Efficient Greybox Fuzzing of Applications in Linux-Based IoT Devices via Enhanced User-Mode Emulation

Yaowen Zheng[†]
Continental-NTU Corporate Lab,
Nanyang Technological University
Singapore
yaowen.zheng@ntu.edu.sg

Yuekang Li[‡]
Continental-NTU Corporate Lab,
Nanyang Technological University
Singapore
yuekang.li@ntu.edu.sg

Cen Zhang
Continental-NTU Corporate Lab,
Nanyang Technological University
Singapore
CEN001@e.ntu.edu.sg

Hongsong Zhu
Beijing Key Laboratory of IoT
Information Security Technology, IIE,
CAS; School of Cyber Security, UCAS
Beijing, China
zhuhongsong@iie.ac.cn

Yang Liu
Nanyang Technological University
Singapore
yangliu@ntu.edu.sg

Limin Sun[‡]
Beijing Key Laboratory of IoT
Information Security Technology, IIE,
CAS; School of Cyber Security, UCAS
Beijing, China
sunlimin@iie.ac.cn

ISSTA’22

Background - Emulation-Based Fuzzing

full-system emulation ***vs*** user-mode emulation

- AFL + QEMU user-mode emulation - compatibility
- AFL + QEMU full-system emulation - efficiency
- AFL + QEMU hybrid emulation(Firm-AFL)

Background - Emulation-Based Fuzzing

AFL + QEMU user-mode emulation

- Wrong launch variables
- Missing dynamically generated files
- Inconsistent NVRAM configurations
- Inconsistent network behaviors
- Inconsistent process resource limits
- Lack of hardware

Goals

- **High compatibility**

  Applications should behave the same as in full-system emulation

- **High efficiency:**

  The speed of fuzzing should be as fast as possible

Solution

- Wrong launch variables
- Missing dynamically generated files
- Inconsistent NVRAM configurations
- Inconsistent network behaviors
- Inconsistent process resource limits

EQUAFL

AFL based framework through Enhanced QEMU User-mode emulation

observation and replay

- observation

  execute the PUT with full system emulation and observes the key behaviors of the system

- replay

  carry out the replay during the user-mode emulation

Overview

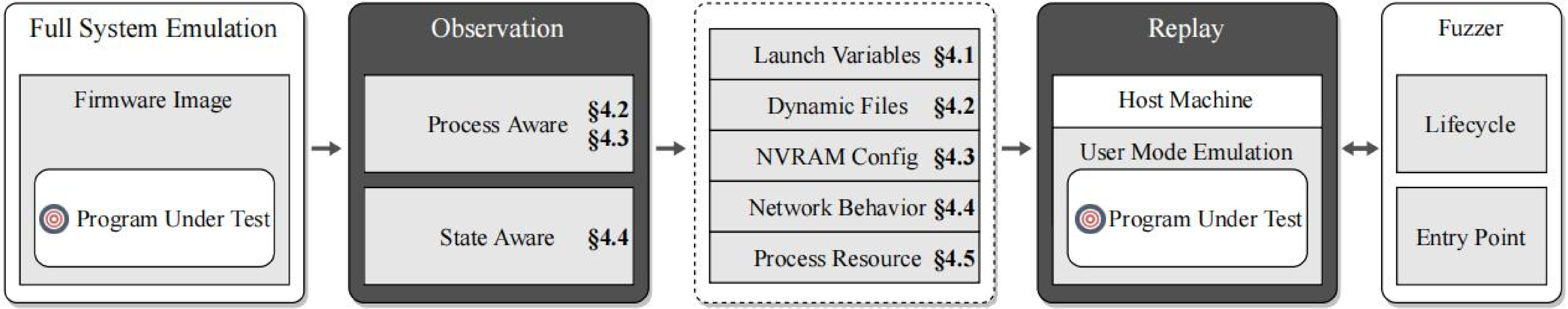


Figure 1: The workflow of EQUAFL

Contributions

- a novel technique
  - automatically set up the execution environment to emulate embedded programs fully in user-mode
  - guarantee both high compatibility and high efficiency.
- a new system
  - EQUAFL
  - a coverage-guided greybox fuzzing framework based on AFL and QEMU
- extensive evaluation
  - ten 0-day vulnerabilities including six CVEs
- source code

  https://github.com/zyw-200/EQUAFL

Approaches - Launch Variables Settlement

- **observation**
  - dump p_{name}, p_{vars} and p_{envs} of the newly starting program
  - instrument the kernel function ***do_execve*** during the full-system emulation

- **replay**
  - recognize p_{vars} and p_{envs} as the target $p*_{vars}$ and $p*_{envs}$,

  where p_{name} equals to $p*_{name}$

Approaches - Filesystem State Synchronization

In full-system emulation, many firmware images mount a temporary filesystem and constantly change the filesystem state during the initialization phase

observe file-related system call execution in the guest machine and re-execute it on the host machine

- **Accurate Process Identification**

identify the current executing process in the guest machine

  - process collection - PGD, PID, PPID (from *task_struct*)
  - process inference - acquire the PGD value of current executing process and match

- **Process-aware observation**
  - observe the relation of files with process awareness to get file descriptor on the host machine
  - M: $P \times FD_{guest} \longrightarrow fd_{host}$

Approaches - NVRAM Configuration

regular files are allocated to store the data of NVRAM configuration

emulation of NVRAM access : redirecting related APIs to the data access in such regular files

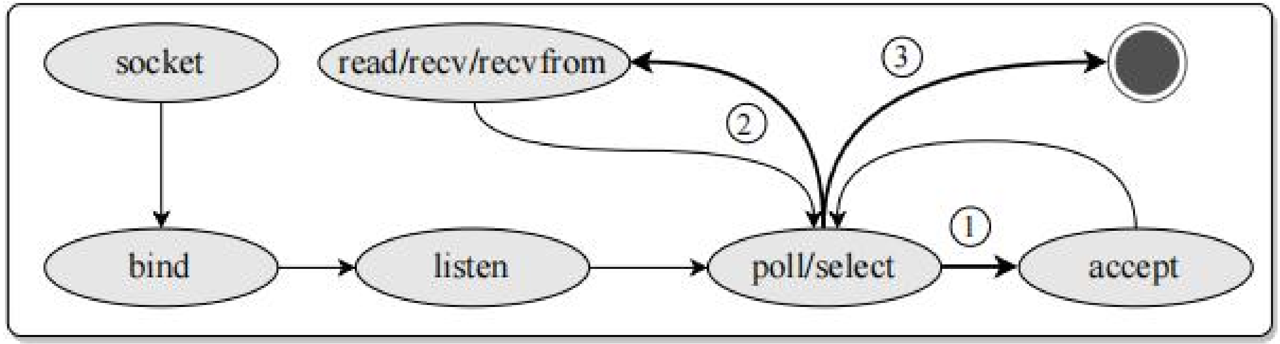
- **observation**
  - achieved in the filesystem state synchronization
- **replay**
  - redirecting the NVRAM access of the PUT to the NVRAM configuration files on the host machine

Approaches - Network Behavior

regular files are allocated to store the data of NVRAM configuration

emulation of NVRAM access : redirecting related APIs to the data access in such regular files

- **state-aware observation**
  - collect network related system call sequences of the PUT during full-system emulation
  - carry out the state machine to guide the emulation of network behaviors
- **replay**
  - follow the state machine to emulate the network-related system calls
  - instrument at the beginning of network-related system calls, and then feed the expected result



Approaches - Process Resource Limits

- **observation**
    - the Linux kernel provides **setrlimit** and **getrlimit** system calls to set or get values of process resource limits
    - monitor **getrlimit** during fullsystem emulation
- **replay**
    - provide the observed values directly to PUT

Implementation - observation

- full-system emulation: FIRMADYNE
- instrument full-system mode of QEMU at the end system calls execution
  - firmware executes to a system call
    - treats it as the starting point of system call execution
    - record current execution context
  - firmware executes to the kernel space
    - instruments at the end of each basic block to detect whether the firmware execution returns back to the user-space
  - firmware executes to the user-space and the execution context are equal to the previous records
    - treats it as the end of the system execution
    - collect the argument and the return value of each system call execution

Implementation - replay

- **deploy the related resource directly on the host machine**
  - launch variables, filesystem state synchronization, NVRAM configuration
- **instrument the user-mode mode of QEMU**
  - network behavior, process resource limits

Implementation - fuzzing

- **PUT lifecycle management**

	Fuzzing entry point	Fuzzing iteration end point
Before	main function	PUT executes to end
After	system call that receive the network input	PUT executes to *poll* or *select* system call

- **Entry point of fuzzing**
  - AFL: feed the test input as a **file** to the PUT
  - EQUAFL: feed the input to the **memory buffer** that store the network input

Evaluation

- **Benchmarks**
  - standard benchmarks: **nbench** and **lmbench**
  - real-world benchmarks: 70 embedded firmware images(D-Link, TRENDnet and NETGEAR)
- **Baselines**
  - AFL-User
  - AFL-Full
  - Firm-AFL

Evaluation - Compatibility

Real-world Dataset

Vendor	NUM	EQUAFL				AFL-User			
		SUCC	ERR	HAN	CRA	SUCC	ERR	HAN	CRA
D-Link	30	28	1	1	0	0	27	3	0
TRENDnet	11	9	0	2	0	0	7	4	0
NETGEAR	29	29	0	0	0	0	2	25	2
SUM	70	66	1	3	0	0	36	32	2

Emulation Accuracy

comparing the execution traces of the same seed with EQUAFL and with AFL-Full

identical: 44 / 66

high similarity: 16 / 66

totally different: 6 / 66 inter-process communication

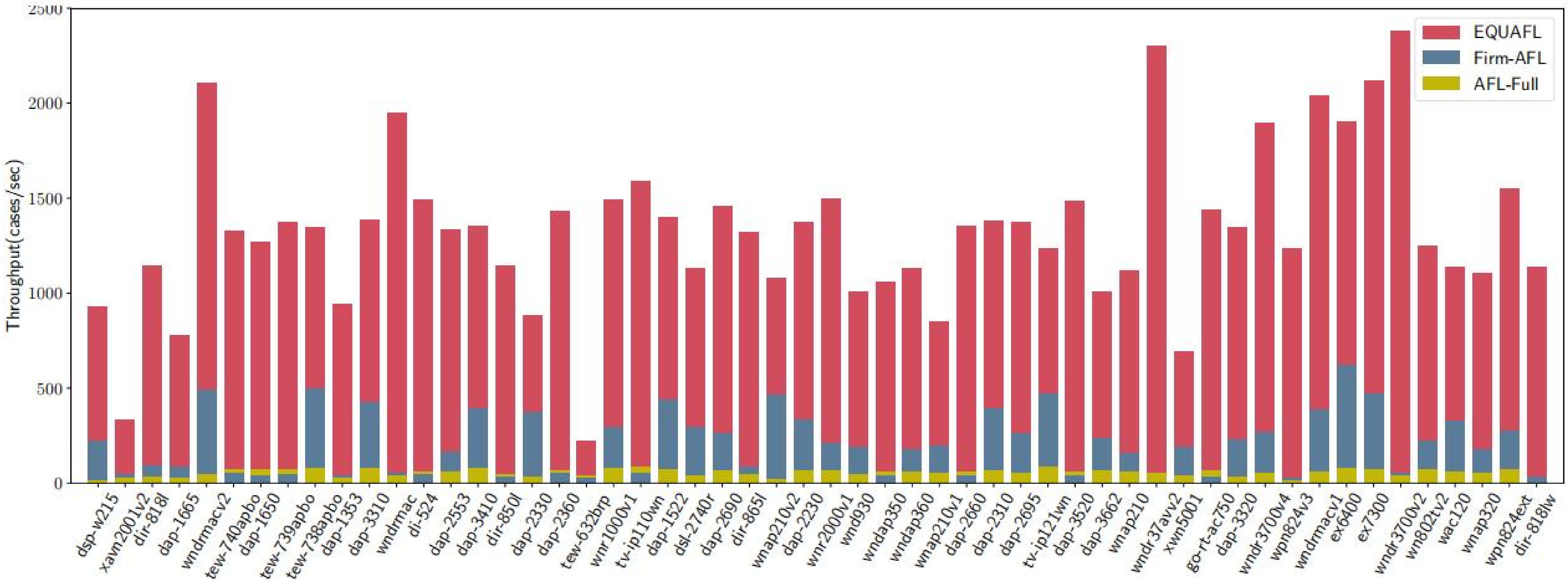
Standard Dataset

execute all the programs in nbench correctly

Evaluation - Efficiency

-相比AFL-Full：提高5.1-78倍（平均26倍）
-相比Firm-AFL：提高2.3-48倍（平均14倍)

Real-world Dataset



Evaluation - Efficiency

Standard Dataset

- **nbench**

  CPU and memory capabilities of a system
  RESULT: similar to the performance of pure user-mode emulation

- **lmbench**

  system call overhead
  RESULT: the overhead is for both file-related and network-related system calls

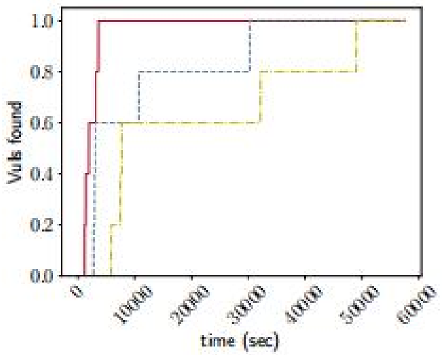
Evaluation - Vulnerability Discovery

Vendor	Vulnerable Product & Firmware Version	Vulnerable Application	Vulnerability NUM (Unknown)	CVE
NETGEAR	WN2000RPTv1 (1.0.1.20), WPN824EXT (1.1.1), WNR2000v1 (1.1.3.9), WNR1000v1 (1.0.1.5), WPN824v3 (1.0.8)	/bin/boa	1 (1)	N/A
NETGEAR	WNDRMACv1 (1.0.0.20), WNDRMACv2 (1.0.0.4), WNDR3700v2 (1.0.0.8), WNDR37AVv2 (1.0.0.10), WNCE4004(1.0.0.22)	/usr/sbin/uhttpd	2 (2)	N/A
D-Link	DIR-825 (2.01EU)	sbin/httpd	1 (1)	CVE-2021-29296
TRENDnet	TEW-632BRP (1.10.31), TEW-634GRU (1.01.06), TEW-652BRP (1.10.14), TEW-673GRU (1.00.36)			
D-Link	DSP-W215	/usr/bin/lighttpd	1 (1)	CVE-2021-29295
D-Link	DSL-2740R (UK_1.01)	/userfs/bin/boa	1 (1)	CVE-2021-29294
D-Link	DAP-2330 (1.01)	/sbin/httpd	4 (3)	CVE-2021-28838 CVE-2021-28839 CVE-2021-28840

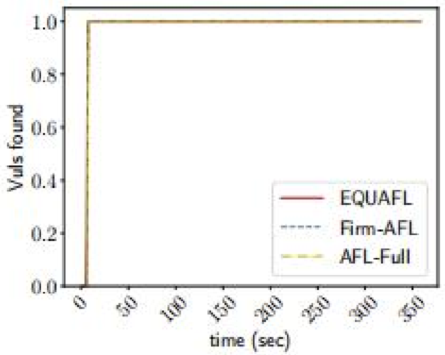
9 NULL pointer dereference vulnerabilities
1 integer overflow vulnerability
6 CVEs

Evaluation - Vulnerability Discovery

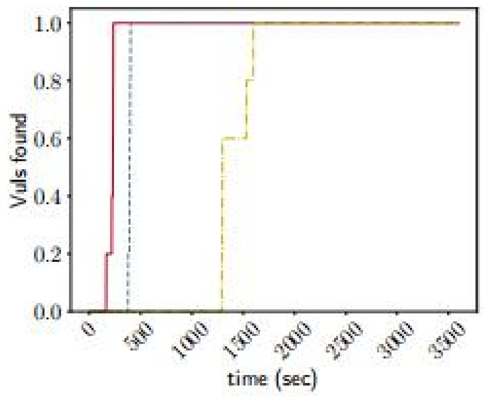
the average number of unique vulnerabilities found over time



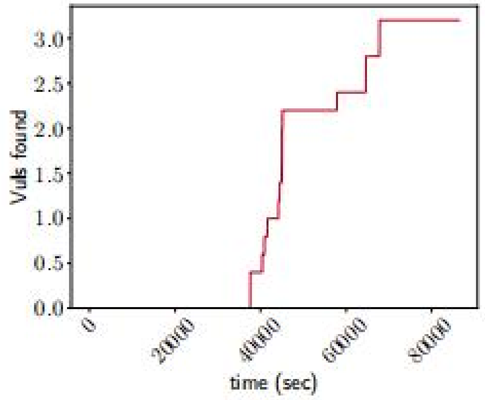
(a) /bin/boa (WN2000RPTv1)



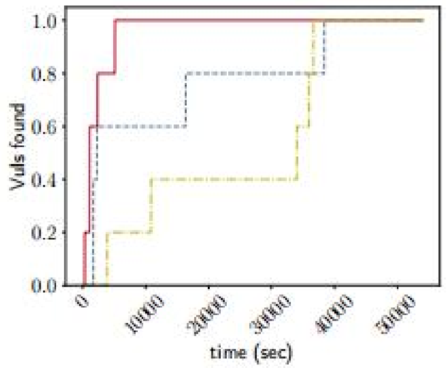
(b) /usr/sbin/uhttpd (WNDRMACv2)



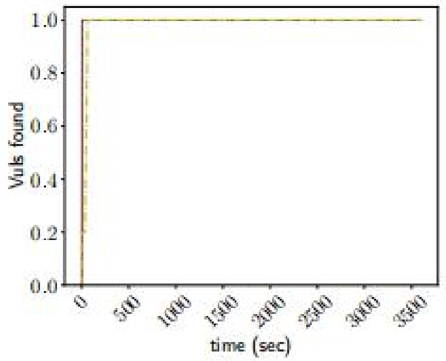
(e) /userfs/bin/boa (DSL-2740R)



(f) /sbin/httpd (DAP-2330)



(c) /sbin/httpd (DIR-825)



(d) /usr/bin/lighttpd (DSP-W215)

EQUAFL can find all vulnerabilities faster than both Firm-AFL and AFL-Full

Evaluation - Vulnerability Discovery

the time to expose the first vulnerability for each technique

Product	EQUAFL	Firm-AFL			AFL-Full		
	μTTE	μTTE	Factor	$\hat{A}12$	μTTE	Factor	$\hat{A}12$
WN2000RPTv1	2220 s	9916 s	4.47	**0.76**	20409 s	9.19	**1.0**
WNDRMACv2	5.0 s	5.2 s	1.04	0.60	5.0 s	1.00	0.50
DIR-825	2011 s	12082 s	6.01	**0.76**	24266 s	12.06	**0.96**
DSP-W215	5 s	5 s	0.96	0.4	39 s	7.37	**1.0**
DSL-2740R	214 s	391 s	1.82	**1.0**	1400 s	6.52	**1.0**
DAP-2330 (vul #1)	42293 s	86400 s	N/A	**1.0**	86400 s	N/A	**1.0**
DAP-2330 (vul #2)	52002 s	86400 s	N/A	**1.0**	86400 s	N/A	**1.0**
DAP-2330 (vul #3)	51972 s	86400 s	N/A	**1.0**	86400 s	N/A	**1.0**
DAP-2330 (vul #4)	80700 s	86400 s	N/A	**1.0**	86400 s	N/A	**1.0**

EQUAFL can find first vulnerability significantly faster than both AFL-Full and Firm-AFL

THREATS TO VALIDITY

- not support access to customized hardware peripherals
- coarse-grained heuristic strategies to bypass the inter-process communication with other applications
- vulnerabilities that spread across multiple applications cannot be revealed

QUESTIONS

- Filesystem State Synchronization部分：为什么要采用本文的方法？为什么不使用更加简单的方法，直接将full-system mode启动后生成的文件系统dump下来使用？
- Network Behavior部分：状态机是如何生成的？具体是如何利用状态机对user-mode下的网络行为进行指导的？
- 本文方法相比Firm-AFL方法的改进在哪些方面？