iFIZZ: Deep-State and Efficient Fault-Scenario Generation to Test IoT Firmware

Peiyu Liu Shouling Ji Qinming Dai Kangjie Lu Lirong Fu Wenzhi Chen
Peng Cheng Wenhai Wang Raheem Beyah

2021

IoT Devices in Real-world Applications

IoT devices are being widely adopted in real-world industries and living environments

Physical Security

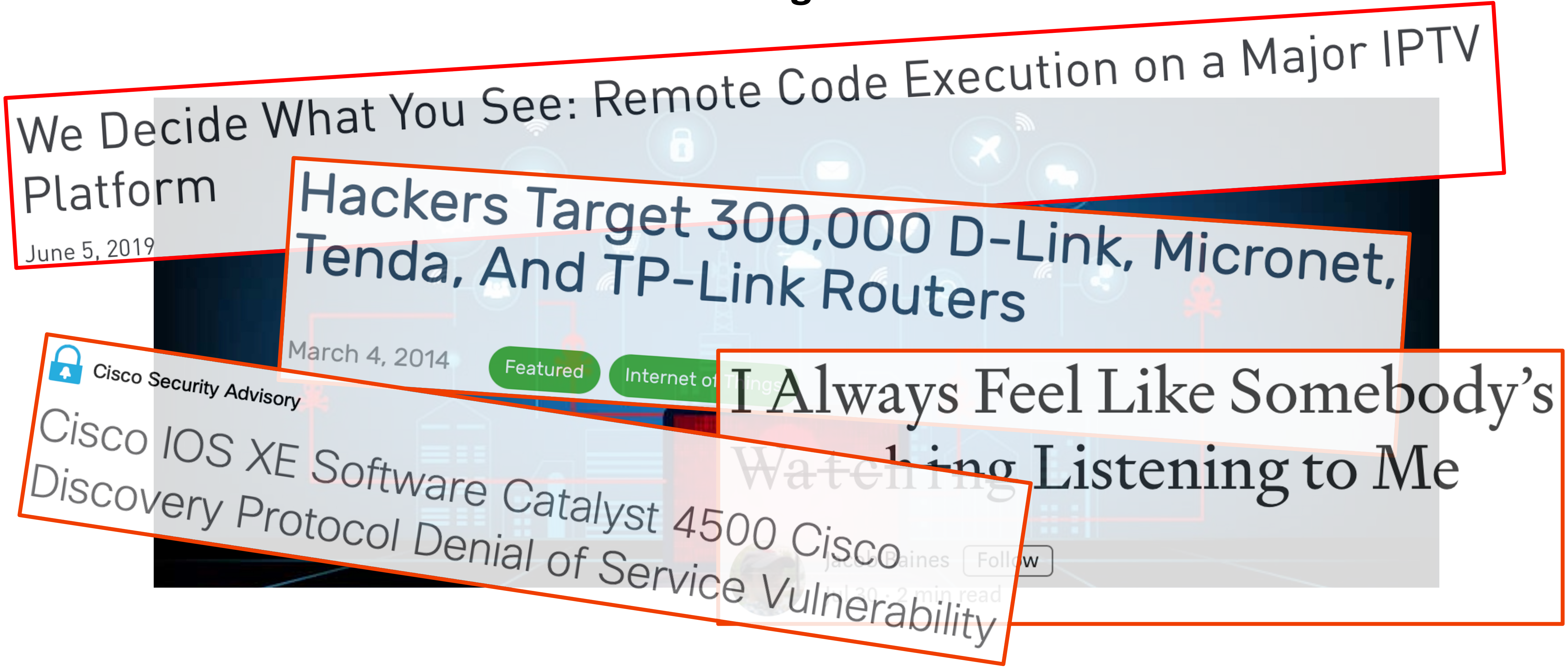
Smart Home

Healthcare

Smart City

IoT Devices are Vulnerable

IoT devices have become attractive targets for attackers

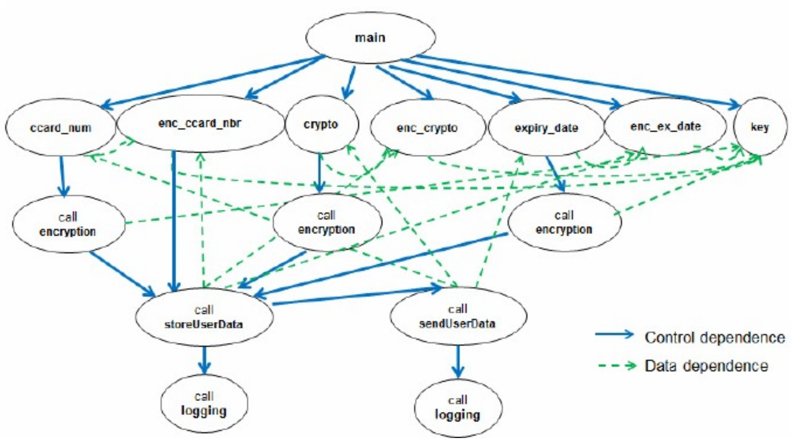


Discovering Vulnerabilities in IoT Firmware

Various detection systems appear to discover vulnerabilities in IoT firmware

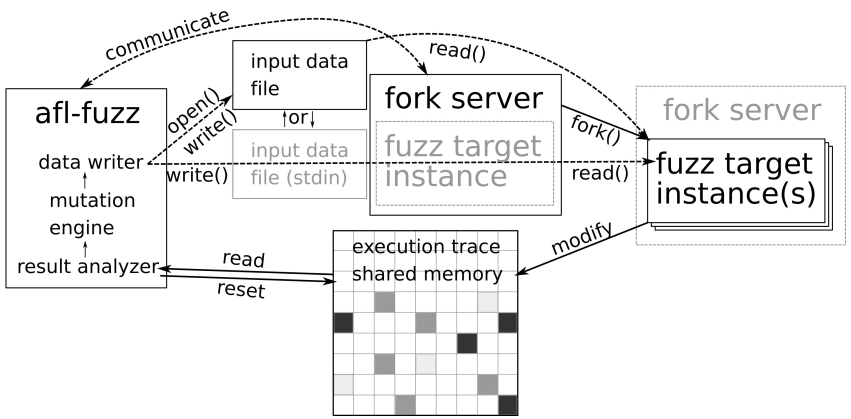
➢ Static Analysis

- Taint analysis
- Symbolic execution
- Graph matching
- Inaccurate



➢ Dynamic Analysis

- Proof of concept
- Fuzzing
- Cannot effectively test error-handling code



Motivation

Error-handling Code in IoT Firmware

Runtime errors are particularly common in IoT firmware

- Complex hardware dependence (Hardware failures)
- Limited hardware and system resources (Memory-allocation failures)

```c
FILE *open_memstream( ... ) {
    register __oms_cookie *cookie;
    ...
    if ((cookie->buf = malloc(...)) == NULL) {
        goto EXIT_cookie;
    }
 EXIT_cookie:
    free(cookie);
    return NULL;
```

An example of error-handling code in IoT firmware.

Error-handling code is intended in erroneous situations where security or reliability issues may potentially occur.

Error-handling Code in IoT Firmware is Buggy

Error-handling code in IoT firmware tends to be error-prone

- Developers may make mistakes when handling complex nested errors.
- More than 28% of IoT patches fix bugs in the error-handling code.
- The patched bug is just the tip of the iceberg.

Program	OpenWRT		DD-WRT	
	Patches	Error-handling	Patches	Error-handling
busybox	43	8(18.6%)	148	38(25.7%)
dnsmasq	66	27(40.9%)	81	29(35.8%)
dropbear	27	5(18.5%)	68	23(33.8%)
iptables	35	8(22.9%)	52	16(30.8%)
Total	171	48(28.1%)	349	106(30.4%)

Study result of IoT firmware patches.

Testing Error-handling Code in IoT Firmware is Important

- If error-handling code is incorrect, the intended protection is void.
- Bugs in error-handling code can cause serious security problems, such as DoS and information leakage.
- An attacker could intentionally trigger the errors to exploit the bugs in error-handling code.
- There are still no existing effective approaches for analyzing IoT error-handling code yet.

It is necessary and critical to comprehensively and effectively test the error-handling code of IoT firmware to detect hidden bugs.

Challenges

Testing Error-handling Code in IoT Firmware is Challenging

Three unique challenges in testing error-handling code in IoT

- C1. Identifying potential runtime errors in IoT firmware
  - Complex hardware dependence and execution environments.
  - The source code of IoT firmware is often not available.
- C2. Effectively covering error-handling code in IoT firmware
  - If an early error stops the execution, the fuzzing will not be able to reach and test deep error paths.

Design of iFIZZ

iFIZZ

iFIZZ: a framework for efficiently testing deep error-handling code in IoT firmware

- Automated identification of potential runtime errors
  - Automated binary-based runtime error identification
- Testing of deep error paths
  - State-aware and bounded fault-scenario generation

Automated Binary-based Runtime Error Identification

- Two characteristics of runtime errors in IoT firmware
  - Error code as the return value
  - Input-independent error conditions
- Identifying self-defined error codes
- Analyzing input-independent error conditions

```
FILE *open_memstream( ... ) {
    register __oms_cookie *cookie;
    ...
    if ((cookie->buf = malloc(...)) == NULL) {
        goto EXIT_cookie;
    }
 EXIT_cookie:
    free(cookie);
    return NULL;
```

An example of error-function.

State-aware and Bounded Fault-scenario Generation

State-aware error producing

- **Observation.** If a runtime error at a specific error stack leads to a crash in a fault-scenario, it is highly possible that the error in the same error stack will trigger the same (redundant) crash in another fault-scenario.

- Reduce redundant fault-scenarios by leveraging the state (defined as runtime context of an error site, i.e., its call stack and its prior error sequences) of error sites.

State-aware and Bounded Fault-scenario Generation

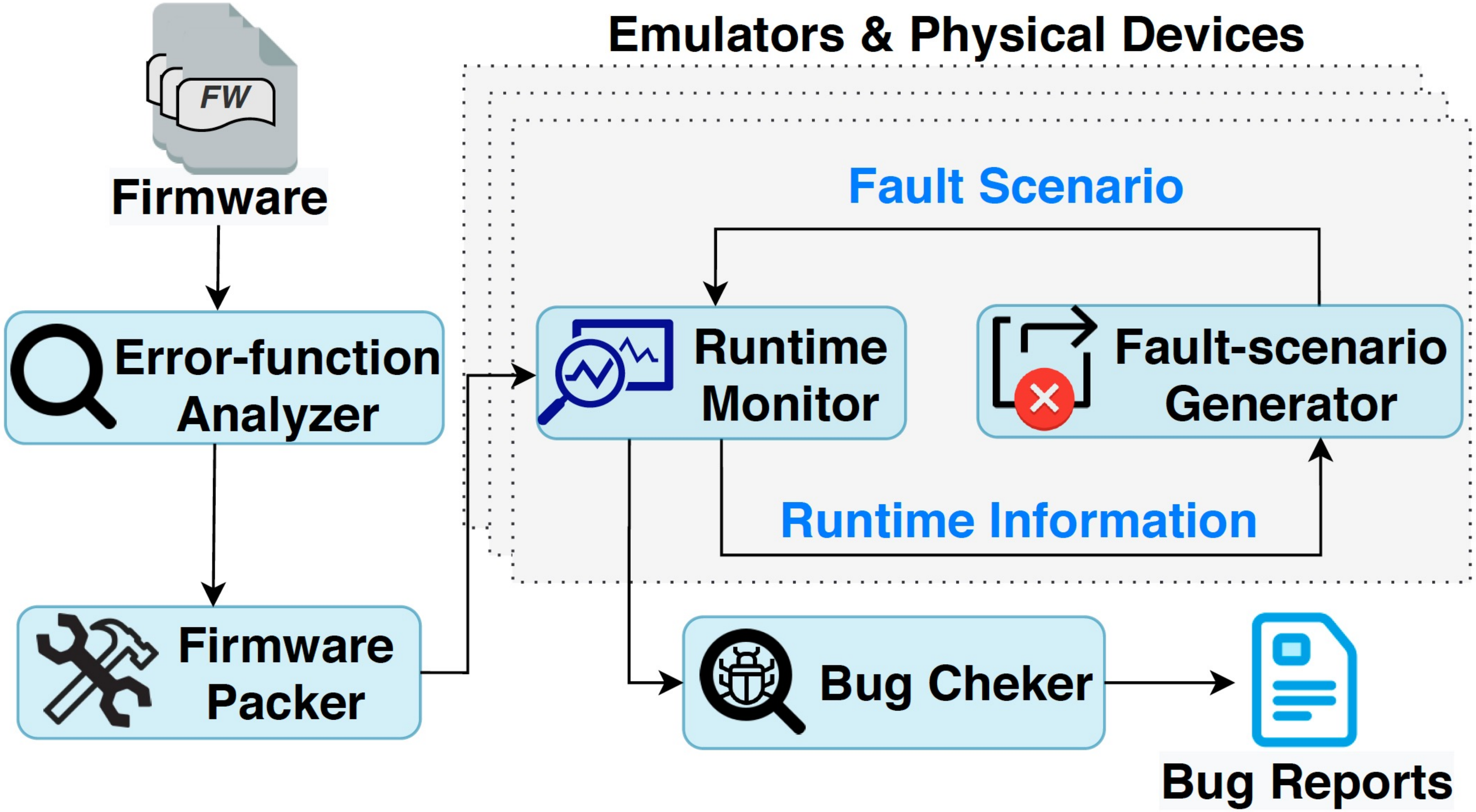
Bounded faults

- **Observations.** (1) Most crashes are caused by only a small number of errors, generating fault-scenarios with a large number of errors is often unnecessary. (2) Most crashes are caused by neighboring errors.

- The maximum number of errors (ME).

- The maximum distance between the first and the last error (MBE).

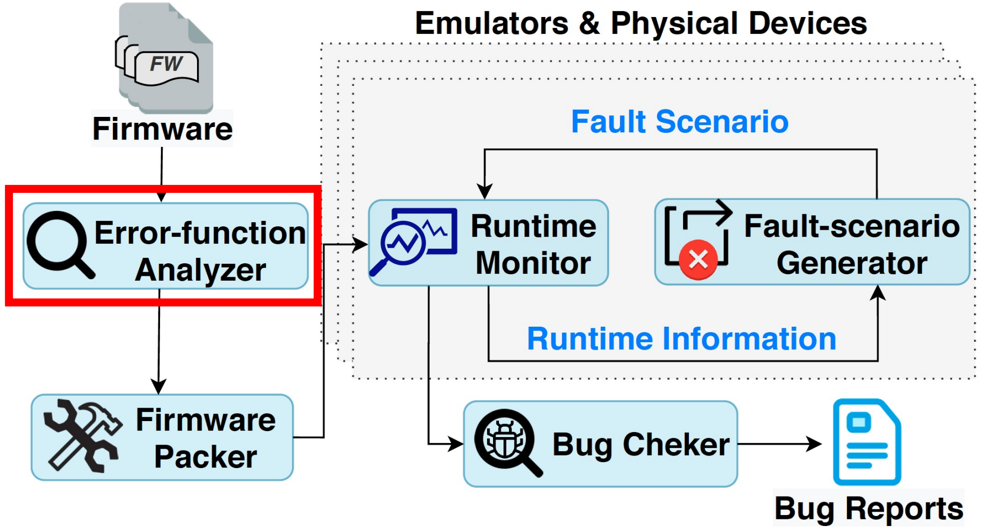
Overall Architecture of iFIZZ



Overall Architecture of iFIZZ

Error-function analyzer

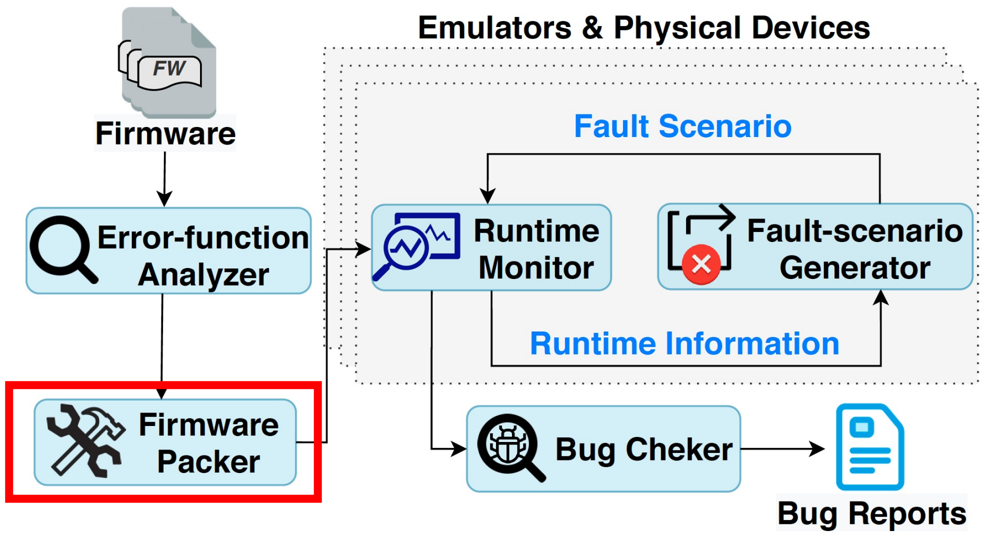
- Unpack firmware images to get the IoT programs.
- Analyze the assemble code of the tested program to identify error-functions.
- leverage automated binary-based runtime error identification method.



Overall Architecture of iFIZZ

Firmware packer

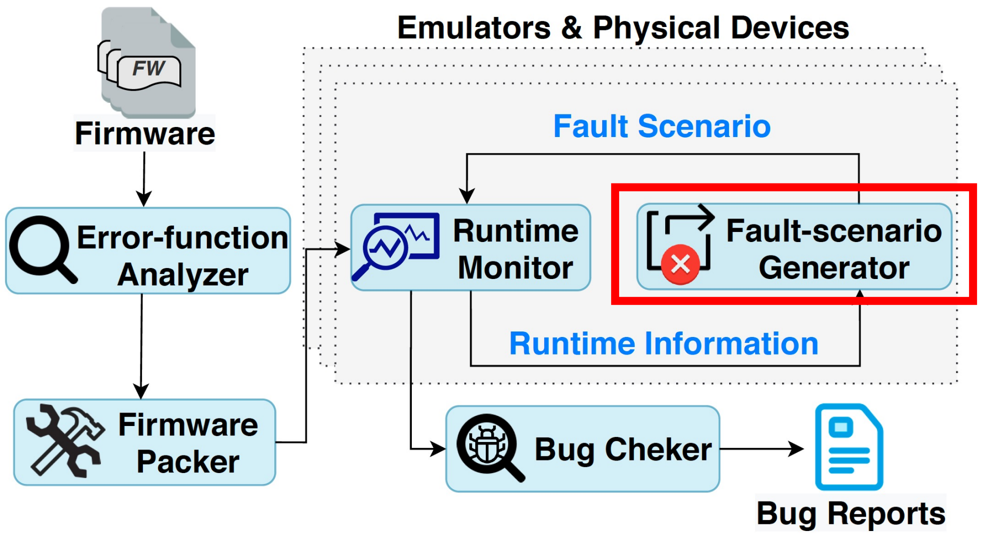
- Repack the tested programs and other necessary tools, e.g., telnet.
- Enable the debug interfaces of the tested firmware.
- Put the fault-scenario generator and the runtime monitor into the tested firmware.



Overall Architecture of iFIZZ

Fault-scenario generator

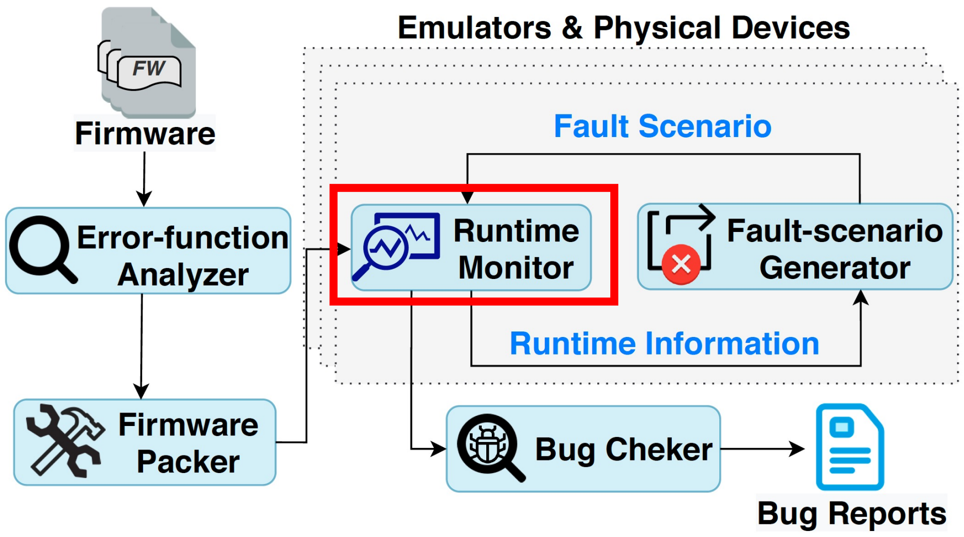
- Create test cases according to our state-aware and bounded fault-scenario generation method.
- A dynamically linked library.



Overall Architecture of iFIZZ

Runtime monitor

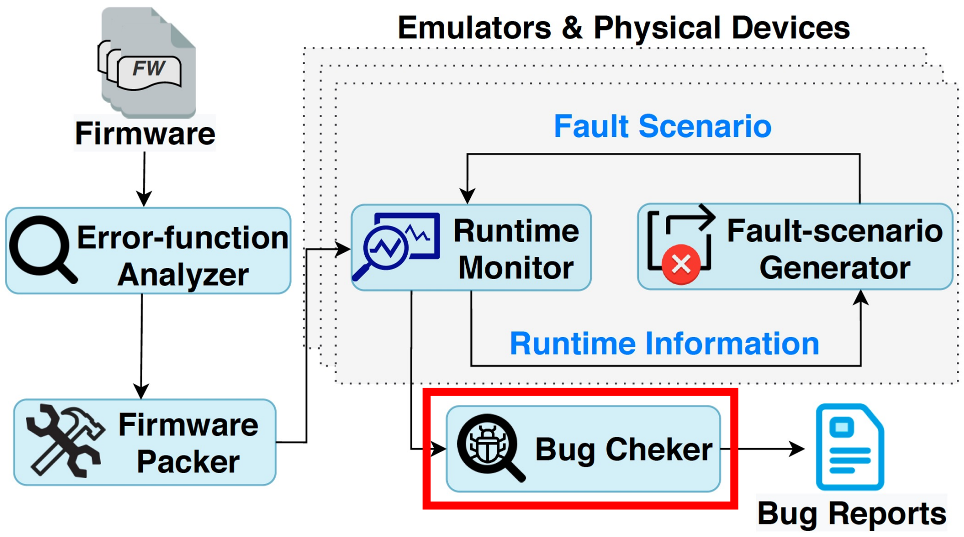
- Obtain the target IoT programs and their corresponding run-commands.
- Produce errors according to fault-scenario by hijacking error-functions.



Overall Architecture of iFIZZ

Bug checker

- Perform an automated analysis of the collected runtime information of detected crashes to generate crash reports.



Evaluation

Experimental Setup

Tested firmware

- 10 IoT firmware produced by 7 vendors are used for evaluation.
- 7 firmware images are tested on emulators, and 3 are tested in physical devices.

Model	Vendor	Version	Device	Arch
DIR-850L	DLink	1.00B05	Router (E)	Mipseb
DGS-1210-48	DLink	2.03.001	Switch (E)	Armel
FW_TV-IP121WN	Trendnet	V2_1.2.1.17	Camera (E)	Mipseb
K2	Phicomm	v163	Router	Mipsel
K2	OpenWRT	17.01.0	Router	Mipsel
TYCAM110	Tuya	V2	Camera	Armel
WAP200	Cisco	2.0.4.0	AP (E)	Mipseb
WAP4410N	Cisco	2.0.7.8	AP (E)	Mipseb
WNAP320	Netgear	v3.0.5.0	AP (E)	Mipseb
WG103	Netgear	V2.2.5	AP (E)	Mipseb

Basic information of the tested firmware.

Error-Function Extraction

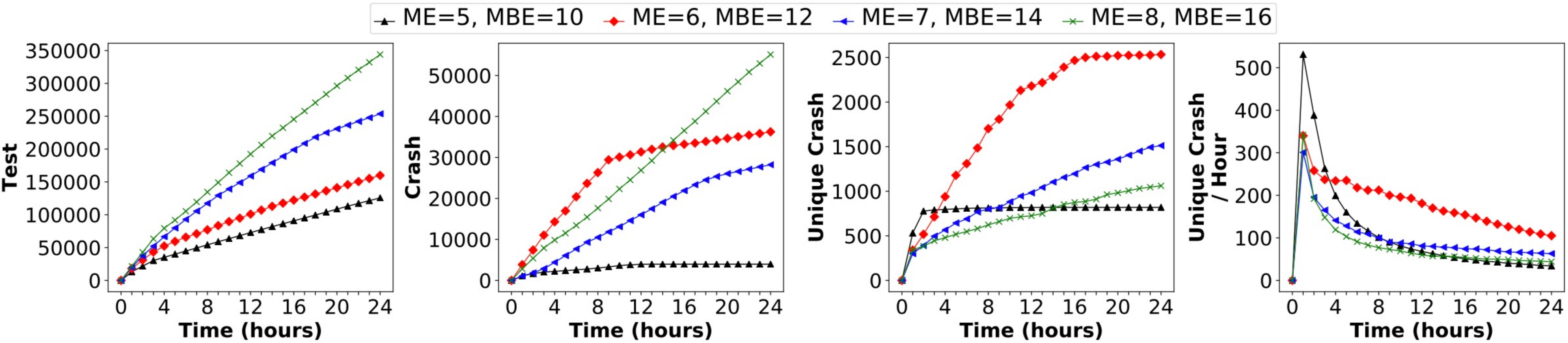
- iFIZZ identifies 140 error-functions out of 3,349 functions.
- 11 false positives in the identified error-functions.

Library	Function	Error-function
libuClibc-0.9.29.so	937	82
libuClibc-0.9.30.so	1090	11
libuClibc-0.9.30.3.so	1138	44
libcrypt-0.9.29.so	3	1
libcrypt-0.9.30.3.so	4	1
libxtables.so.2.0.0	40	2
Total	**3349**	**140**

Result of error-function extraction.

Variation of Results with Respect to Different ME and MBE

- In a certain testing time (24 hours in our test), a set of moderate bounds (ME = 6 and MBE = 12) can improve the efficiency of discovering unique crashes.

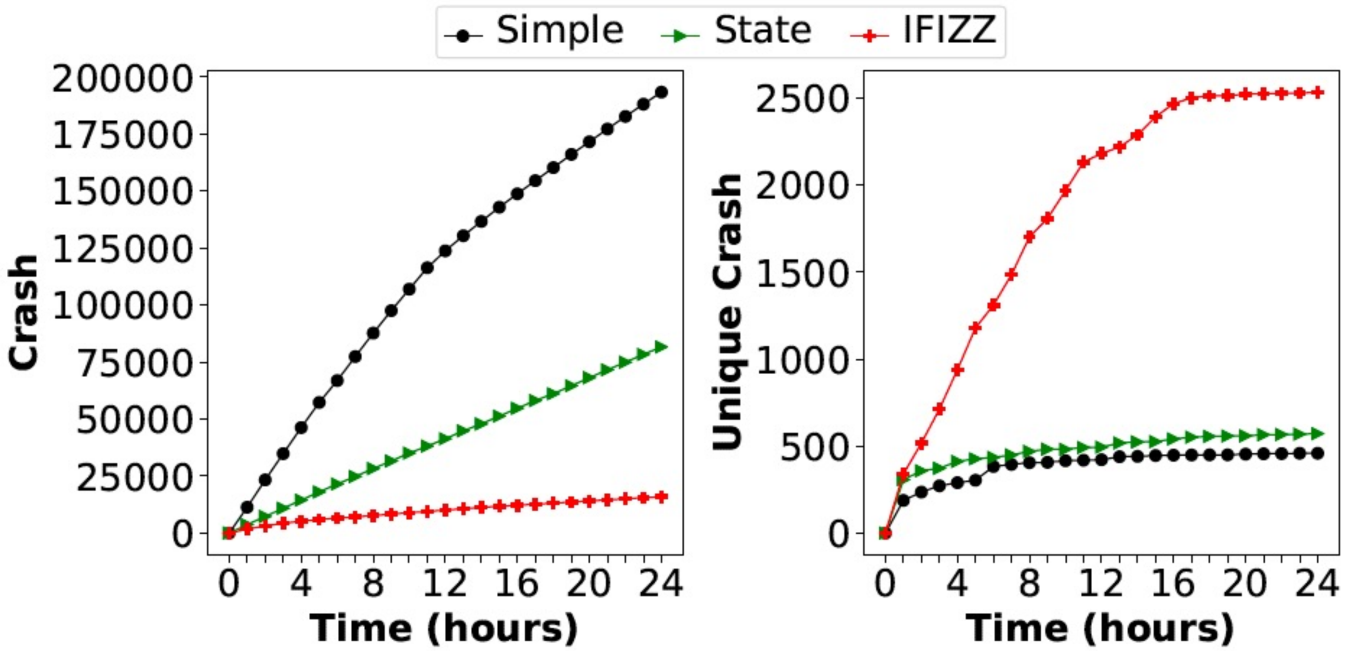


Variation of results with respect to different ME and MBE.

Fault-scenario Generation

Unique crashes

- ➢ iFIZZ can find the most unique crashes.

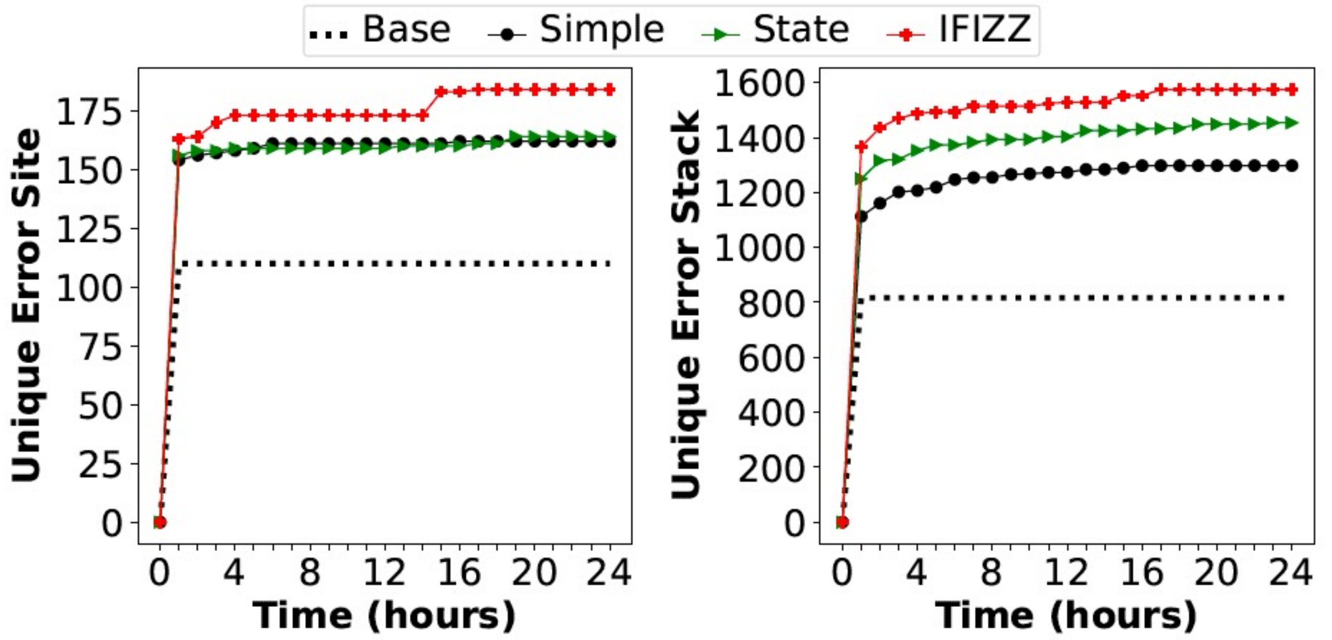


Crashes discovered by different fault-scenario generation approaches.

Fault-scenario Generation

Error-path coverage

- iFIZZ can cover the most error sites and error stacks.

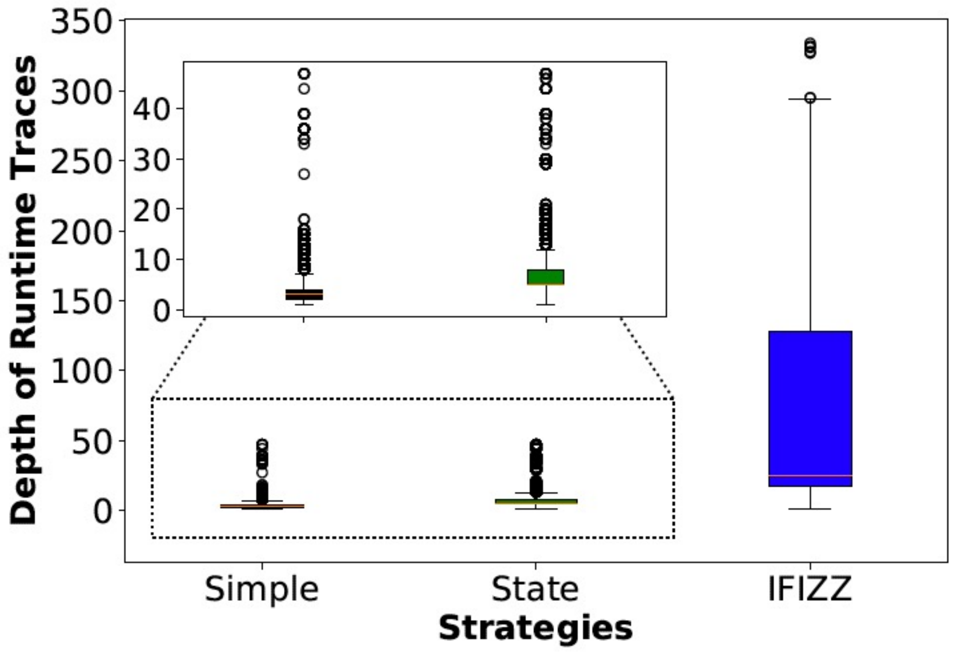


Code coverage of different fault-scenario generation approache.

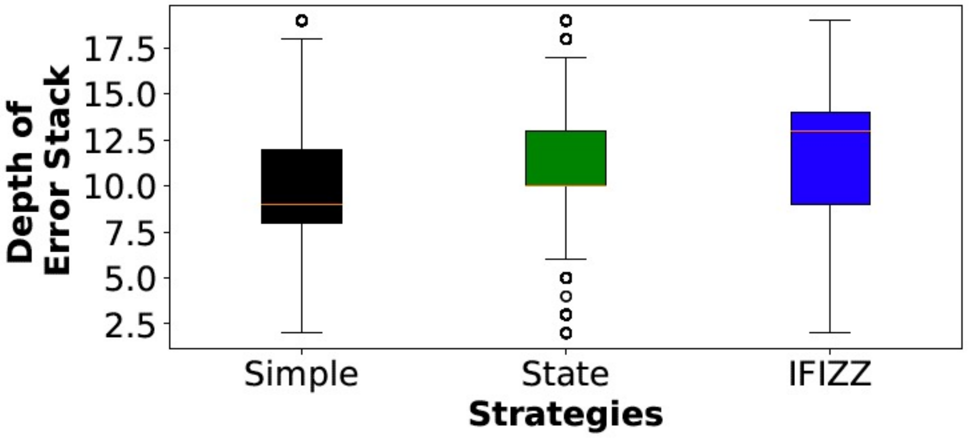
Fault-scenario Generation

Error-path depth

- iFIZZ can trigger deeper error paths than other tools.



Depth of runtime traces covered by different fault-scenario generation approaches.



Depth of error stacks covered by different fault-scenario generation approaches.

Results of Error-handling Testing

Detected bugs

➢ iFIZZ finds 46 program bugs and 63 library bugs in the tested firmware images.

Firmware	Unique Crash	Confirmed Bug	BP	BL
DIR-8505	167	9	2	7
DGS-1210-48	6	4	2	2
FW_TV-IP121WN	21	2	0	2
K2	127	4	2	2
OpenWRT	45	5	3	2
TYCAM110	227	32	17	15
WAP200	190	11	2	9
WAP4410N	3079	7	0	7
WNAP320	2112	23	13	10
WG103	2270	12	5	7
Total	**8244**	**109**	**46**	**63**

Detected bugs in IoT firmware.

Comparison with Existing Tools

iFIZZ vs. FirmAFL

- iFIZZ can find significantly more unique crashes than FirmAFL.
- iFIZZ can report unique crashes more efficiently.

Program/Lib	iFIZZ Crash	iFIZZ Unique Crash	FirmAFL Crash	FirmAFL Unique Crash
bzcat	28	**12**	5.07M	1
cmp	53	**13**	0	0
wc	56	**21**	182	19
uniq	89	**23**	0	0
Total	226	**69**	>5M	20

Results of iFIZZ and FirmAFL.

Case Study

```
void get_cmdln_options(int argc, char *argv[]) {
      str=(char*)malloc(strlen(...)+14);
      snprintf(str, strlen(...)+14,"%s/...",pwd_entry->
       pw_dir);
```

Arbitrary null write in bwm-ng.

```
FILE *open_memstream(...) {
      register __oms_cookie *cookie;
      if ((cookie = malloc(...))) != NULL) {
            if ((cookie->buf = malloc(...)) == NULL) {
                  goto EXIT_cookie;
            }
            ...
      }
      free(cookie->buf);
  EXIT_cookie:
      free(cookie);
      return NULL;
}
```

Null pointer dereference in uClibc.

Discussion

Discussion

False positives and false negatives

- Error-function identification
- Bug detection

Exploitability of error-handling bugs

Manual analysis

Related Work

Related Work

Analysis of error-handling code

- Jiang *et al.*, USENIX Security'20
- Bai *et al.*, USENIX ATC'16
- Jana *et al.*, USENIX Security'16
- Kang *et al.*, ASE'16
- Cong *et al.*, ASE'16
- Lawall *et al.*, ISSTA'15
- Zhang *et al.*, ICSE'13
- Saha *et al.*, ICSE'09
- …

Vulnerable IoT device discovery and analysis

- Zheng *et al.*, USENIX Security'19
- Muench *et al.*, NDSS'18
- Chen *et al.*, NDSS'18
- Xu *et al.*, CCS'17
- Feng *et al.*, CCS'16
- Chen *et al.*, NDSS'16
- Shoshitaishvili *et al.*, NDSS'15
- Costin *et al.*, USENIX Security'14
- …

Conclusion

Conclusion

- We presented a novel framework named iFIZZ to effectively test deep error-handling code of IoT firmware.
- We propose multiple new techniques in iFIZZ. (1) Automated binary-based error-function identification. (2) State-aware and bounded fault-scenario generation.
- We evaluate iFIZZ on 8 widely-used IoT firmware images from leading vendors. It in total finds 59 new bugs. iFIZZ covers 67.3% more error paths than normal execution, and the depth of error-handling code covered by iFIZZ is 15.3 times deeper than that covered by traditional fault injection on average.
- We will open-source iFIZZ for facilitating future IoT security research.

liupeiyu@zju.edu.cn