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

Jiongyi Chen[1], Wenrui Diao[2], Qingchuan Zhao[3], Chaoshun Zuo[3], Zhiqiang Lin[3,4], XiaoFeng Wang[5], Wing Cheong Lau[1], Menghan Sun[1], Rongai Yang[1], and Kehuan Zhang[1]

Chinese University of Hong Kong[1], Jinan University[2], University of Texas at Dallas[3], Ohio State University[4], Indiana University Bloomington[5]

NDSS 2018

Presented By

Md Mahbubur Rahman

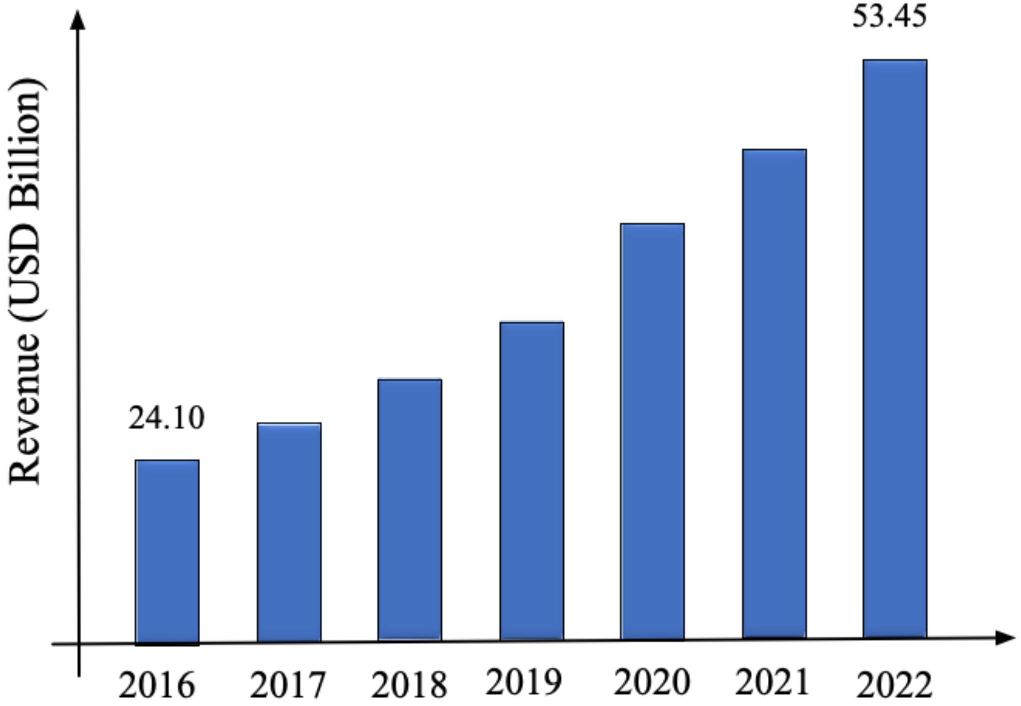
Wayne State University

Outline

- IoT Trend
- Motivation
- IoTFuzzer (This paper)
- Challenges
- Architecture: IoTFuzzer
- Implementation and Evaluation
- Conclusion

Internet of Things (IoT) Market

- Applications
  - Smart Home, Smart City, Agricultural IoT, etc.
- Market growth by 2020
  - 20.4 billion IoT devices
  - $3 trillion
- Smart Home
  - $53.45 billion by 2022



Smart Home market value
(Source: Zion Research Analysis 2017)

Is IoT Secure?

- NOT really!
- Attacks: 2014-2016
  - Mo
- Mirai b
  - Onl
  - Dist
- Reaper botnet attack

Firmwares of the IoT devices are not properly implemented & protected!!

What’s Done!

- Few attempts have been made that closely deal with firmwares .

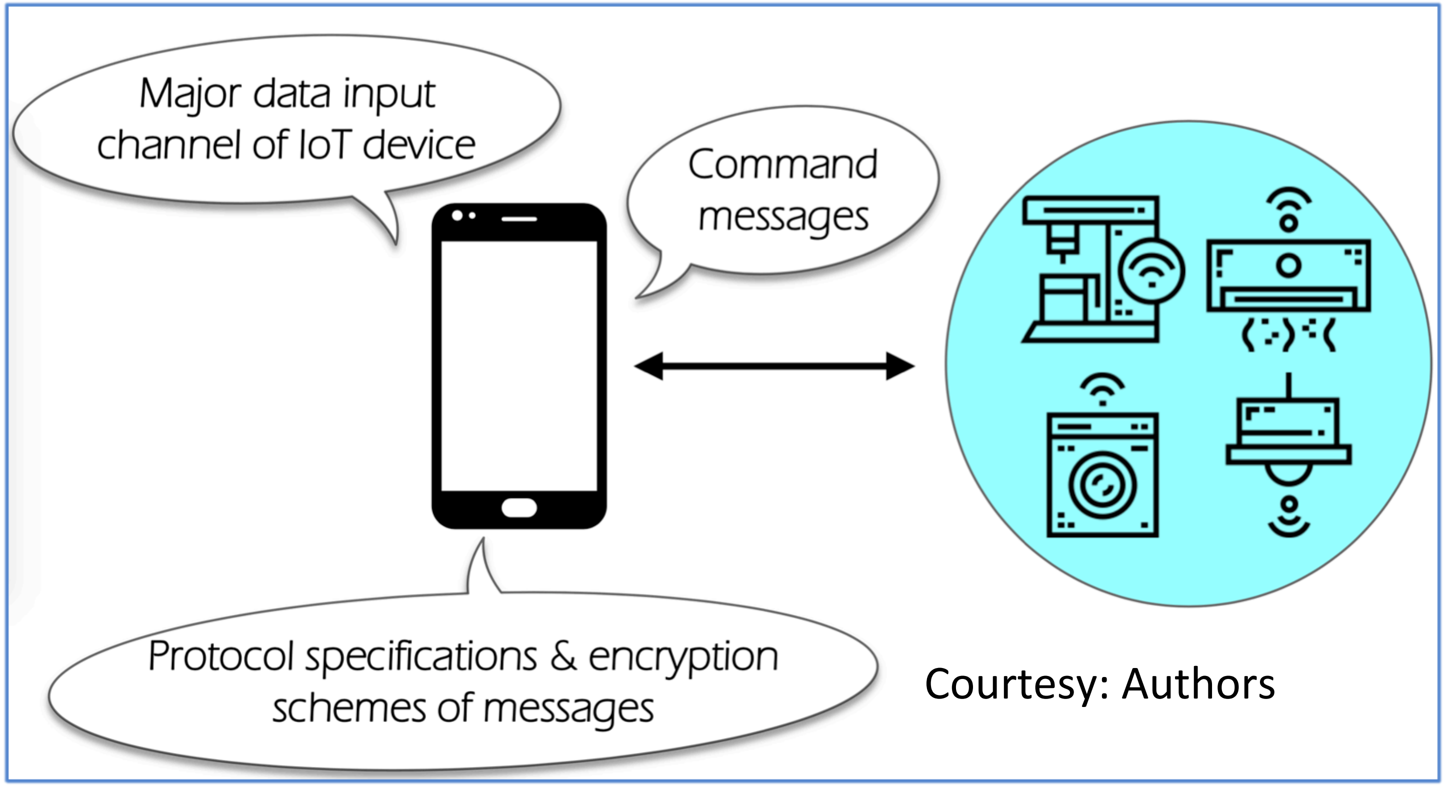
  [Davidson et al. USENIX Sec.’13, Cui et al. NDSS’13, Chen Black Hat’09, Shoshitaishvili et al. NDSS’15]

- Limitati
  - Firm
  - Firm npression/ encr
  - Executable analysis: requires lots of manual efforts and is not accurate

It is worth looking into the IoT official applications

IoT Official Application

- Controls and manages IoT applications



Major data input channel of IoT device
Command messages
Protocol specifications & encryption schemes of messages

Courtesy: Authors

Contains rich information about the IoT system

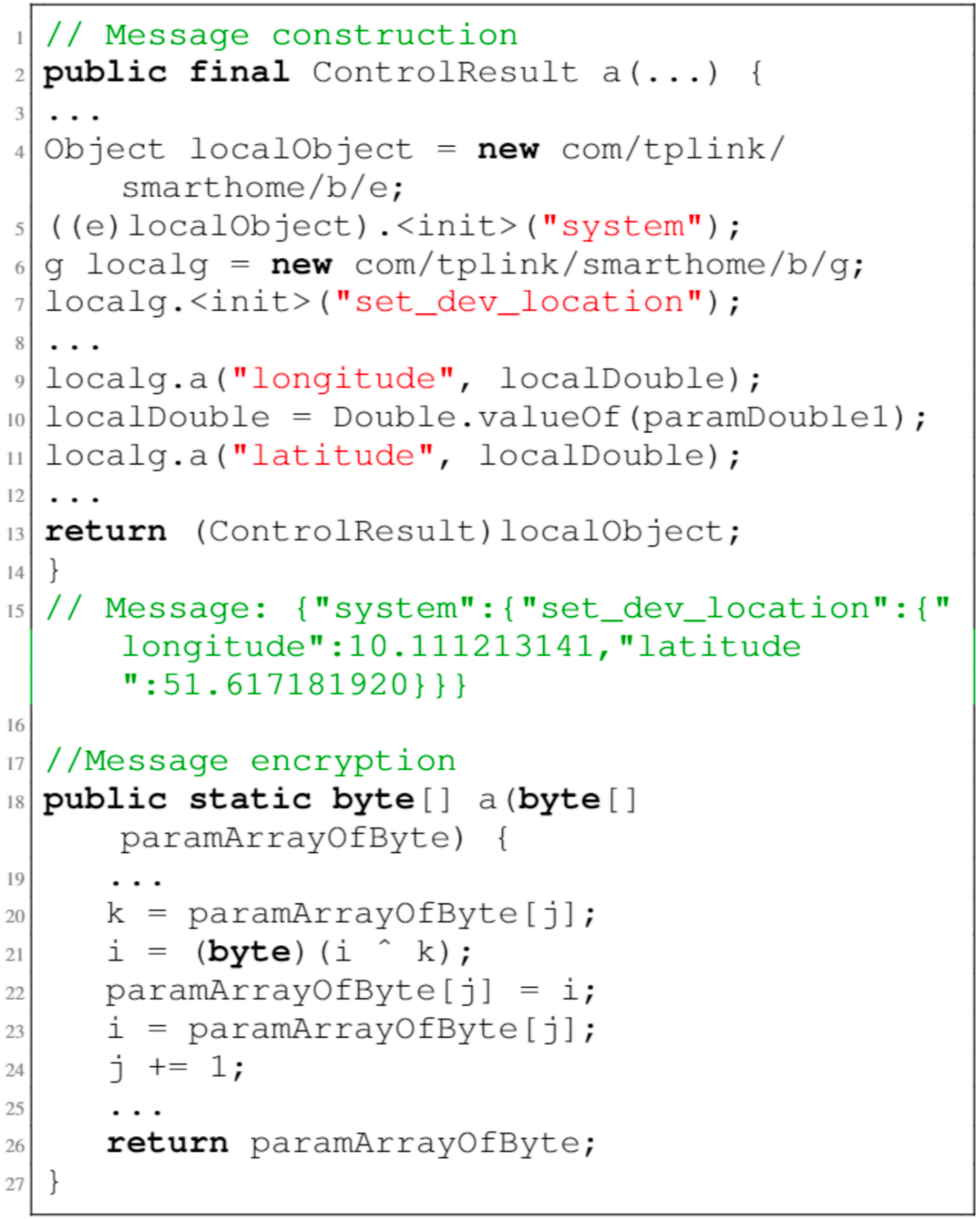
IoTFuzzer: A Firmware-free Fuzzing Framework

- Detects memory corruptions in IoT devices
    - Null-pointer exceptions, buffer overflow, out-of-bound accesses, etc.
- Leverages official apps and program logics to create meaningful test messages
- Fuzzes in a protocol-guided way without explicitly reverse engineering the protocols

IoTFuzzer: Challenges

- Diverse data formats and protocols
  - XML, JSON, key-value pairs
- Proprietary cryptographic functions
- Crash monitoring
  - How to determine the real-time status of the device?

TP-Link Kasa Code Snippet



```
// Message construction
public final ControlResult a(...) {
...
Object localObject = new com/tplink/
    smarthome/b/e;
((e)localObject).<init>("system");
g localg = new com/tplink/smarthome/b/g;
localg.<init>("set_dev_location");
...
localg.a("longitude", localDouble);
localDouble = Double.valueOf(paramDouble1);
localg.a("latitude", localDouble);
...
return (ControlResult)localObject;
}
// Message: {"system":{"set_dev_location":{"
    longitude":10.111213141,"latitude
    ":51.617181920}}}

//Message encryption
public static byte[] a(byte[]
    paramArrayOfByte) {
    ...
    k = paramArrayOfByte[j];
    i = (byte)(i ^ k);
    paramArrayOfByte[j] = i;
    i = paramArrayOfByte[j];
    j += 1;
    ...
    return paramArrayOfByte;
}
```

IoTFuzzer: Solutions

- Diverse data formats and protocols
  - Mutate protocol fields before they are constructed as message
- Proprietary cryptographic functions
  - Reuse cryptographic functions in the runtime
- Crash monitoring
  - Insert heartbeat messages

IoTFuzzer: Scope and Assumptions

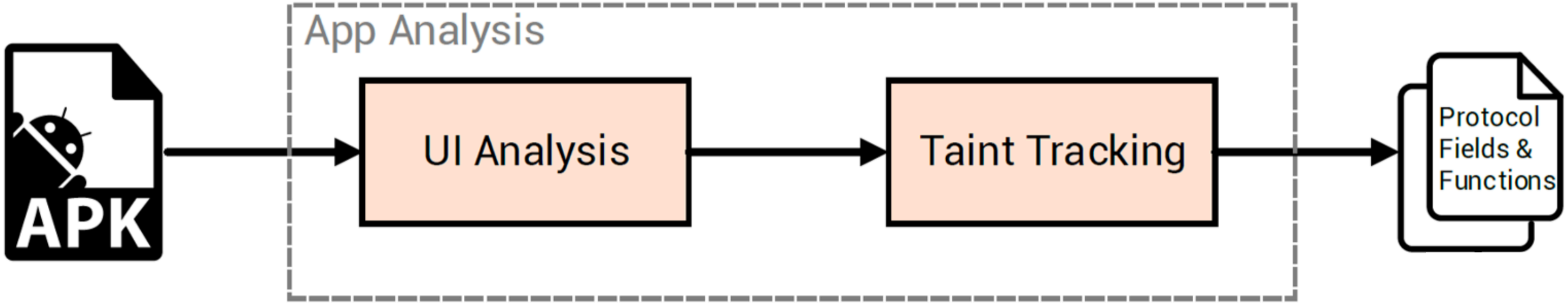
- Goal: Automatically generate protocol-aware messages to the IoT devices to discover memory corruptions

- Assumptions
  - IoT device under testing are configurable and controllable with mobile apps
  - Wi-Fi communication protocol
  - Android apps

IoTFuzzer: Architecture

- 2-phase architecture

- Phase 1:
    - App analysis

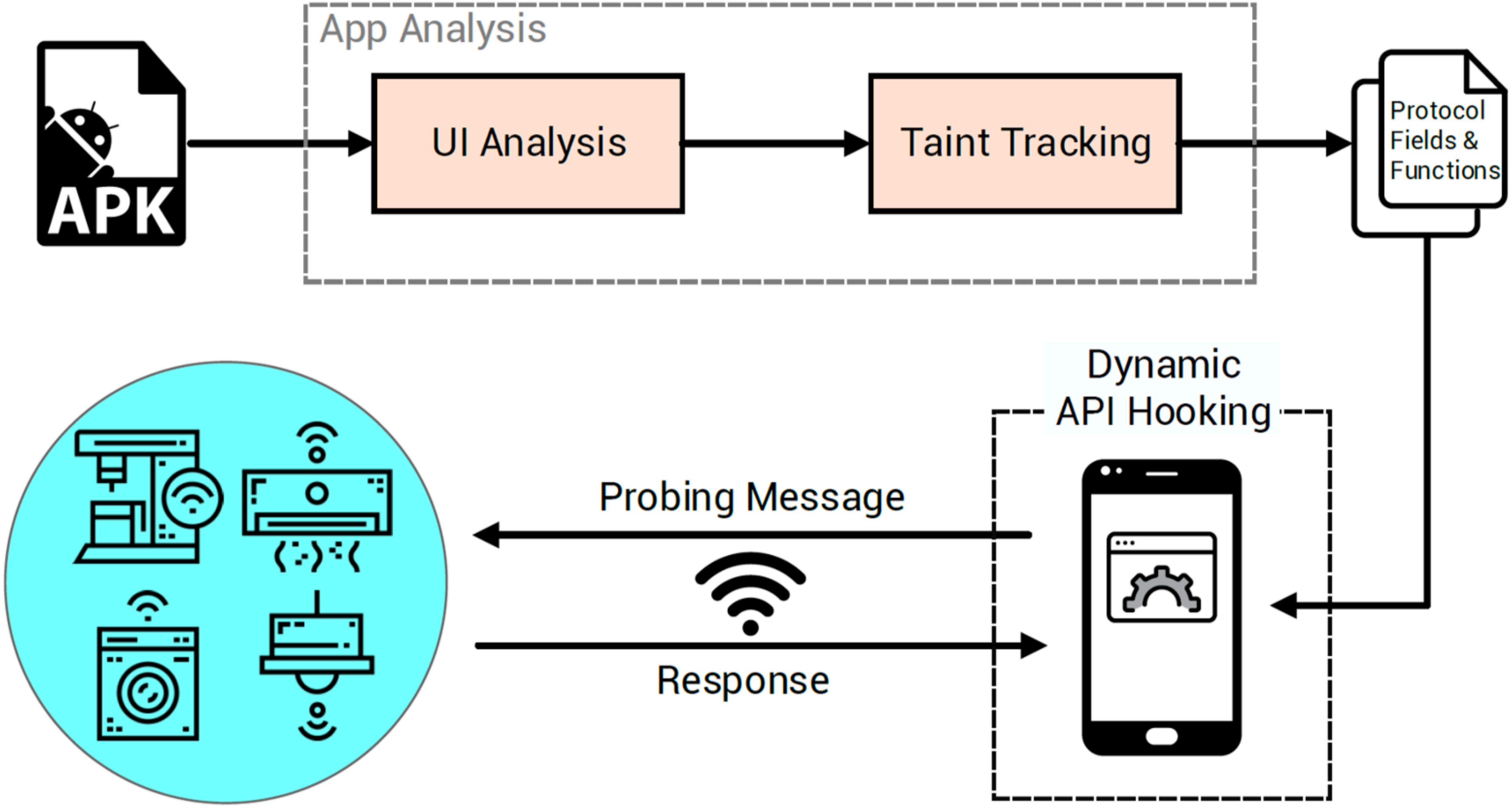


IoTFuzzer: Architecture

- 2-phase architecture

- Phase 1:
  - App analysis

- Phase 2:
  - Fuzzing



IoTFuzzer: Architecture – Phase 1

❑UI Analysis

- Call Path Construction
  - Identify networking UI elements by constructing call paths from networking APIs to UI event handlers
  - Networking APIs: URL.openConnection(), Socket.getOutputStream(), etc
  - Androguard [1]

- Activity Transition Graph Construction
  - To trigger networking API events
  - Monkeyrunner [2]

1. "Androguard: Reverse engineering, Malware and goodware analysis of Android applications," https://github.com/androguard/androguard
2. "monkeyrunner," https://developer.android.com/studio/test/monkeyrunner/index.html

IoTFuzzer: Architecture – Phase 1

- Taint Analysis
  - Identify protocol fields (variables) and functions
  - TaintDroid [W. Enck et al. TOCS'14]

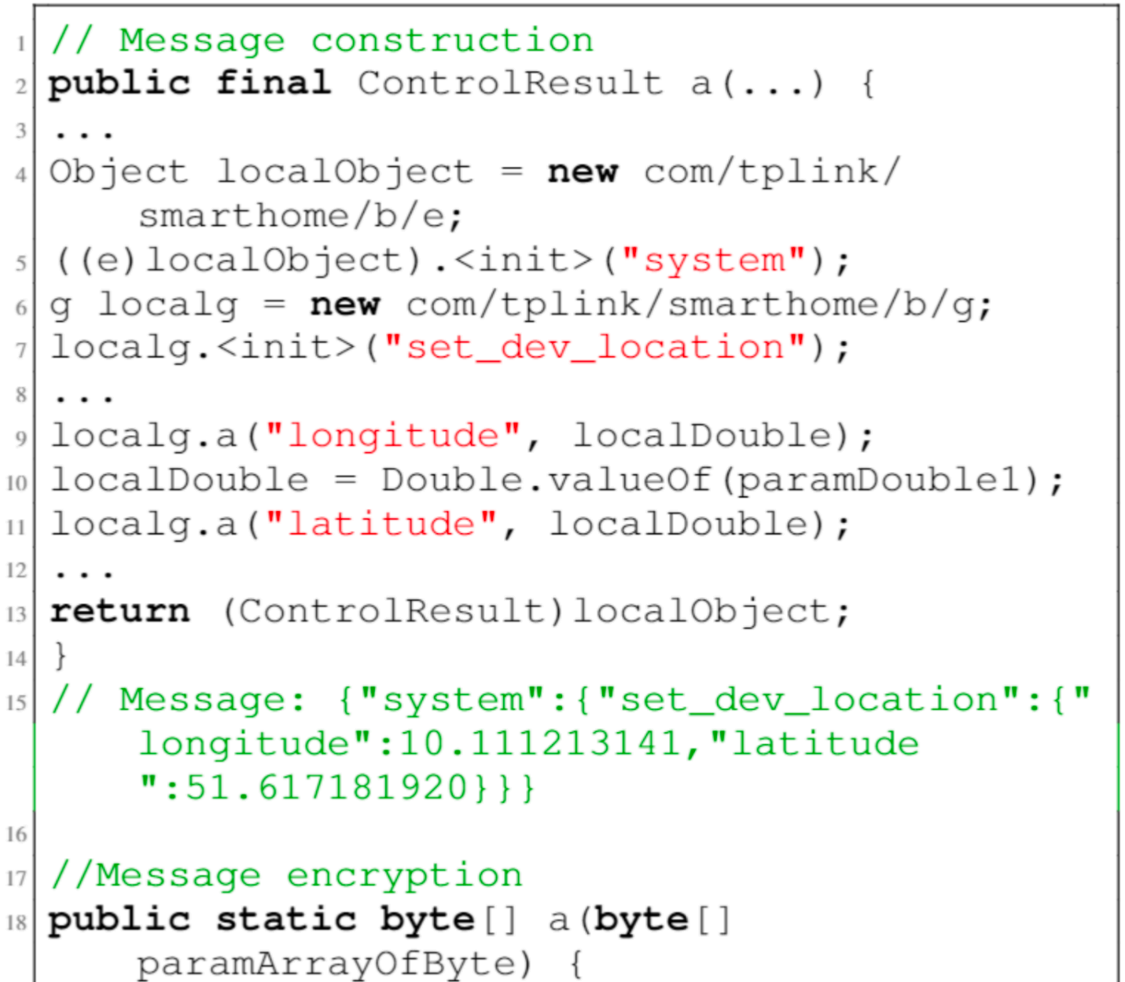
- Taint Sources: strings, system APIs, user inputs

- Taint Sinks: data used at networking APIs and encryption functions

- Cryptographic Function Identification
  - Lots of related work
  - IoTFuzzer employs a lightweight technique
  - Cryptographic functions contain arithmetic operations and called during the message delivery execution

IoTFuzzer: Architecture – Phase 1

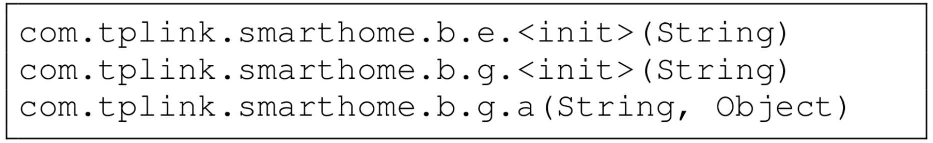
Code example



```
// Message construction
public final ControlResult a(...) {
...
Object localObject = new com/tplink/
    smarthome/b/e;
((e)localObject).<init>("system");
g localg = new com/tplink/smarthome/b/g;
localg.<init>("set_dev_location");
...
localg.a("longitude", localDouble);
localDouble = Double.valueOf(paramDouble1);
localg.a("latitude", localDouble);
...
return (ControlResult)localObject;
}
// Message: {"system":{"set_dev_location":{"
    longitude":10.111213141,"latitude
    ":51.617181920}}}

//Message encryption
public static byte[] a(byte[]
    paramArrayOfByte) {
```

Taint Tracking Output



```
com.tplink.smarthome.b.e.<init>(String)
com.tplink.smarthome.b.g.<init>(String)
com.tplink.smarthome.b.g.a(String, Object)
```

IoTFuzzer: Architecture – Phase 2

❑Runtime Mutation

- Function Hooking
  - Dynamically hooks the recorded functions and mutate the protocol fields at runtime to generate probe messages
  - Xposed [3]

- Fuzzing Scheduling: to fuzz only a subset of all protocol fields

- Fuzzing Policy:
  - Change the length of the strings to check overflow and out-of-bound access
  - Change integer, double, or float (large values) to check overflow and out-of-bound access
  - Change object types and provide empty values to check misinterpretation and null-pointer exepction

1. Rovo89, “Xposed Module Repository,” http://repo.xposed.info/

IoTFuzzer: Architecture – Phase 2

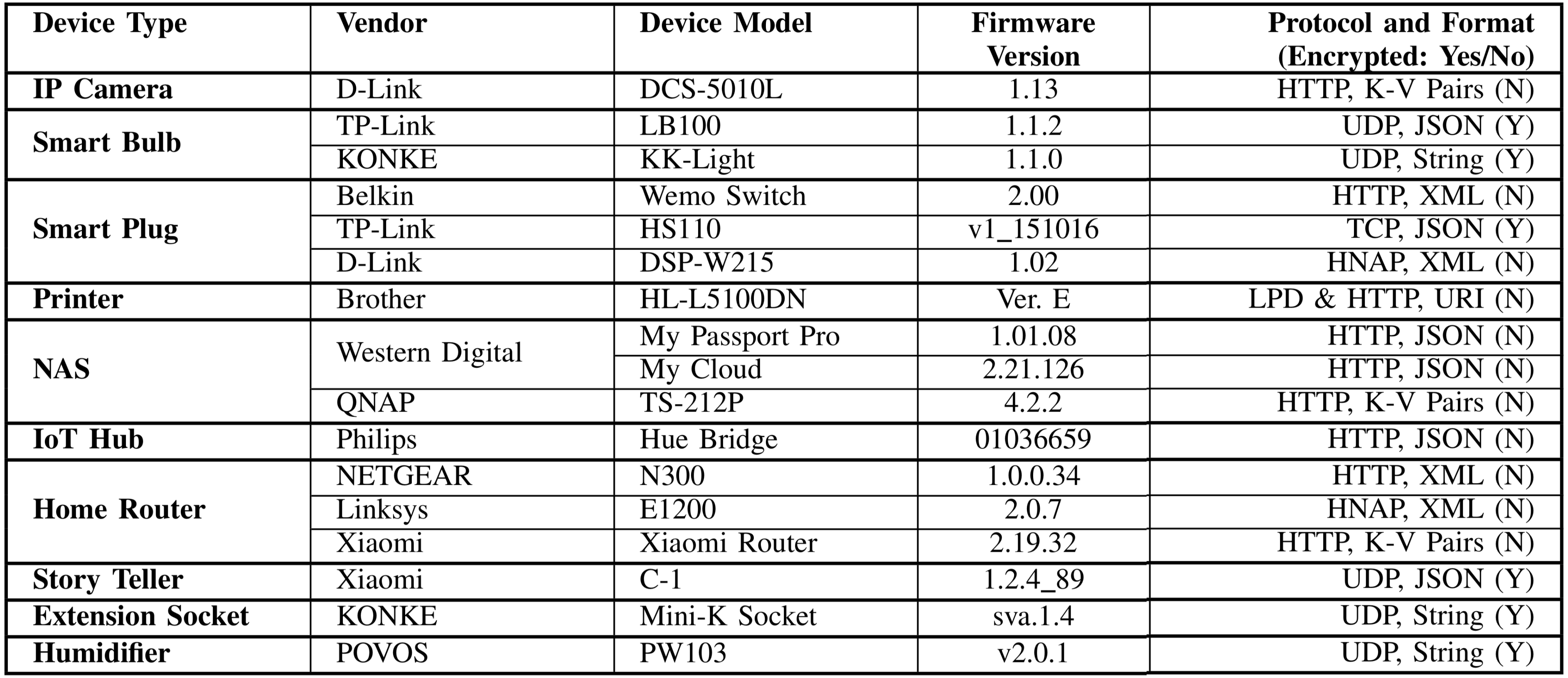
❑Response monitoring

- Response Types
  - Expected response
  - Unexpected response
  - No response
  - Disconnection

- Crash Detection
  - TCP-based connection: disconnection
  - UDP-based connection: insert a heartbeat message after every 10 probe messages

Implementation

- Implemented on 17 off-the-shelf IoT devices (apps are available on Google Play)

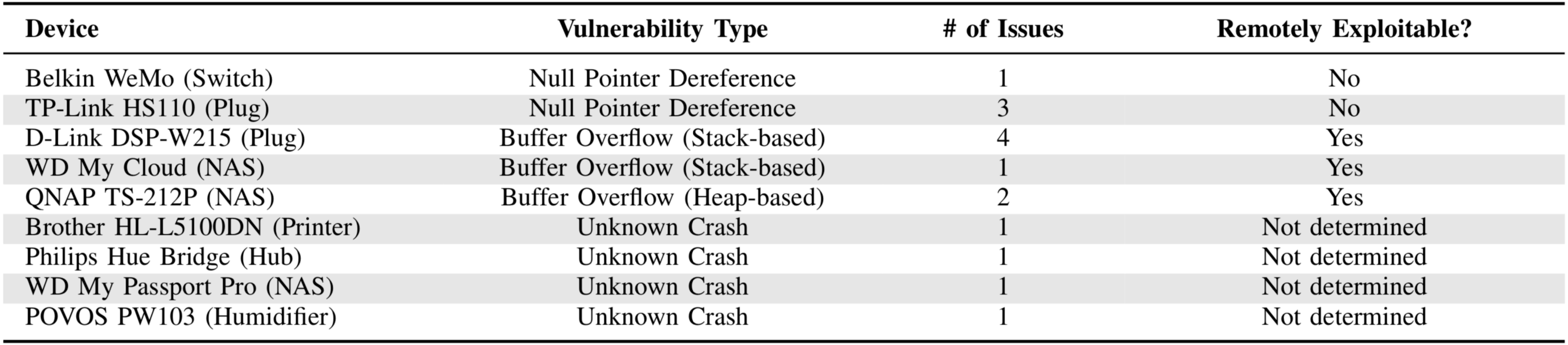


Device Type	**Vendor**	**Device Model**	**Firmware Version**	**Protocol and Format (Encrypted: Yes/No)**
IP Camera	D-Link	DCS-5010L	1.13	HTTP, K-V Pairs (N)
Smart Bulb	TP-Link	LB100	1.1.2	UDP, JSON (Y)
	KONKE	KK-Light	1.1.0	UDP, String (Y)
Smart Plug	Belkin	Wemo Switch	2.00	HTTP, XML (N)
	TP-Link	HS110	v1_151016	TCP, JSON (Y)
	D-Link	DSP-W215	1.02	HNAP, XML (N)
Printer	Brother	HL-L5100DN	Ver. E	LPD & HTTP, URI (N)
NAS	Western Digital	My Passport Pro	1.01.08	HTTP, JSON (N)
		My Cloud	2.21.126	HTTP, JSON (N)
	QNAP	TS-212P	4.2.2	HTTP, K-V Pairs (N)
IoT Hub	Philips	Hue Bridge	01036659	HTTP, JSON (N)
Home Router	NETGEAR	N300	1.0.0.34	HTTP, XML (N)
	Linksys	E1200	2.0.7	HNAP, XML (N)
	Xiaomi	Xiaomi Router	2.19.32	HTTP, K-V Pairs (N)
Story Teller	Xiaomi	C-1	1.2.4_89	UDP, JSON (Y)
Extension Socket	KONKE	Mini-K Socket	sva.1.4	UDP, String (Y)
Humidifier	POVOS	PW103	v2.0.1	UDP, String (Y)

Evaluation

- Testing Environment
  - UI Analysis: Ubuntu 14-04 Intel Core i7 quad-core 2.81 GHz CPU 8GB RAM
  - Taint Tracking: Google's Nexus 4
  - Network: Fully controlled local Wi-Fi

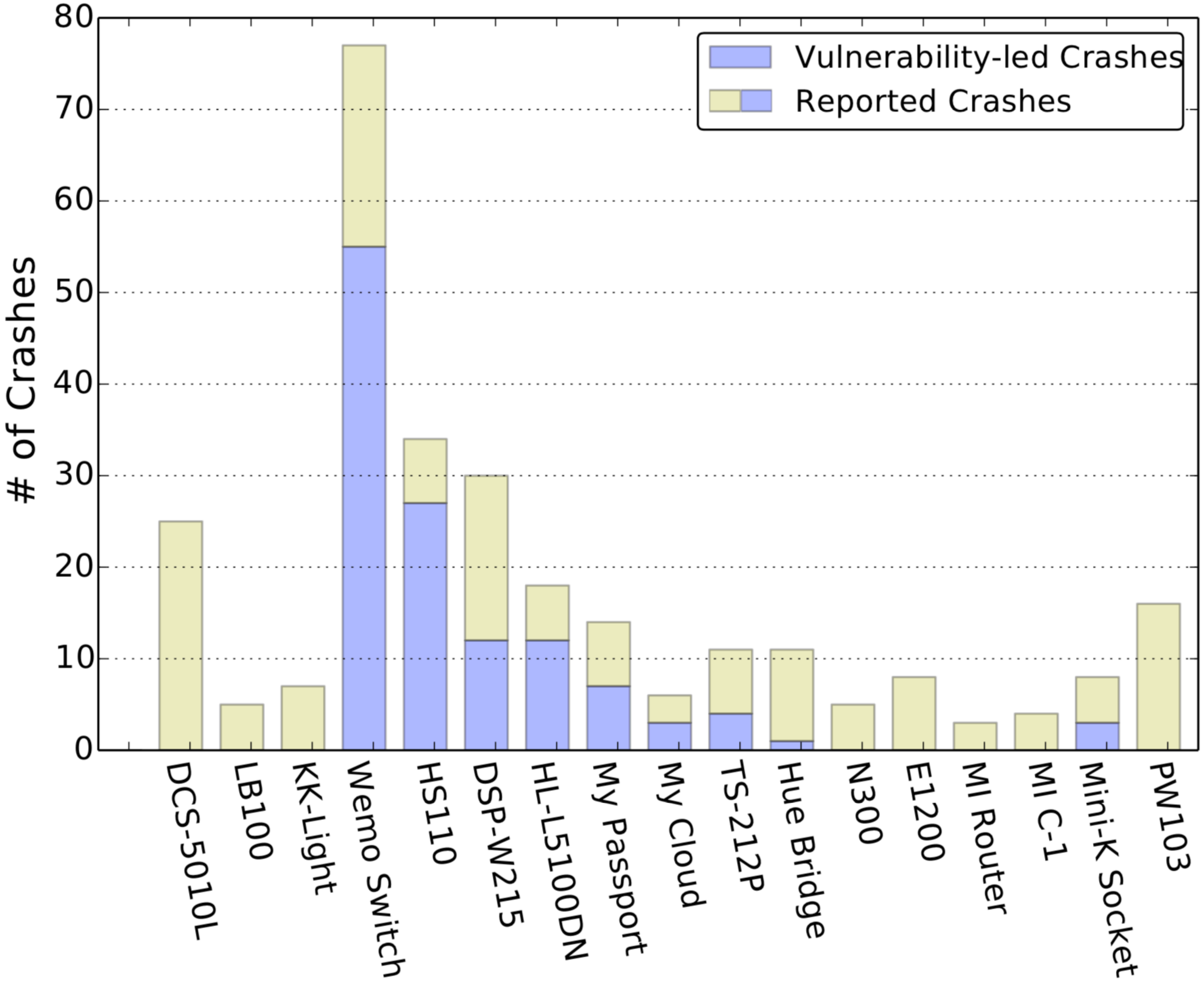
- 15 memory corruptions were found including 8 previously unknown



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

- Fuzzing accuracy



Conclusion

- IoTFuzzer: Limitations
  - Only support Wi-Fi connections
  - Can only fuzz app-related code in IoT devices
  - Only detects memory related corruptions that lead to crashes

Questions?