So Many Fuzzers, So Little Time Experience from Evaluating Fuzzers on the Contiki-NG Network (Hay)Stack



Clément Poncelet clement.poncelet@it.uu.se Uppsala University Uppsala, Sweden



Konstantinos Sagonas kostis@it.uu.se

Uppsala University Uppsala, Sweden National Technical University of Athens Athens, Greece



Nicolas Tsiftes nicolas.tsiftes@ri.se RISE Research Institutes of Sweden Stockholm, Sweden KTH Digital Futures Stockholm, Sweden

Introduction

- Work done as part of the SSF <u>aSSIsT project</u>
 - -Goal: Investigate techniques to secure IoT software
 - -Case Study: Detect and correct bugs in the Contiki-NG OS
 - **Obvious Idea**: Let's use fuzz testing!

The ASE'2022 paper:

- Describes experiences from using a variety of fuzzers
 - -Over a period of more than three years
- Investigates trade-offs in state-of-the-art fuzzing techniques
 - Mutation-based vs. hybrid fuzzers
 - To use or not to use sanitizers when fuzzing?

Fuzz Testing the Contiki-NG Network Stack

Open source OS for resource-constrained IoT devices

HTTP	MQTT	CoAP	SNMP	Application layer
тс	P	UDP /	DTLS	Transport layer
		RF	۶L	
IP	v6	IPv6	ND	Network layer
		ICM	Pv6	
	6LoW	/PAN		Adaptation layer
CS	MA	TS	СН	MAC layer
BL	.E	IEEE 80	02.15.4	Physical layer
		HTTP MQTT TCP IPV6 IPV6 6LoW CSMA BLE	HTTP MQTT CoAP T UDP / IPV6 IPV6 IPV6 IPV6 ICM ICM SCSMA TS6 BLE IEEE 80	HTTPMQTTCoAPSNMPTUDP / DTLSIPICDP / DTLSIP / IPICDP / ODICM > ODICDP / ODCS MATS / DDB EIEEE 8/2.15.4

Initial Fuzzing Attempt Using AFL (ca mid 2018)

american fuzzy lop	2.52b (fuzzer.n	ative)	"unique" crashes
<pre>process timing run time : 0 days, 1 hrs, 0 mi</pre>	in, 7 sec	overall i cycles do	only one bug!
last new path : 0 days, 0 hrs, 7 mi last uniq crash : 0 days, 0 hrs, 5 mi last uniq hang : 0 days, 0 hrs, 12 m	in, 35 sec in, 47 sec nin, 36 sec	total pat uniq crash uniq han	chs les lgs
<pre>cycle progress now processing : 209* (95.43%) paths timed out : 1 (0.46%)</pre>	map coverage map density count coverage	0.70% / 1. 1.35 bits/	42% /tup
now trying : interest 16/8 stage execs : 3392/6247 (54.30%)	 findings in de favored paths new edges on 	epth 107 (48.86% 143 (65.30%	
total execs : 2.85M exec speed : 826.4/sec — fuzzing strategy yields ————————————————————————————————————	total crashes total tmouts	: 68.1k (55 u : 77.8k (46 u - path geome	inique) inique) etry
bit flips : 27/63.7k, 13/63.5k, 12/ byte flips : 2/7961, 11/7654, 3/7312 arithmetics : 25/433k, 3/391k, 2/213k	/63.2k 2 <	levels : pending : pend fav :	20 48 0
known ints : 4/24.8k, 7/109k, 14/198 dictionary : 0/0, 0/0, 0/8096 havoc : 139/828k, 11/417k	3k	own finds : imported : stability :	218 n/a 100.00%
trim : 25.14%/2397, 1.58%		[:pu000: 166%]

Some (Quick) Lessons Learned

The number of "unique" crashes is not a good measure of a fuzzer's efficacy.
It's the number of fixes that matters!

Suggestion:

One should <u>stop</u> a fuzzer soon after it has come up with the first few "unique" crashes, <u>fix</u> the root of the problem, and <u>re-test</u>.

Mutation-based Fuzzers Used

AFL: American Fuzzy Lop (afl-fuzz)

Coverage-based fuzzer with a genetic algorithm. afl-clang-fast

AFL supported by Clang-based instrumentation. Honggfuzz [Google]

Supports evolutionary, feedback-driven fuzzing.

Mopt-AFL [USENIX Security 19]

Guides AFL to select mutations based on a particle swarm optimization algorithm.

Fuzzing vs. Symbolic Execution

- Mutation-based fuzzing:
 - + Explores the program at nearly native speed
 - Unable to exercise "difficult"/"interesting" paths
- Symbolic/concolic execution:
 - + Effective at producing inputs that explore paths guarded by complex conditions
 - Significant run-time overhead

Obvious Idea: Combine these techniques!

Hybrid Fuzzing

- Combine fuzzing with symbolic execution to
 - Increase code coverage
 - Find more bugs
- Hybrid fuzzers:
 - Use the mutation-based component as long as possible
 - Keep track of the coverage achieved
 - When coverage stops increasing, call the symbolic execution engine to provide inputs that (hopefully) exercise some new path

Fuzzing Tools Used

Mutation-based Fuzzers AFL (AFL-gcc) AFL-clang-fast (AFL-cf) Honggfuzz MOpt-AFL (MOpt)

Hybrid Fuzzers

Angora [S&P'19] QSYM [USENIX SECURITY 18] Intriguer [CCS'19] SymCC [USENIX SECURITY 20]

Vulnerabilities in Contiki-NG

PR fixing the bug		Commits and afte	s befor r the f	re ix	Most of the bugs have CVEs	
Id	PR#	Commit SHAs	Protocol	CVE	Error description	Discovered by
uIP-overflow	813	alcba56-ea6c688	uIP	CVE-2020-13985	Integer overflows in IPv6 extension header options.	AFL
uIP-ext-hdr	867	150a3fe-b5d997f	uIP/RPL*		Unsafe IPv6 extension header processing.	AFL
uIP-len	871	b5d997f-8340735	uIP		Unverified IPv6 header length before packet processing.	AFL
6LoWPAN-frag SRH-param ND6-overflow 6LoWPAN-ext-hdr SRH-addr-ptr 6LoWPAN-decompr 6LoWPAN-hdr-iphc	972 1183 1410 1409 1431 1482 1506	6553688-5884a12 beff30b-ebd4cae f417a5f-5bfb30d 5bfb30d-48a3799 3a3dbfe-3f9a601 425587d-aa6e26f 0dada69-6c8373d	6LoWPAN RPL* IPv6 ND 6LoWPAN RPL* 6LoWPAN 6LoWPAN	CVE-2021-21282 CVE-2021-21279 CVE-2021-21280 CVE-2021-21257 CVE-2021-21410	Buffer overflow in 6LoWPAN fragment reassembly. Unverified Source Routing Header (SRH) parameter. Infinite loop in ND6 due to integer wrap around. Out-of-bounds write when processing external header. Unverified address pointer in the Source Routing Header. Out-of-bounds read when decompressing packets. Out-of-bounds read from hc06_ptr in a loop condition.	AFL + external Angora + QSYM QSYM Angora + QSYM AFL MOPT + SYMCC many tools but with ASAN
SNMP-oob-varbinds	1541	285cee0-457fa6c	SNMP		Out-of-bounds read from varbinds in a loop condition.	AFL
SNMP-validate-input	1517	457fa6c-9daacb6	SNMP		Bad length check for SNMP input packets.	AFL
uIP-RPL-classic-prefix	1589	cd208ed-7c2d686	RPL	CVE-2022-35927	Unverified DIO prefix info lengths.	external
uIP-RPL-classic-div	1598	f608483-e427f48	RPL		Division by zero from DIO with O lifetimes.	AFL
6LoWPAN-UDP-hdr	1646	b65cfa3-92783e8	6LoWPAN	CVE-2022-36052	Out-of-bounds read when decompressing UDP header.	MOpt + EffectiveSan
6LoWPAN-payload	1647	92783e8-2dfbaee	6LoWPAN	CVE-2022-36054	Out-of-bounds write when decompressing payload.	MOpt + EffectiveSan
uIP-buf-next-hdr	1648	2dfbaee-80a5479	uIP	CVE-2022-36053	Out-of-bounds read in uipbuf.	MOpt + EffectiveSan
uIP-RPL-classic-sllao	1654	8512556-e58b583	IPv6 ND	CVE-2022-35926	Out-of-bounds read in ND6 option headers.	EffectiveSan into SymCC

increased difficulty

Research Questions

- **RQ.1 (Effectiveness)** Are hybrid fuzzers superior in exposing vulnerabilities and bugs than mutation-based fuzzers?
- RQ.2 (Efficiency) Do some fuzzers employ techniques that allow them to expose bugs fast(er)? If so, which?
- RQ.3 (Consistency) Are there any fuzzer implementations that are able to expose (some of) the bugs in all/most of their runs?

Run <u>focused</u> experiments looking for a particular bug

"Ground Truth" Results

Some bugs are easy

Id AFL-gcc AFL-cf MOPT Honggfuzz Angora QSvm Intriguer SvmCC uIP-overflow 10 00:17:20 10 00:35:40 10 00:03:00 0 © 10 00:53:29 10 00:23:59 10 00:49:58 10 00:01:39 uIP-ext-hdr 10 03:32:17 10 03:23:20 10 00:12:11 10 00:50:12 10 02:44:41 10 00:57:23 9 05:05:31 10 00:11:35 uIP-len 5 06:59:39 0 © 4 09:03:11 0 © 5 08:48:08 5 04:45:32 3 01:24:00 1 01:35:04 uIP-buf-next-hdr 0 © 0 © 0 © 0 © 0 © 0 © uIP-RPL-classic-prefix 6 06:21:52 2 18:52:46 7 03:57:22 0 © 6 09:55:47 10 05:14:50 2 07:11:56 0 ©									
uIP-overflow 10 00:17:20 10 00:35:40 10 00:03:00 0 10 10 00:23:59 10 00:49:58 10 00:01:39 uIP-ext-hdr 10 03:32:17 10 03:23:20 10 00:12:11 10 00:50:12 10 02:44:41 10 00:57:23 9 05:05:31 10 00:11:35 uIP-len 5 06:59:39 0 4 09:03:11 0 5 08:48:08 5 04:45:32 3 01:24:00 1 01:35:04 uIP-buf-next-hdr 0	Id	AFL-gcc	AFL-cf	МОрт	Honggfuzz	Angora	QSym	Intriguer	SymCC
uIP-ext-hdr 10 03:32:17 10 03:23:20 10 00:12:11 10 00:50:12 10 02:44:41 10 00:57:23 9 05:05:31 10 00:11:35 uIP-len 5 06:59:39 0 4 09:03:11 0 5 08:48:08 5 04:45:32 3 01:24:00 1 01:35:04 uIP-buf-next-hdr 0 <td>uIP-overflow</td> <td>10 00:17:20</td> <td>10 00:35:40</td> <td>10 00:03:00</td> <td>0 🕒</td> <td>10 00:53:29</td> <td>10 00:23:59</td> <td>10 00:49:58</td> <td>10 00:01:39</td>	uIP-overflow	10 00:17:20	10 00:35:40	10 00:03:00	0 🕒	10 00:53:29	10 00:23:59	10 00:49:58	10 00:01:39
uIP-len 5 06:59:39 0 4 09:03:11 0 6 5 08:48:08 5 04:45:32 3 01:35:04 uIP-buf-next-hdr 0<	uIP-ext-hdr	10 03:32:17	10 03:23:20	10 00:12:11	10 00:50:12	10 02:44:41	10 00:57:23	9 05:05:31	10 00:11:35
uIP-buf-next-hdr 0 • 0 0 •	uIP-len	5 06:59:39	0 🕒	4 09:03:11	0 🕒	5 08:48:08	5 04:45:32	3 01:24:00	1 01:35:04
uIP-RPL-classic-prefix 6 06:21:52 2 18:52:46 7 03:57:22 0 🕑 6 09:55:47 10 05:14:50 2 07:11:56 0 🕒	uIP-buf-next-hdr	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒
	uIP-RPL-classic-prefix	6 06:21:52	2 18:52:46	7 03:57:22	0 🕒	6 09:55:47	10 05:14:50	2 07:11:56	0 🕒
uIP-RPL-classic-div 7 10:46:12 6 11:09:41 8 07:35:17 4 16:52:41 4 10:54:35 5 08:05:55 3 01:25:26 6 06:00:12	uIP-RPL-classic-div	7 10:46:12	6 11:09:41	<mark>8</mark> 07:35:17	4 16:52:41	4 10:54:35	5 08:05:55	3 01:25:26	6 06:00:12
uIP-RPL-classic-sllao 0 🕒 0 🕑 0 🕒 0 🕒 0 🕒 0 🕒 0 🕒 0	uIP-RPL-classic-sllao	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒

uIP-RPL-classic-div 7 10:46:12 uIP-RPL-classic-sllao 0 🕒 Other bugs are challenging

"Ground Truth" Results

Id	AFL-gcc	AFL-cf	МОрт	Honggfuzz	Angora	QSym	Intriguer	ЅумСС
uIP-overflow uIP-ext-hdr uIP-len uIP-buf-next-hdr	10 00:17:20 10 03:32:17 5 06:59:39 0 (b)	10 00:35:40 10 03:23:20 0 (b) 0 (b) 0 (b) 0 (b)	10 00:03:00 10 00:12:11 4 09:03:11 0 (±)	0 (±) 10 00:50:12 0 (±) 0 (±)	10 00:53:29 10 02:44:41 5 08:48:08 0 (±)	10 00:23:59 10 00:57:23 5 04:45:32 0 (b)	10 00:49:58 9 05:05:31 3 01:24:00 0 (b)	10 00:01:39 10 00:11:35 1 01:35:04 0 (±)
uIP-RPL-classic-prefix uIP-RPL-classic-div uIP-RPL-classic-sllao	6 06:21:52 7 10:46:12 0 🕒	2 18:52:46 6 11:09:41 0 🕒	7 03:57:22 8 07:35:17 0 (±)	0 (±) 4 16:52:41 0 (±)	6 09:55:47 4 10:54:35 0 (±)	10 05:14:50 5 08:05:55 0 •	2 07:11:56 3 01:25:26 0 (*)	0 (b) 6 06:00:12 0 (b)

Id	AFI	L-gcc	A	AFL-cf	1	МОрт	Ho	onggfuzz	A	Angora		QSүм	In	triguer	S	умСС
6LoWPAN-frag	10 00	0:17:19	3	12:26:18	10	00:12:34	0	Ŀ	10	00:18:50	10	<mark>00:08:32</mark>	10	00:23:19	10	01:40:39
SRH-param	10 00	0:21:23	0	Ŀ	10	00:19:44	0	╚	10	00:17:09	10	00:16:49	10	00:18:06	10	<mark>00:07:34</mark>
ND6-overflow	0	Ŀ	0	Ŀ	0	Ŀ	0	╚	0	Ŀ	4	11:15:41	0	╚	0	Ŀ
6LoWPAN-ext-hdr	6 07	7:13:16	1	13:11:24	10	<mark>07:52:40</mark>	0	╚	9	12:58:16	7	02:36:38	7	08:54:21	3	14:56:08
SRH-addr-ptr	8 02	2:14:37	0	Ŀ	8	00:31:38	0	Ŀ	7	03:08:56	10	<mark>00:44:15</mark>	8	00:29:51	0	Ŀ
6LoWPAN-decompr	0	Ŀ	0	Ŀ	0	Ŀ	0	╚	0	Ŀ	0	Ŀ	0	╚	0	Ŀ
6LoWPAN-hdr-iphc	0	Ŀ	0	Ŀ	0	Ŀ	0	╚	0	Ŀ	0	Ŀ	0	╚	0	Ŀ
6LoWPAN-UDP-hdr	0	Ŀ	0	Ŀ	0	Ŀ	0	Ŀ	0	╚	0	Ŀ	0	╚	0	Ŀ
6LoWPAN-payload	0	٩	0	٩	0	٩	0	Ŀ	0	٩	0	Ŀ	0	٩	0	٩

Answers to RQ.1 – RQ.3

- Did not detect any clear superiority of hybrid fuzzers wrt ability to expose bugs compared to mutation-based fuzzers (RQ.1).
- No fuzzer is uniformly superior to all others.
- Three fuzzers (MOpt, SymCC, and QSYM) stand out in terms of ability to expose bugs fast (RQ.2) and in doing so more consistently (RQ.3).
- The consistency and effectiveness of a hybrid fuzzer is dependent on the consistency and effectiveness of its mutation-based component.

"Bonus" Research Question

Quite often, fuzzers are aided by sanitizers. A sanitizer:

- + Exposes and triages bugs more accurately.
- Imposes a non-negligible time overhead (12x).

Few published works investigate this trade-off.

• RQ.4 (Sanitizer Impact) Do sanitizers pay off for their runtime overhead in terms of exposing more vulnerabilities within a timelimited fuzzing run?

Impact	of	4dd	resa	sSa	nitiz	er (ASa	an)
Bug det fewer ti or slov	ected mes wer		Faster	r detectio	n	-	[G	oogle]
Id	A -gcc	AFL-cf	МОрт	Honggfuzz	Angora	QSym	Intriguer	ЅумСС
uIP-overflow uIP-ext-hdr uIP-len	▼ 2 ▼ 01:42:53 ▼ 5	▲ 00:01:06 ▲ 00:53:06	 00:16:53 01:08:33 4 	▼ 00:21:10 —	 ▲ 00:05:25 ▲ 00:27:20 ▼ 5 	 ▲ 00:08:51 ▼ 00:55:37 ▼ 5 	 ▲ 00:12:28 ▲ 1 ▼ 3 	 ▼ 00:29:24 ▼ 02:26:25 ▲ 1
uIP-RPL-classic-prefix uIP-RPL-classic-div	▼ 4 ▼ 7	▼ 2 ▼ 6	▼ 5 ▼ 8	▼ 2	▼ 5 ▼ 3	▼ 9 ▼ 5	▼ 2 ▼ 3	▲ 1 ▼ 6

Bugs detected one more time

Impact of AddressSanitizer (ASan)

Id	AF	L-gcc	AF	L-cf	M	Орт	Ho	nggfuzz	A	ngora	Ç	QSvm	Int	riguer	S	умСС
uIP-overflow uIP-ext-hdr uIP-len	▼ ▼ 0 ▼	2)1:42:53 5	▲ 00 ▲ 00	0:01:06 0:53:06 —	▼ 00 ▼ 01 ▼):16:53 1:08:33 4	•	 00:21:10 		00:05:25 00:27:20 5	▲ (▼ (▼	00:08:51 00:55:37 5	▲ (▲ ▼	00:12:28 1 3	▼ ▼ ▲	00:29:24 02:26:25 1
uIP-RPL-classic-prefix uIP-RPL-classic-div	•	4 7	▼ ▼	2 6	v	5 8	V	2	▼ ▼	5 3	▼ ▼	9 5	▼ ▼	2 3	▲ ▼	1 6

Id	AFI	l-gcc	AF	L-cf]	МОрт	Ho	onggfuzz	A	Angora		QSym	I	ntriguer	S	умСС
6LoWPAN-frag	•	1		5	▼	2		—	▼	00:09:26	V	00:23:27	V	01:39:59		01:00:01
SRH-param	▼	10		_	•	10		_	▼	10	•	10	▼	10	▼	10
6LoWPAN-ext-hdr		4		9		07:34:24		10		1		3		3		7
SRH-addr-ptr	▼	8		_	•	8		_	▼	7	•	10	▼	8		4
6LoWPAN-decompr		10		10		10		10		10		10		10		10
6LoWPAN-hdr-iphc		9		10		10		10		8		8		9		9

Impact of Effective Type Sanitizer

Id	AFL-	gcc/-clang	AF	L-cf	M	Юрт	Ho	nggfuzz	1	Angora		QSчм	I	ntriguer	5	ЅумСС
uIP-overflow		00:06:31	▲ 00):26:21	V 0	0:13:19		-		00:47:22		00:09:03		00:29:43	▼	00:04:13
uIP-ext-hdr		02:29:10	V 00):11:45	V (0:11:53	V	10		02:02:15	•	00:17:45		1	•	00:12:30
uIP-len		5		_		6		10		5		5		7		9
uIP-buf-next-hdr		2		_		3		—		1		2		2		7
uIP-RPL-classic-prefix	•	6	V	2	V	4		-	V	6	V	8	V	06:11:14		5
uIP-RPL-classic-div	•	4	▼	6	•	5	▼	2	▼	2	▼	4	▼	1	▼	02:53:42

Id	AFL	-gcc/-clang	AF	L-cf		МОрт	Hor	nggfuzz	An	gora	Ç	QSvm	In	ntriguer	5	бумСС
6LoWPAN-frag		10	V	2	▼	9		_	▼	10	▼	9	▼	10	▼	10
SRH-param	•	01:52:25		_	▼	02:10:30		—	▼	3	•	00:08:12	▼	01:10:58	▼	00:02:00
6LoWPAN-ext-hdr		4	•	1		07:49:05		_		1		3		3		7
SRH-addr-ptr		2		_		1		_		2	▼	1		2		10
6LoWPAN-decompr		10		_		10		10		10		10		10		10
6LoWPAN-hdr-iphc		10		_		10		1		10		10		10		9
6LoWPAN-payload		10		—		10		—		10		10		10		10

Answer(s) to RQ.4

Sanitizers:

- Make detection of "easy to expose" bugs slower.
- Make fuzzers more consistent in detecting bugs.
- Are crucial for detection of "difficult to expose" bugs and vulnerabilities.

Overall, we find that sanitizers pay off for their (non-negligible) overhead when the "easy" bugs have been fixed.

Read the Paper for

- More tables and experiments.
- More lessons learned.
- More suggestions on how to compare fuzzers.
- Related work.
 - Other suites for benchmarking fuzzers.
 - Comparison with results reported in "similar" papers (using different sets of fuzzers).
- Information about the paper's artifact.

https://github.com/assist-project/so-many-fuzzers-artifact

In Summary

Fuzzing Tools Used

Mutation-based Fuzzers

Hybrid Fuzzers

- AFL (AFL-gcc)
- AFL-clang-fast (AFL-cf)
- Honggfuzz
- MOpt-AFL (MOpt)

- Angora [S&P'19]
- QSYM [USENIX SECURITY 18]
- Intriguer [CCS'19]
- SymCC [USENIX SECURITY 20]

"Ground Truth" Results

Id	A	FL-gcc		AFL-cf		МОрт	Ho	onggfuzz		Angora		QSym	Iı	ntriguer	:	бумСС
uIP-overflow	10	00:17:20	10	00:35:40	10	00:03:00	0	٩	10	00:53:29	10	00:23:59	10	00:49:58	10	00:01:39
uIP-ext-hdr	10	03:32:17	10	03:23:20	10	00:12:11	10	00:50:12	10	02:44:41	10	00:57:23	9	05:05:31	10	00:11:35
uIP-len	5	06:59:39	0	٩	4	09:03:11	0	٩	5	08:48:08	5	04:45:32	3	01:24:00	1	01:35:04
uIP-buf-next-hdr	0	٩	0	٢	0	٢	0	٩	0	G	0	۲	0	٩	0	٩
uIP-RPL-classic-prefix	6	06:21:52	2	18:52:46	7	03:57:22	0	٩	6	09:55:47	10	05:14:50	2	07:11:56	0	٢
uIP-RPL-classic-div	7	10:46:12	6	11:09:41	8	07:35:17	4	16:52:41	4	10:54:35	5	08:05:55	3	01:25:26	6	06:00:12
uIP-RPL-classic-sllao	0	۲	0	٢	0	٢	0	۲	0	٩	0	۲	0	۲	0	٩
Id	A	AFL-gcc		AFL-cf	1	МОрт	Ho	nggfuzz	A	ngora	(QSym	In	triguer	s	умСС
Id 6LoWPAN-frag	A 10	AFL-gcc 00:17:19	3	AFL-cf 12:26:18	N 10	МОрт 00:12:34	Ho	nggfuzz	A 10	.ngora 00:18:50	10	QSум 00:08:32	In 10	triguer 00:23:19	S 10	умСС 01:40:39
Id 6LoWPAN-frag SRH-param	A 10 10	AFL-gcc 00:17:19 00:21:23	3	AFL-cf 12:26:18 ©	10 10	ИОрт 00:12:34 00:19:44	Ho 0 0	nggfuzz	A 10 10	ngora 00:18:50 00:17:09	10 10	QSум 00:08:32 00:16:49	In 10 10	triguer 00:23:19 00:18:06	S 10 10	хмСС 01:40:39 <mark>00:07:34</mark>
Id 6LoWPAN-frag SRH-param ND6-overflow	A 10 10 0	AFL-gcc 00:17:19 00:21:23	3 0 0	AFL-cf 12:26:18 ⑤	10 10 0	МОрт 00:12:34 00:19:44 ©	Hor 0 0 0	nggfuzz © ©	A 10 10 0	00:18:50 00:17:09	10 10 4	QSym 00:08:32 00:16:49 11:15:41	In 10 10 0	triguer 00:23:19 00:18:06 ©	5 10 10 0	умСС 01:40:39 00:07:34 ©
Id 6LoWPAN-frag SRH-param NDG-overflow 6LoWPAN-ext-hdr	A 10 10 0 6	AFL-gcc 00:17:19 00:21:23 9 07:13:16	3 0 0 1	AFL-cf 12:26:18 (5) (5) 13:11:24	10 10 0 10	MOPT 00:12:34 00:19:44 © 07:52:40	Hor 0 0 0	nggfuzz © © ©	A 10 10 9	ngora 00:18:50 00:17:09 (*) 12:58:16	10 10 4 7	QSум 00:08:32 00:16:49 11:15:41 02:36:38	In 10 10 0 7	triguer 00:23:19 00:18:06 © 08:54:21	5 10 10 0 3	умСС 01:40:39 00:07:34 © 14:56:08
Id 6LoWPAN-frag SRH-param NDG-overflow 6LoWPAN-ext-hdr SRH-addr-ptr	A 10 10 0 6 8	AFL-gcc 00:17:19 00:21:23 0 (5) 07:13:16 02:14:37	3 0 0 1 0	AFL-cf 12:26:18 © 13:11:24 ©	10 10 0 10 8	MOPT 00:12:34 00:19:44 © 07:52:40 00:31:38	0 0 0 0 0	nggfuzz © © © ©	A 10 10 0 9 7	ngora 00:18:50 00:17:09 () 12:58:16 03:08:56	10 10 4 7 10	QSум 00:08:32 00:16:49 11:15:41 02:36:38 00:44:15	In 10 10 0 7 8	triguer 00:23:19 00:18:06 (9 08:54:21 00:29:51	5 10 10 0 3 0	умСС 01:40:39 00:07:34 © 14:56:08 ©
Id GLoWPAN-frag SRH-param NDG-overflow GLoWPAN-ext-hdr SRH-addr-ptr GLOWPAN-decomp	A 10 10 0 6 8 0	AFL-gcc 00:17:19 00:21:23 07:13:16 02:14:37 ©	3 0 0 1 0 0	AFL-cf 12:26:18 (b) (c) 13:11:24 (c) (c)	10 10 0 10 8 0	MOPT 00:12:34 00:19:44 © 07:52:40 00:31:38 ©	Hor 0 0 0 0 0 0 0	nggfuzz © © © © © ©	A 10 10 9 7 0	ngora 00:18:50 00:17:09 () 12:58:16 03:08:56 ()	10 10 4 7 10 0	QSум 00:08:32 00:16:49 11:15:41 02:36:38 00:44:15 ©	In 10 10 7 8 0	triguer 00:23:19 00:18:06 © 08:54:21 00:29:51 ©	5 10 10 0 3 0 0	умСС 01:40:39 00:07:34 © 14:56:08 © ©
Id GLOWPAN-frag SRH-param NDG-overflow GLOWPAN-ext-hdr SRH-addr-ptr GLOWPAN-decompr GLOWPAN-hdr-iptr	A 10 10 0 6 8 0 0	AFL-gcc 00:17:19 00:21:23 07:13:16 02:14:37 6 02:14:37 6 5	3 0 0 1 0 0 0 0	AFL-cf 12:26:18 (b) 13:11:24 (c) (c) (c) (c)	10 10 0 10 8 0 0	MOPT 00:12:34 00:19:44 © 07:52:40 00:31:38 © ©	Hor 0 0 0 0 0 0 0 0	nggfuzz	A 10 10 9 7 0 0	00:18:50 00:17:09 12:58:16 03:08:56 © ©	10 10 4 7 10 0 0	QSYM 00:08:32 00:16:49 11:15:41 02:36:38 00:44:15 © ©	In 10 0 7 8 0 0	triguer 00:23:19 00:18:06 08:54:21 00:29:51 © ©	5 10 10 0 3 0 0 0 0	умСС 01:40:39 00:07:34 © 14:56:08 © © ©
Id 6LoWPAN-frag SRH-param ND6-overflow 6LoWPAN-ext-hdr SRH-addr-ptr 6LoWPAN-decompr 6LoWPAN-dc-iphc 6LoWPAN-hdr-iphc	A 10 10 0 0 6 8 0 0 0	AFL-gcc 00:17:19 00:21:23 07:13:16 02:14:37 6 6 02:14:37 6 6 6	3 0 0 1 0 0 0 0 0	AFL-cf 12:26:18 © 13:11:24 © © ©	10 10 0 10 8 0 0 0	MOPT 00:12:34 00:19:44 © 07:52:40 00:31:38 © © ©	Hor 0 0 0 0 0 0 0 0 0 0	nggfuzz	A 10 0 9 7 0 0 0	00:18:50 00:17:09 12:58:16 03:08:56 © © ©	10 10 4 7 10 0 0 0	QSYM 00:08:32 00:16:49 11:15:41 02:36:38 00:44:15 © © ©	In 10 0 7 8 0 0 0	triguer 00:23:19 00:18:06 08:54:21 00:29:51 © © ©	S 10 0 3 0 0 0 0 0	YMCC 01:40:39 00:07:34 (*) 14:56:08 (*) (*) (*) (*) (*) (*)

Research Questions

- RQ.1 (Effectiveness) Are hybrid fuzzers superior in exposing bugs and vulnerabilities than mutation-based fuzzers?
- **RQ.2 (Efficiency)** Do some fuzzers employ techniques that allow them to expose bugs fast? If so, which?
- RQ.3 (Consistency) Are there any fuzzer implementations that are able to expose (some of) the bugs in all/most of their runs?

Impact of AddressSanitizer (ASan)

Id	A	FL-gcc		AFL	cf	M	Юрт	Ho	nggfuz	z	Ango	a		QSYM	I	Intriguer		SYMCO
uIP-overflow	•		2	A 00:0	1:06	V 0	0:16:53		-	-	▲ 00:05	:25		00:08:51		00:12:28	,	00:29:
uIP-ext-hdr		01:42:5	3	▲ 00:5	3:06	V 0	1:08:33		00:21:10		▲ 00:27	:20		00:55:37		1	,	• 02:26:
uIP-len	•		5		-	•	4		-	-	•	5	•	5		3		•
P-RPL-classic-prefix			4	•	2		5		_	-		5		9		2		
										_								
uIP-RPL-classic-div Id	AFI	gec	7 AF	▼ L-cf	6 N	• ИОрт	8 Ho	ongg	fuzz	2 A	ngora	3	• QS	5 YM	Int	triguer	5	YMCC
Id	AFI	-gec	AF	L-cf	6 N	• ИОрт	8 He	ongg	fuzz	2 A	ngora	3	QS	5 YM	Int	triguer	5	• • • • • • • • • • • • • •
Id SPH-param	AFI	2-gec	7 AF	L-cf	6 	• ИОрт	8 He	ongg	fuzz 	2 A	ngora 00:09:26	3	QS:	5 YM :23:27 10	Int	riguer 01:39:59	s	• • • • • • • • • • • • • •
IIP-RPL-classic-div Id 6LoWPAN-frag SRH-param 6LoWPAN-grad	AFI	1 10 4	7 AF	L-cf	6 N	• ИОрт 107:34:2	8 Ho 2 10 24	ongg	fuzz 		ngora 00:09:26 10	3	QS:	5 YM :23:27 10 3	Int (triguer 01:39:59 10 3	s	• • • • • • • • • • • • • •
Id 6LoWPAN-frag SRH-param 6LoWPAN-ext-hdr SRH-add-ptr	AFI	1 10 4 8	7 AF	L-cf	6 N	• ИОРТ 1 07:34:2	8 Ho 2 10 24	ongg	fuzz 10 		ngora 00:09:26 10 1 7	3	QS:	5 YM :23:27 10 3 10	Int	3 triguer 01:39:59 10 3 8	S	• • • • • • • • • • • • • •
IDP-RPL-classic-div Id 6LoWPAN-frag SRH-param 6LoWPAN-ext-hdr SRH-addr-ptr 6LoWPAN-decompr	AFI	1 10 4 8 10	AF	L-cf 5 - 9 - 10	6	ИОрт 1 07:34:2	8 Ho 2 10 24 8	ongg	fuzz 		ngora 00:09:26 10 1 7	3	QS 00:	5 YM :23:27 10 3 10 10	Int v (riguer 01:39:59 10 3 8 10	5	• • • • • • • • • • • • • •