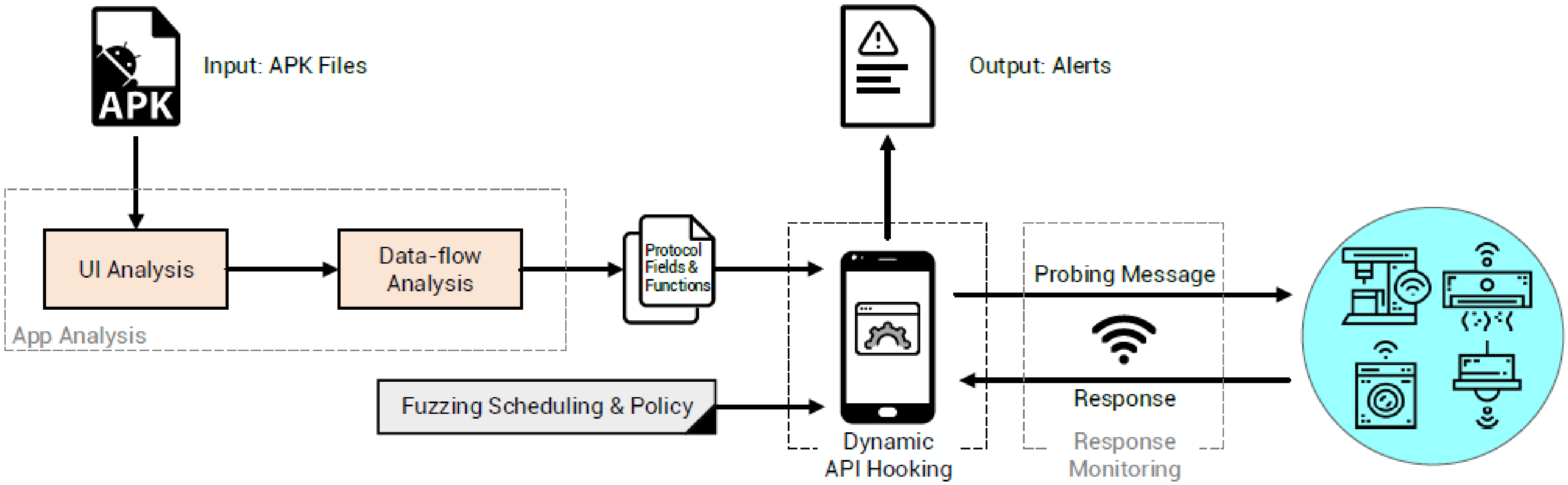


Fig. 2: Overview of IoTFuzzer

Fuzzing

- Runtime Mutation
  - Dynamic Function Hooking
  - Intercept function calls and mutate the fuction arguments
  - Fuzzing Scheduling
  - Only mutate a subset of function parameters
  - Fuzzing policy
    - Changing the lengths of strings
    - Changing the integer, double or float values
    - Changing the types or provide empty values

Fuzzing

- Response monitoring
- Device status inferred from IoT device responses
  - Expected Response
  - Unexpected Response – Error is triggered
  - No Response - Error may be triggered
  - Disconnected –System crash

Fuzzing

- TCP-based connection: look for disconnection
- UDP-based connection: send heart-beat message from app

Implementation

- 17 representative IoT devices from different categories

TABLE I: Summary of IoT Devices under Testing

Device Type	Vendor	Device Model	Firmware Version	Official Mobile App (Android[1])	Protocol and Format (Encrypted: Yes/No)
IP Camera	D-Link	DCS-5010L	1.13	`com.dlink.mydlinkmyhome`	HTTP, K-V Pairs (N)
Smart Bulb	TP-Link	LB100	1.1.2	`com.tplink.kasa_android`	UDP, JSON (Y)
	KONKE	KK-Light	1.1.0	`com.kankunitus.smartplugcronus`	UDP, String (Y)
Smart Plug	Belkin	Wemo Switch	2.00	`com.belkin.wemoandroid`	HTTP, XML (N)
	TP-Link	HS110	v1_151016	`com.tplink.kasa_android`	TCP, JSON (Y)
	D-Link	DSP-W215	1.02	`com.dlink.mydlinkmyhome`	HNAP, XML (N)
Printer	Brother	HL-L5100DN	Ver. E	`com.brother.mfc.brprint`	LPD & HTTP, URI (N)
NAS	Western Digital	My Passport Pro	1.01.08	`com.wdc.wd2go`	HTTP, JSON (N)
		My Cloud	2.21.126	`com.wdc.wd2go`	HTTP, JSON (N)
	QNAP	TS-212P	4.2.2	`com.qnap.qmanager`	HTTP, K-V Pairs (N)
IoT Hub	Philips	Hue Bridge	01036659	`com.philips.lighting.hue`	HTTP, JSON (N)
Home Router	NETGEAR	N300	1.0.0.34	`com.dragonflow`	HTTP, XML (N)
	Linksys	E1200	2.0.7	`com.cisco.connect.cloud`	HNAP, XML (N)
	Xiaomi	Xiaomi Router	2.19.32	`com.xiaomi.router`	HTTP, K-V Pairs (N)
Story Teller	Xiaomi	C-1	1.2.4_89	`com.xiaomi.smarthome`	UDP, JSON (Y)
Extension Socket	KONKE	Mini-K Socket	sva.1.4	`com.kankunitus.smartplugcronus`	UDP, String (Y)
Humidifier	POVOS	PW103	v2.0.1	`com.benteng.smartplugcronus`	UDP, String (Y)

Remarks: All IoT apps mentioned in this table could be obtained from Google Play.

Evaluation

- 15 serious vulnerabilities (memory corruptions) in 9 devices.

TABLE II: Summary of Discovered Vulnerabilities

Device	Vulnerability Type	# of Issues	Remotely Exploitable?
Belkin WeMo (Switch)	Null Pointer Dereference	1	No
TP-Link HS110 (Plug)	Null Pointer Dereference	3	No
D-Link DSP-W215 (Plug)	Buffer Overflow (Stack-based)	4	Yes
WD My Cloud (NAS)	Buffer Overflow (Stack-based)	1	Yes
QNAP TS-212P (NAS)	Buffer Overflow (Heap-based)	2	Yes
Brother HL-L5100DN (Printer)	Unknown Crash	1	Not determined
Philips Hue Bridge (Hub)	Unknown Crash	1	Not determined
WD My Passport Pro (NAS)	Unknown Crash	1	Not determined
POVOS PW103 (Humidifier)	Unknown Crash	1	Not determined

Evaluation

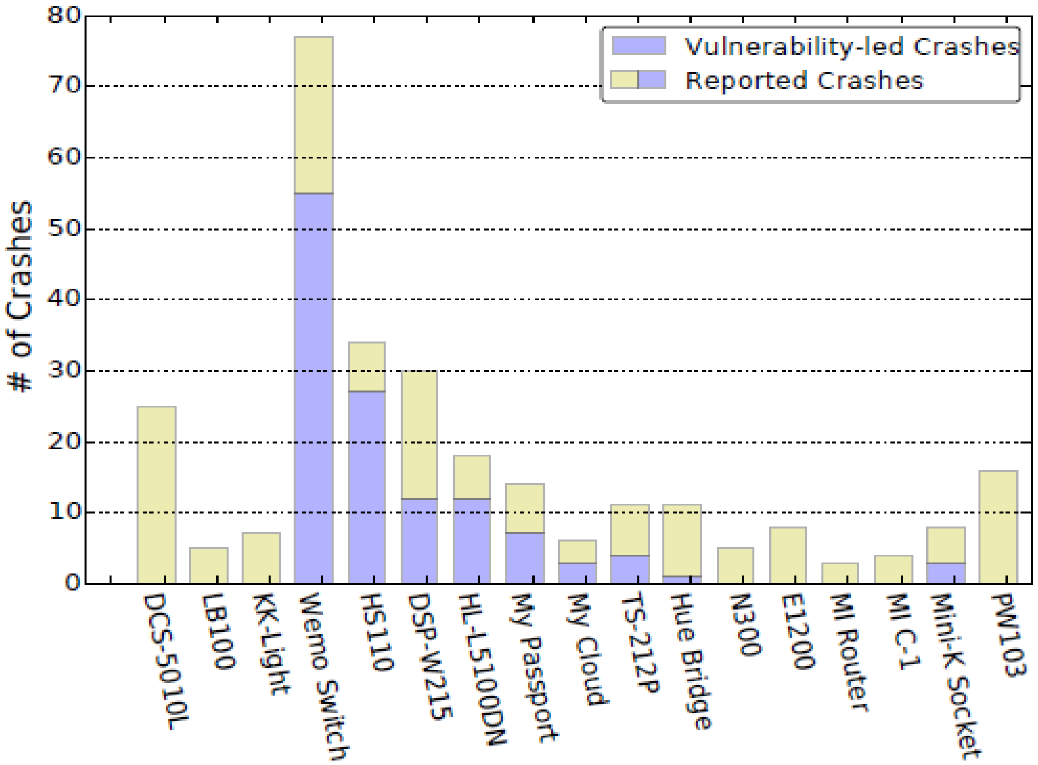


Fig. 5: Fuzzing Accuracy

Discussion

- Provides high specification coverage, low code coverage
- Does not consider cloud relay
- cannot generate memory corruption types and root causes directly
- final vulnerability confirmation always requires some kinds of manual efforts.
- False positives & negatives

Conclusion

- IoTFuzzer- first IoT fuzzing framework
- Protocol guided fuzzing achieved without protocol specifications

THANK YOU!!!