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

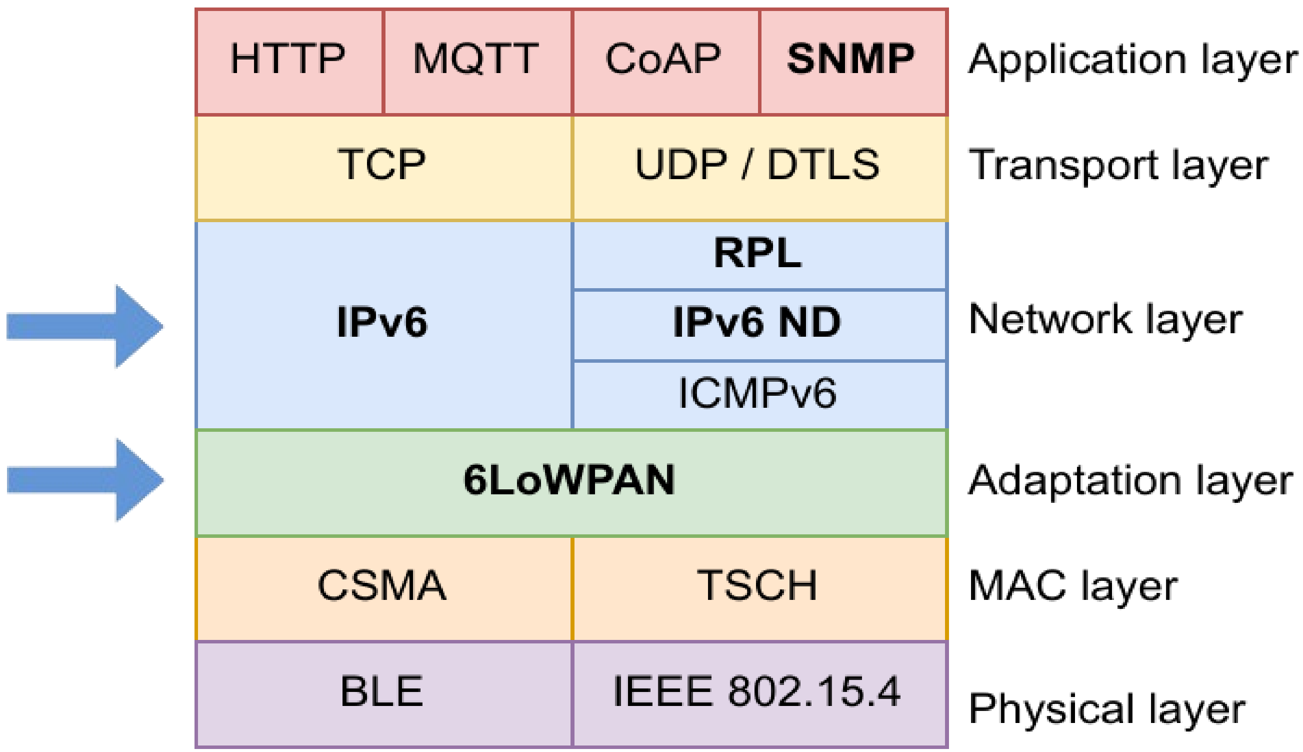
- Work done as part of the SSF aSSIsT project
  - **Goal**: Investigate techniques to secure IoT software
  - **Case Study**: Detect and correct bugs in the Contiki-NG OS
  - **Obvious Idea**: Let's use fuzz testing!

The ASE'2022 paper:

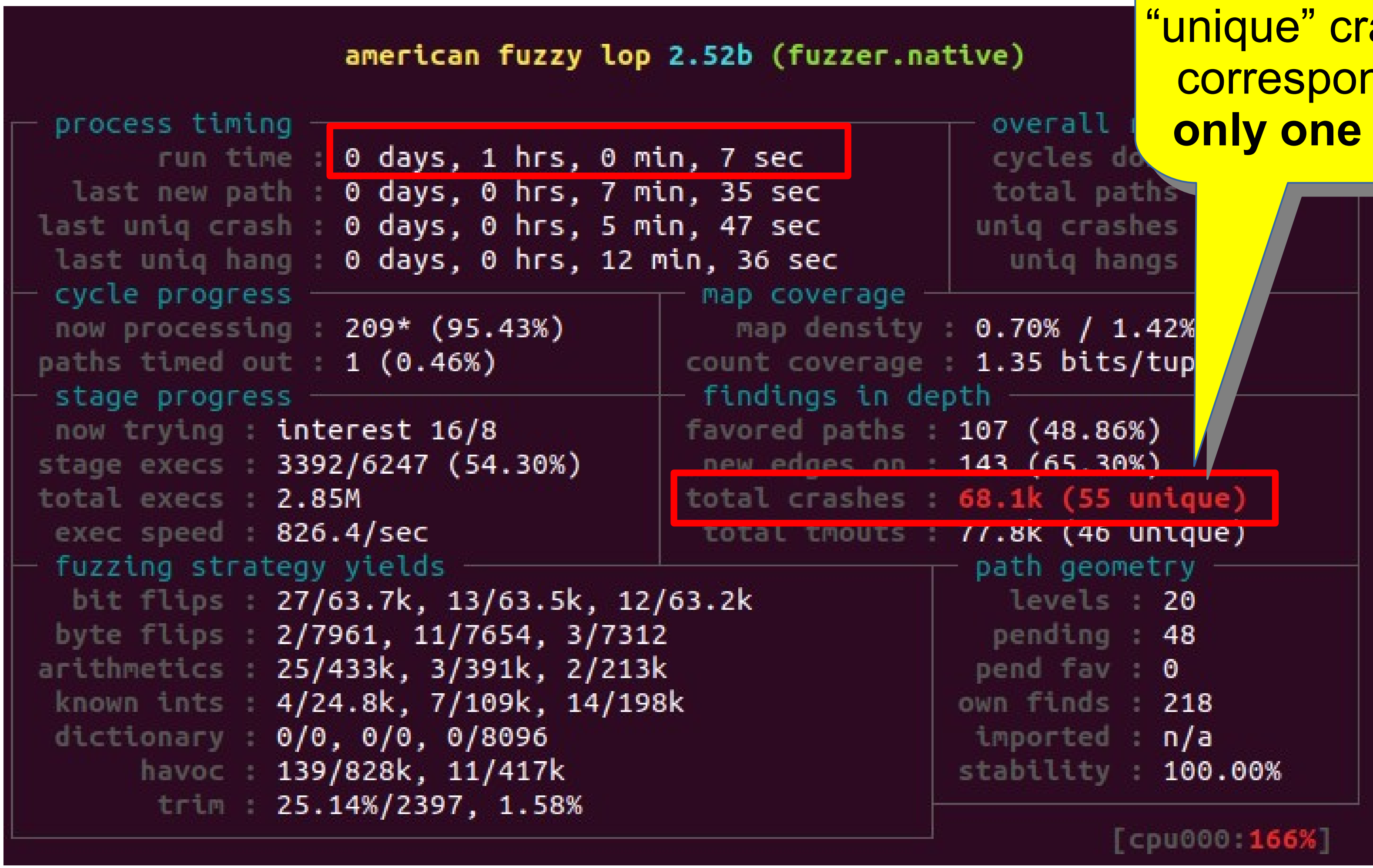
- Describes experiences from using a variety of fuzzers
  - Over a period of more than three years
- Investigates trade-offs in state-of-the-art fuzzing techniques
  - Mutation-based vs. hybrid fuzzers
  - To use or not to use sanitizers when fuzzing?

Fuzz Testing the Contiki-NG Network Stack

Open source OS for resource-constrained IoT devices



Initial Fuzzing Attempt Using AFL
(ca mid 2018)



All these “unique” crashes correspond to **only one** bug!

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 stop a fuzzer soon after it has come up with the first few “unique” crashes, fix the root of the problem, and re-test.

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 before and after the fix

Most of the bugs have CVEs

Id	PR♯	Commit SHAs	Protocol	CVE	Error description	Discovered by
uIP-overflow	813	a1cba56-ea6c688	uIP		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	CVE-2020-13985	Unverified IPv6 header length before packet processing.	AFL
6LoWPAN-frag	972	6553688-5884a12	6LoWPAN		Buffer overflow in 6LoWPAN fragment reassembly.	AFL + external
SRH-param	1183	beff30b-ebd4cae	RPL*	CVE-2021-21282	Unverified Source Routing Header (SRH) parameter.	Angora + QSYM
ND6-overflow	1410	f417a5f-5bfb30d	IPv6 ND	CVE-2021-21279	Infinite loop in ND6 due to integer wrap around.	QSYM
6LoWPAN-ext-hdr	1409	5bfb30d-48a3799	6LoWPAN	CVE-2021-21280	Out-of-bounds write when processing external header.	Angora + QSYM
SRH-addr-ptr	1431	3a3dbfe-3f9a601	RPL*	CVE-2021-21257	Unverified address pointer in the Source Routing Header.	AFL
6LoWPAN-decompr	1482	425587d-aa6e26f	6LoWPAN	CVE-2021-21410	Out-of-bounds read when decompressing packets.	MOPT + SYMCC
6LoWPAN-hdr-iphc	1506	0dada69-6c8373d	6LoWPAN		Out-of-bounds read from hc06_ptr in a loop condition.	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 0 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 focused experiments looking for a particular bug

“Ground Truth” Results

Some bugs are easy

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
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 ⏲	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-sllao	0 ⏲	0 ⏲	0 ⏲	0 ⏲	0 ⏲	0 ⏲	0 ⏲	0 ⏲

Other bugs are challenging

uIP-RPL-classic-div | 7 10:46:12
uIP-RPL-classic-sllao | 0 ⏲

"Ground Truth" Results

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
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 🕒	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-sllao	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
6LoWPAN-frag	**10** 00:17:19	3 12:26:18	**10** 00:12:34	0 🕒	**10** 00:18:50	**10** 00:08:32	**10** 00:23:19	**10** 01:40:39
SRH-param	**10** 00:21:23	0 🕒	**10** 00:19:44	0 🕒	**10** 00:17:09	**10** 00:16:49	**10** 00:18:06	**10** 00:07:34
ND6-overflow	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	4 11:15:41	0 🕒	0 🕒
6LoWPAN-ext-hdr	6 07:13:16	1 13:11:24	**10** 07:52:40	0 🕒	9 12:58:16	7 02:36:38	7 08:54:21	3 14:56:08
SRH-addr-ptr	8 02:14:37	0 🕒	8 00:31:38	0 🕒	7 03:08:56	**10** 00:44:15	8 00:29:51	0 🕒
6LoWPAN-decompr	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒
6LoWPAN-hdr-iphc	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒
6LoWPAN-UDP-hdr	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒
6LoWPAN-payload	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒	0 🕒

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 time-limited fuzzing run?

Impact of AddressSanitizer (ASan)

[Google]

Bug detected fewer times or slower

Faster detection

Id	A -gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
uIP-overflow	▼ 2	▲ 00:01:06	▼ 00:16:53	—	▲ 00:05:25	▲ 00:08:51	▲ 00:12:28	▼ 00:29:24
uIP-ext-hdr	▼ 01:42:53	▲ 00:53:06	▼ 01:08:33	▼ 00:21:10	▲ 00:27:20	▼ 00:55:37	▲ 1	▼ 02:26:25
uIP-len	▼ 5	—	▼ 4	—	▼ 5	▼ 5	▼ 3	▲ 1
uIP-RPL-classic-prefix	▼ 4	▼ 2	▼ 5	—	▼ 5	▼ 9	▼ 2	▲ 1
uIP-RPL-classic-div	▼ 7	▼ 6	▼ 8	▼ 2	▼ 3	▼ 5	▼ 3	▼ 6

Bugs detected one more time

Impact of AddressSanitizer (ASan)

[Google]

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
`uIP-overflow`	▼ 2	▲ 00:01:06	▼ 00:16:53	—	▲ 00:05:25	▲ 00:08:51	▲ 00:12:28	▼ 00:29:24
`uIP-ext-hdr`	▼ 01:42:53	▲ 00:53:06	▼ 01:08:33	▼ 00:21:10	▲ 00:27:20	▼ 00:55:37	▲ 1	▼ 02:26:25
`uIP-len`	▼ 5	—	▼ 4	—	▼ 5	▼ 5	▼ 3	▲ 1
`uIP-RPL-classic-prefix`	▼ 4	▼ 2	▼ 5	—	▼ 5	▼ 9	▼ 2	▲ 1
`uIP-RPL-classic-div`	▼ 7	▼ 6	▼ 8	▼ 2	▼ 3	▼ 5	▼ 3	▼ 6

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
`6LoWPAN-frag`	▼ 1	▲ 5	▼ 2	—	▼ 00:09:26	▼ 00:23:27	▼ 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

[PLDI'18]

Id	AFL-gcc/-clang	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
uIP-overflow	▲ 00:06:31	▲ 00:26:21	▼ 00:13:19	—	▲ 00:47:22	▲ 00:09:03	▲ 00:29:43	▼ 00:04:13
uIP-ext-hdr	▲ 02:29:10	▼ 00:11:45	▼ 00:11:53	▼ 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	▼ 2	▼ 4	—	▼ 6	▼ 8	▼ 06:11:14	▲ 5
uIP-RPL-classic-div	▼ 4	▼ 6	▼ 5	▼ 2	▼ 2	▼ 4	▼ 1	▼ 02:53:42

Id	AFL-gcc/-clang	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
6LoWPAN-frag	▼ 10	▼ 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

- *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]

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?

“Ground Truth” Results

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
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 ◷	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-sllao	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
6LoWPAN-frag	**10** 00:17:19	3 12:26:18	**10** 00:12:34	0 ◷	**10** 00:18:50	**10** 00:08:32	**10** 00:23:19	**10** 01:40:39
SRH-param	**10** 00:21:23	0 ◷	**10** 00:19:44	0 ◷	**10** 00:17:09	**10** 00:16:49	**10** 00:18:06	**10** 00:07:34
ND6-overflow	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	4 11:15:41	0 ◷	0 ◷
6LoWPAN-ext-hdr	6 07:13:16	1 13:11:24	**10** 07:52:40	0 ◷	9 12:58:16	7 02:36:38	7 08:54:21	3 14:56:08
SRH-addr-ptr	8 02:14:37	0 ◷	8 00:31:38	0 ◷	7 03:08:56	**10** 00:44:15	8 00:29:51	0 ◷
6LoWPAN-decompr	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷
6LoWPAN-hdr-iphc	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷
6LoWPAN-UDP-hdr	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷
6LoWPAN-payload	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷	0 ◷

Impact of AddressSanitizer (ASan)

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
uIP-overflow	▼ 2	▲ 00:01:06	▼ 00:16:53	—	▲ 00:05:25	▲ 00:08:51	▲ 00:12:28	▼ 00:29:24
uIP-ext-hdr	▼ 01:42:53	▲ 00:53:06	▼ 01:08:33	▼ 00:21:10	▲ 00:27:20	▼ 00:55:37	▲ 1	▼ 02:26:25
uIP-len	▼ 5	—	▼ 4	—	▼ 5	▼ 5	▼ 3	▲ 1
uIP-RPL-classic-prefix	▼ 4	▼ 2	▼ 5	—	▼ 5	▼ 9	▼ 2	▲ 1
uIP-RPL-classic-div	▼ 7	▼ 6	▼ 8	▼ 2	▼ 3	▼ 5	▼ 3	▼ 6

Id	AFL-gcc	AFL-cf	MOpt	Honggfuzz	Angora	QSym	Intriguer	SymCC
6LoWPAN-frag	▼ 1	▲ 5	▼ 2	—	▼ 00:09:26	▼ 00:23:27	▼ 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