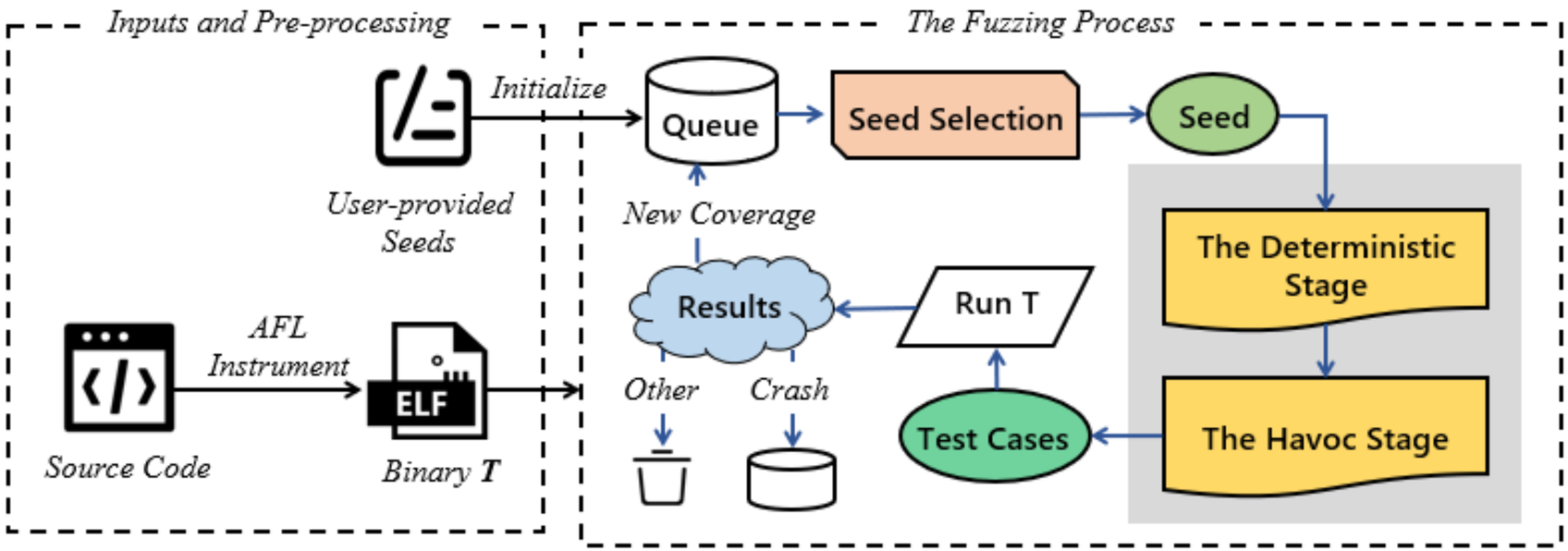
AFLTurbo: Speed up Path Discovery for Greybox Fuzzing

Lei Sun[1] , Xumei Li[1] , Haipeng Qu[1] , Xiaoshuai Zhang[2]

1. Ocean University of China
2. Queen Mary University of London

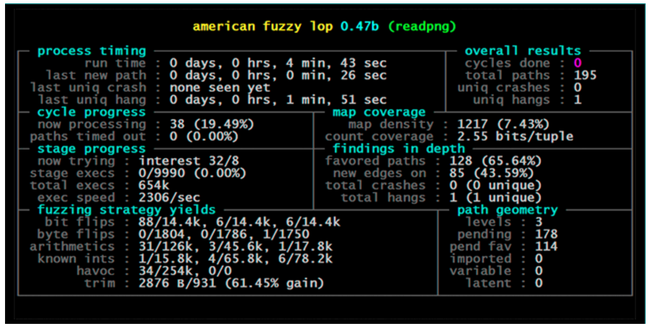
Coverage-based Greybox Fuzzing

Coverage-based Greybox Fuzzing is the state of the art method to find vulnerabilities.



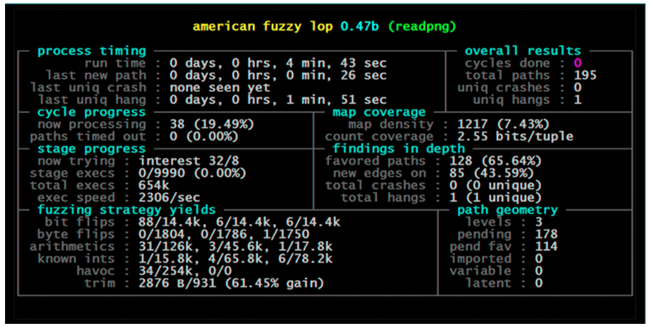
AFL

American Fuzz Lop(AFL) is one of the most famous CGF.



AFLFast **FairFuzz** **CollAFL** **QSYM** **.....**

But…



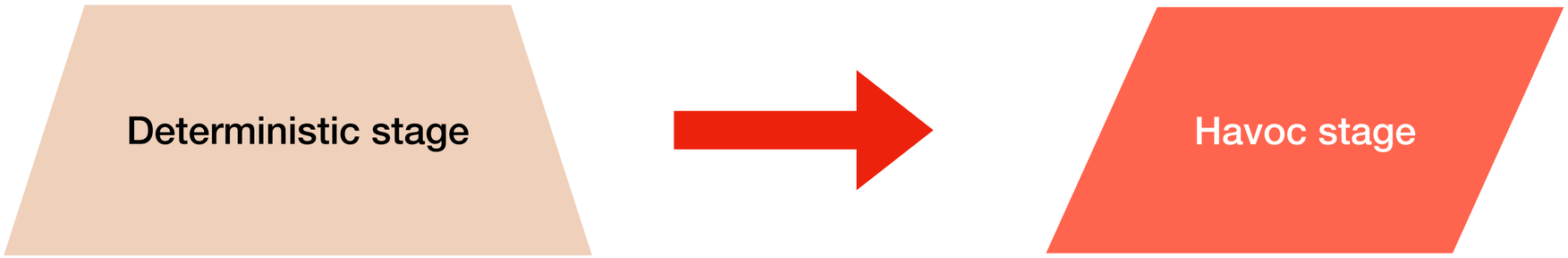
None of previous work has pay attention to:

C1: Aggressively mutation overhead

C2: Ineffective mutation region selection

Which prevent AFL from making progress efficiently

AFL mutation



Bit flip

Byte flip

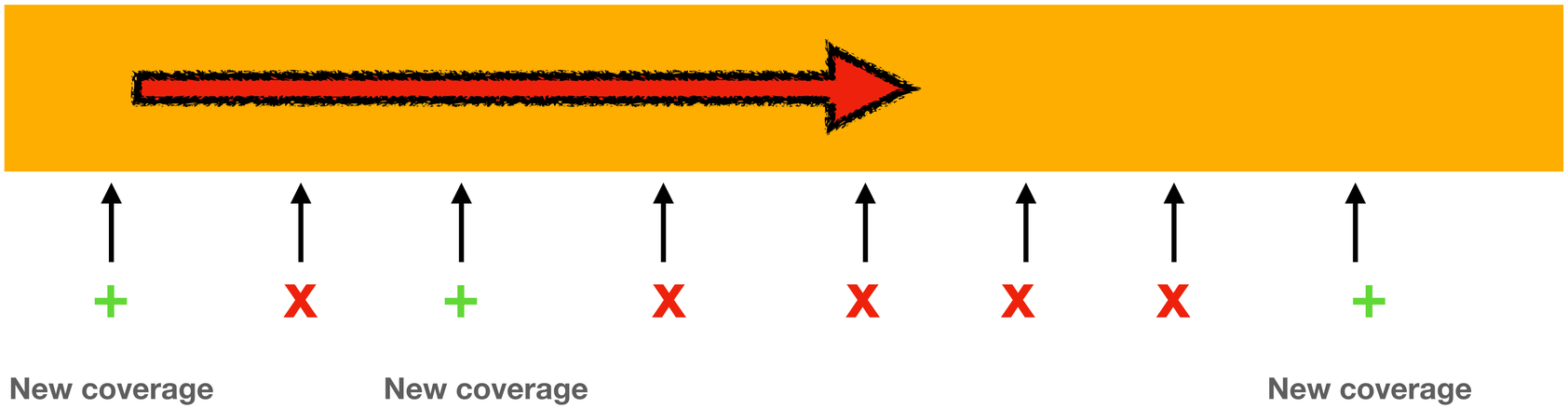
Arithmetic operation

Interesting value replacement

Token replacement

Randomly mangled input

Deterministic mutation



L: length of input

Number of new inputs generated by bit flip = L * 8 * 3

Aggressively growing mutation overhead

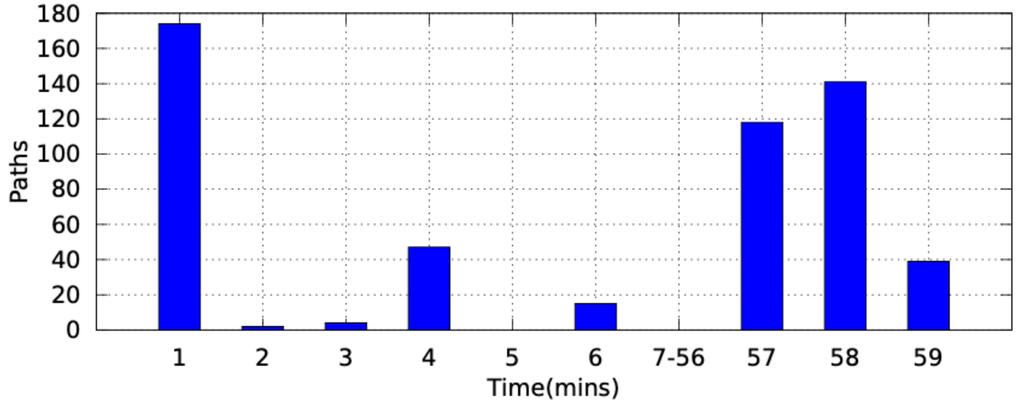
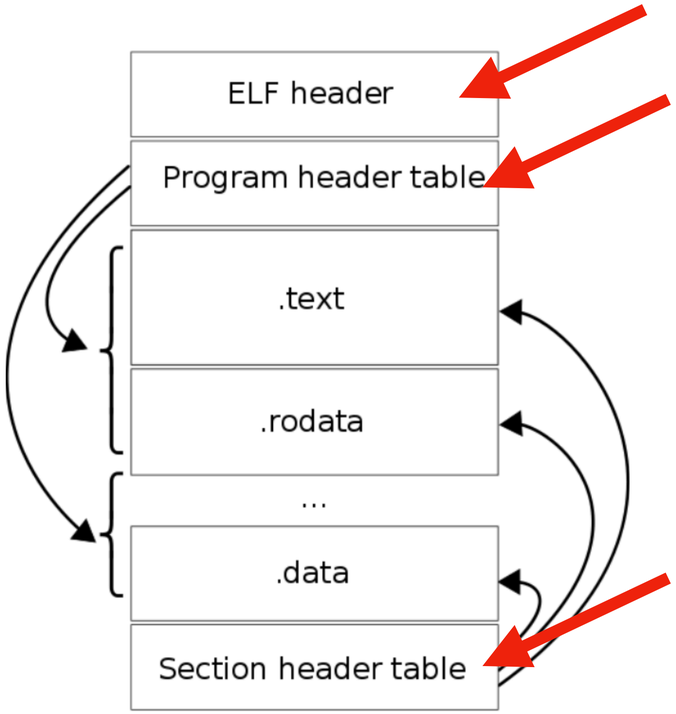


Fig. 1: Number of newly discovered paths of different periods in an hour during the *bitflip 1/1* phase, when fuzzing *readelf*. AFL doesn't find any new paths from the 7th minute to the 56th minute.

A single mutation type (*bitflip 1/1*) costs one hour.

AFL doesn't find new coverage for 49 minutes.

Ineffective mutation region selection



ELF format

AFL mutates every bytes.

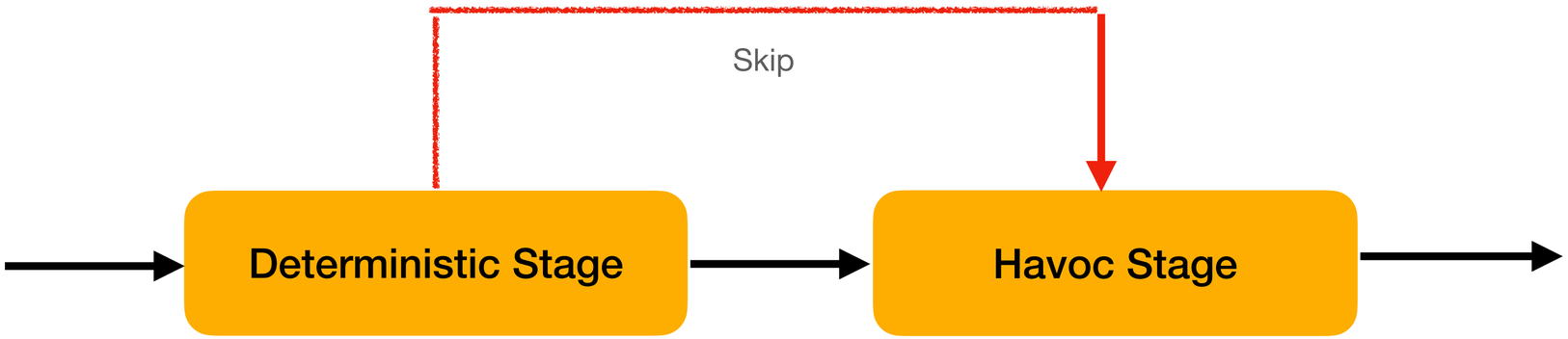
While *readelf* only processes a small portion of data

Our methods

- C1:
  - Interruptible mutation
- C2:
  - Locality-based mutation
  - Hotspot-aware fuzzing

Interruptible mutation

- Design a hang monitor to monitor hang status
- Break out deterministic stage based on two thresholds
  - Time
  - New discovered path count

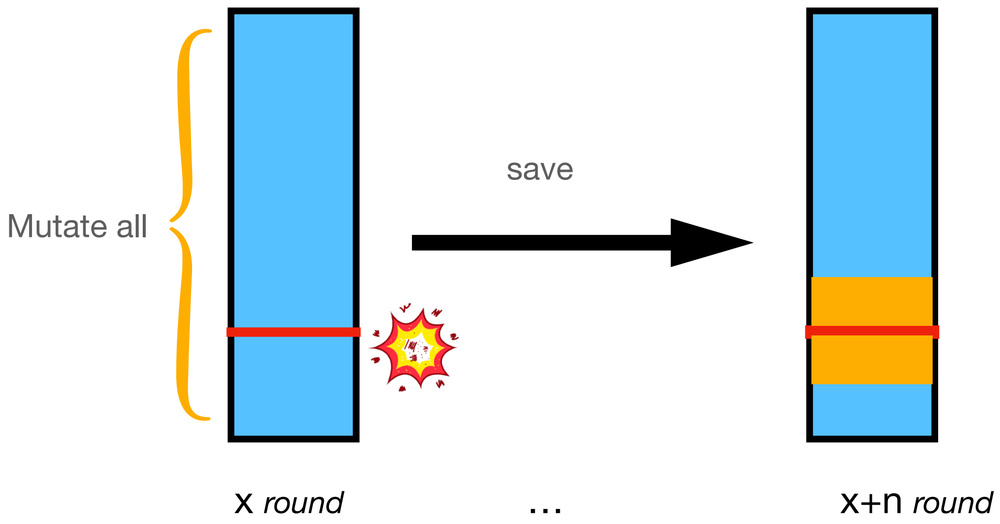


- C2: Ineffective mutation region selection

- How to find useful regions in test cases

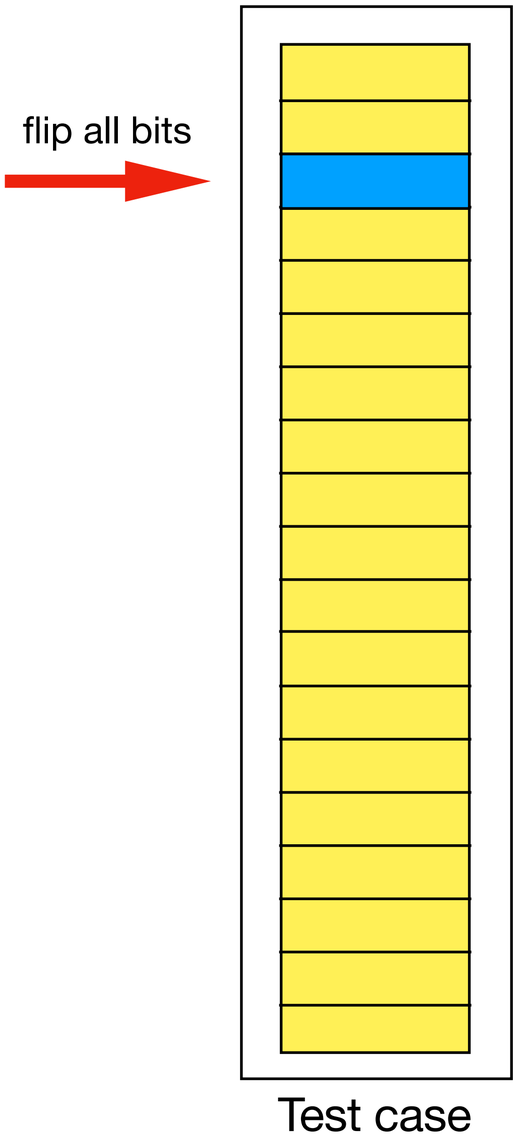
Locality-based mutation

- Inspired by the principle of locality in computer science
- Record mutation offset when generate a new test case
- Focus on mutating regions around the recorded offset in future rounds



Hotspot-aware fuzzing

1. Pre-evaluate test case in a coarse-grained way
   - Split test case into large number of chunks
   - Flip all bits in one chunk and check coverage change
   - Record these chunks that affect coverage
2. Only mutate recorded chunks



Evaluation

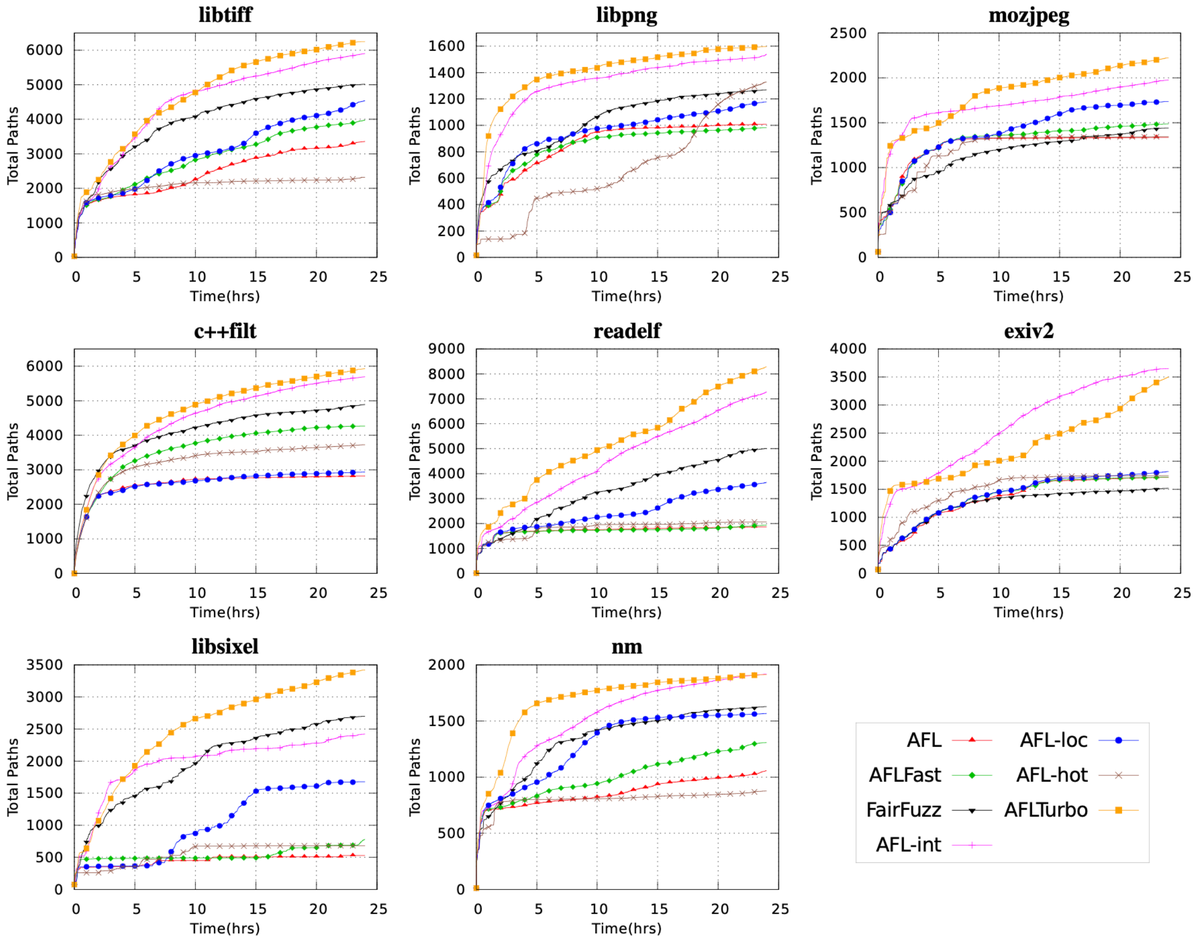
- Comparison with other fuzzers (AFL, AFLFast, FairFuzz)
  - Code coverage
  - Vulnerability discovery

- Fuzzing open source projects

Evaluation - code coverage

- 8 open source programs (libtiff, libpng, mozjpeg, readelf, exiv2, etc)
- 24 hours for each testing
- 10 time to reduce the randomness of fuzzing
- seeds are selected using two criteria:
  - small & valid: to generate more paths in 24 hours
  - randomly chosen from internet

Evaluation - code coverage



- AFLTurbo find 141%, 101%, 41% more paths than AFL, AFLFast, and FairFuzz

Evaluation - vulnerability discovery

- Standard benchmark: LAVA-M
- Found 124 bugs in LAVA-M, while AFL, AFLFast and FairFuzz only found 8, 4, 20

Fuzzer	base64		md5sum		uniq		who		total	
	crash	path	crash	path	crash	path	crash	path	crash	path
AFL	0	1432	0	4350	0	1221	8	1183	8	8186
AFLFast	0	1480	0	4341	0	1279	4	1377	4	8477
FairFuzz	15	1811	0	4699	1	1255	4	1506	20	9271
AFL-int	11	1811	3	5910	15	1261	14	1845	43	10827
AFL-loc	0	1377	0	4259	0	1272	6	1295	6	8203
AFL-hot	0	1493	0	4851	3	1375	18	1842	21	9561
AFLTurbo	87	1847	3	6586	17	1321	17	1845	124	11599

Evaluation - new vulnerabilities

Project	CVE-ID	Description	Severity
ffjpeg	CVE-2019-19887	NULL pointer dereference	Medium
ffjpeg	CVE-2019-19888	dividing by zero	Medium
libIEC61850	CVE-2019-19930	integer overflow	Medium
libIEC61850	CVE-2019-19931	heap overflow	High
libIEC61850	CVE-2019-19944	out of bound access	Medium
libIEC61850	CVE-2019-19957	out of bound access	Medium
libIEC61850	CVE-2019-19958	integer overflow	Medium
libIEC61850	CVE-2020-7054	heap overflow	High
stb	CVE-2020-6617	assertion failure	High
stb	CVE-2020-6618	heap overflow	High
stb	CVE-2020-6619	assertion failure	High
stb	CVE-2020-6620	heap overflow	High
stb	CVE-2020-6621	heap overflow	High
stb	CVE-2020-6622	heap overflow	High
stb	CVE-2020-6623	assertion failure	High
libsixel	CVE-2019-20056	assertion failure	Medium
libsixel	CVE-2019-20205	integer overflow	High
libsixel	CVE-2020-11721	uninitialized variable	Medium

- 20 new vulnerabilities, 18 of them are assigned CVE IDs
- 10 vulnerabilities are rated as high severity and 8 are rated as medium severity.

Conclusion

- We design three new techniques to improve AFL.
- Increase code coverage & improve the capability of bug detection
- Evaluation found and 20 bugs, 18 CVEs
- Source code: https://github.com/sleicasper/aflturbo

Thank You!