Fuzzing results

Seyed Behnam Andarzian
Cristian Daniele
Erik Poll
Digital Security group
Radboud University Nijmegen

Fuzzing case studies

Group	Case study	Case study task
1	AudioConverter	
2	picojpeg	converts jpeg to tga
3	SVG2ASS, STL2OBJ, ImageToASCII	converts SVG vector image files into ASS files
4	Exiv2	processes metadata from images
5	FFmpeg	media player
6	Pixelsorter	sorts pixels
7	LibGD	image manipulation library
8	PDF Text Extraction	extracts text from PDF files
9	xpdf	PDF reader
10	GEGL	graphics library
11	GOCR	optical character recognition
12	Tifig	image converter
13	SoX	audio converter
17	Lepton	JPEG compression/decompression tool
20	mpv	media player
21	LibVips	image processing

Tools used

	Case study	Tool used
1	AudioConverter	AFL, zzuf
2	picojpeg	AFL++, Radamsa
3	SVG2ASS, STL2OBJ, ImageToASCII	AFL++, Radamsa, Honggfuzz
4	Exiv2	AFL++, Radamsa
5	FFmpeg	zzuf, AFL++, Honggfuzz
6	Pixelsorter	Zzuf, Ramdamsa, Honggfuzz, AFL++, AFL
7	LibGD	AFL++, Radamsa, Honggfuzz
8	PDF Text Extraction	AFL++ Honggfuzz
9	xpdf	AFL++, Radamsa, Honggfuzz
10	GEGL	AFL++, Radamsa, Honggfuzz
11	GOCR	AFL, Radamsa, Honggfuzz
12	Tifig	AFL++, Radamsa , Honggfuzz
13	SoX	AFL++, Radamsa, Honggfuzz
17	Lepton	zzuf, AFL++, Honggfuzz
20	mpv	zzuf, AFL++, Honggfuzz
21	LibVips	AFL++, Radamsa, Honggfuzz

Fuzzing results

	Case study	Tools that found crashes	Tools that didn't find bugs
1	AudioConverter	zzuf with TSan	Afl with and without ASan
2	picojpeg	AFL++ with or without ASan	Radamsa
3	SVG2ASS, STL2OBJ, ImageToASCII	AFL++, Honggfuzz and Radamsa	-
4	Exiv2	AFL++ with ASan, Radamsa with and without ASan	-
5	FFmpeg	AFL++ with Asan (no crashes but hangs)	zzuf and Honggfuzz
6	Pixelsorter	AFL/AFL++ (with and without ASan)	Radamsa, Honggfuzz, Zzuf
7	LibGD	AFL++, Radamsa, Honggfuzz with ASan	
8	PDF Text Extraction	Honggfuzz without ASan	AFL++
9	xpdf	AFL++, Radamsa and Honggfuzz with and without ASan	-
10	GEGL	AFL++ with and without ASan, Radamsa	Honggfuzz
11	GOCR	AFL++ with ASan, Honggfuzz only finds hangs	Radamsa
12	Tifig	Honggfuzz and AFL++ (with or without ASan & MSan?)	Radamsa
13	SoX	Honggfuzz and AFL++ with and without sanitizer, Radamsa with UBSan or MSan	-
17	Lepton	AFL++ (with or without ASan?), zzuf	Honggfuzz
20	mpv	AFL++ with FRIDA, Hongfuzz only finds hangs	zzuf
21	LibVips	AFL++ with and without ASan, Honggfuzz	Radamsa

Misc. issues

- afl unique != unique
- afl slow or zzzzz….
- Are hangs really hangs?
- *Who actually reporting bugs found?*
- *Good/bad experiences using cloud?*

Unexpected findings

- Group 8 (PDF text extraction): AFL++ with ASan could not find bugs but Honggfuzz without ASan could

- Group 9 (xpdf): AFL++ and Radamsa without ASan found more bugs than AFL++ with ASan

Group 8 – PDF text extraction

Fuzzer Tool	Input file	Fuzzer Output			
		Number of executions	Number of crashes	Unique Crashes	Time ran
afl	minimal.pdf	264k	0	0	1h50min
afl	tiny.pdf	273k	0	0	1h53min
afl+ASAN	minimal.pdf, tiny.pdf, weirdCharsFonts.pdf, pdf_just_jpg.pdf	391k	0		55min
afl+ASAN	minimal.pdf, tiny.pdf, weirdCharsFonts.pdf, pdf_just_jpg.pdf	436k	0		55min
Hongg	minimal.pdf	527k	2156	20	4h17
Hongg	test{1-5}.pdf	1M	817	23	7h55
Hongg	pdf_just_jpg.pdf	917k	5813	14	9h11

- Unexpected: AFL++ with ASan could not find bugs but Honggfuzz without ASan could
  - but code was left uninstrumented using AFL++??
- Honggfuzz could not be made to work with ASan

Group 9 - xpdf

Number	Tool	Time	# test cases	Issues found	crashes/mill. TC	crashes/hour
Experiment 1	AFL++	3h 45m (15m/core)	36270	117 crashes. 3 hangs	3226	31
Experiment 2	AFL++ with ASan	3h 45m (15m/core)	32393	31 crashes. 2 hangs	957	8
Experiment 3	HonggFuzz	7h 45m (116m/core)	50945	19 crashes. 2 hangs	373	3
Experiment 4	HonggFuzz with instrumentation	37h 45m (1132m/core)		0 crashes. 12 timeouts	0	0
Experiment 5	Radamsa	20m	68897	20 crashes	290	60
Experiment 6	Radamsa with ASan	30m	50630	10 crashes	197	20

- **Unexpected: AFL++ and Radamsa without ASan found more bugs than AFL++ with ASan**
  - **but: Ramamsa commonly reports the same bug many times**

Group 2 - picojpeg

Fuzzer	Sanitizer	Seed	Duration	# executions	Issues found
afl++	none	minimal JPEG	~ 14h	~ 100.6M	26 crashes & 7 hangs
	ASan			~ 52.8M	80 crashes & 10 hangs
	MSan			~ 1.9M	25 crashes & 6 hangs
	none	Radboud Logo		~ 65.8M	62 crashes & 15 hangs
	ASan			~ 26.9M	115 crashes & 15 hangs
	MSan			~ 1.7M	24 crashes & 8 hangs

- **Minimal JPEG (107 bytes) vs Radboud logo (40kB) makes little difference**
- **One bug because of malloc returning null**
- **Still open question if hangs were bugs or not**

Group 2 – picojpeg – root cause analysis

Fuzzer	Sanitizer	Seed	Duration	# executions	Issues found
afl++	none	minimal JPEG	~ 14h	~ 100.6M	26 crashes & 7 hangs
	ASan			~ 52.8M	80 crashes & 10 hangs
	MSan			~ 1.9M	25 crashes & 6 hangs
	none	Radboud Logo		~ 65.8M	62 crashes & 15 hangs
	ASan			~ 26.9M	115 crashes & 15 hangs
	MSan			~ 1.7M	24 crashes & 8 hangs

We checked how many different stack traces were produced by the crashes. That way, we could reduce the number of problems a lot. The address sanitizer found two variants of a stack buffer overflow and one variant of each a global buffer overflow, heap buffer overflow, segmentation fault, and the program requesting more memory than available, i.e. six different problems. The memory sanitizer showed two segmentation faults, two uses of uninitialized values, and one memory allocation that exceeds the available memory, i.e. five different problems.

Using the stack traces, we were able to spot nine different places in the source code of the SUT that produced the errors. Two crashes had insufficient backlogs to spot the failing line. Due to time constraints and the complexity of the JPEG format and thus the resulting code, we were not able to spot the root causes. One error could have been mitigated by checking the result of `malloc` before using it. The `malloc` was returning `NULL` in this specific case because the requested size was many orders of magnitudes bigger than the available memory.

Group 3 – Svg2ass

Tool	Workers	Time (total)	Test cases	Mutations	Hangs	Crashes
AFL++ (initial test)	1	48m	1		0	3
AFL++ (server, various flags)	4	22d 21h	1	550.078.548	136	296
Honggfuzz	8	13h 40m	4	83.046.724	2	2587
Radamsa	1	2h 54m	1	1.056.395	0	77
AFL++ with ASan	1	1d 2h	1	51.586.646	15	70
Radamsa with ASan	1	3h 12m	1	375.509	0	278

- **ASan slows things down by factor (but it usually worth it?)**
- **Every crash attributed to the same error:**

```
double free or corruption (!prev)
Aborted
```

One interesting observation was that 292 of these crashes were found within the first day of running the fuzzers. It seems that after a day, most inputs that it tried had

that, ASan reported over 368,000 memory leaks in the SVG2ASS application, which we would not have found using the regular fuzzing setup without any sanitization.

Group 4 – exiv2

- Speed-up from 20 to 1000-2000 execs/sec by using persistent mode
- As expected, no bugs in most recent version as project uses fuzzer now

Version	Instrumentation	Inputs	Deterministic	Time	Execs	Crashes
v0.27.4-RC2	ASAN+UBSAN	JPEG	No	01:58:35	6.41M	48
			Yes	01:33:14	4.61M	22
		PNG	No	02:07:05	8.93M	36
			Yes	03:40:17	9.35M	0
v0.27.5	ASAN+UBSAN	JPEG	No	03:11:41	9.89M	0
			Yes	03:11:16	6.22M	0
		PNG	No	02:12:56	10.2M	0
			Yes	08:32:30	33.9M	0

Table 1: AFL++ run on two versions of exiv2

Group 12 - Tifig

Number	Tool	Time	No of initial files	No of test cases	Issues found	Crashes/hour
Tifig	AFL++	1hr 1min	1	36822	34	26,36
Tifig	Radamsa**	Inconclusive	1	Inconclusive	Inconclusive	Inconclusive
Tifig	Honggfuzz	5 min	3	1,481	329	3948
Tiv	AFL++	1hr 13min	3	305972	0	0
Tiv	Radamsa**	Inconclusive	1	Inconclusive	Inconclusive	Inconclusive
myHTML	AFL++	3hr 22min	1	8383972	0	0

- **Many (all?) 'unique ' crashes found by Honggfuzz were the same bug**

Although all of the tools helped and gave us some nice insights into what caused the crashes. We found that ASan returned the easiest to understand reasons for the crashes. We also concluded that valgrind, was a great tool but it is very slow.

- **Root cause analysis:**
  - **1 heap buffer overflow,**
  - **1 SEGV on unknown address (found using MSan rather than ASan)**
  - **1 heap use-after-free?**

  Some bugs already submitted, but project abandonded

Group 3 – svg2ass – root cause analysis

- inspected using GDB
- error in free() statement
- problem in parsing of (malformed) XML structure, with no tag behind <

```
239     case ST_CONTENT:
240         m = strchr( p, '<' );
241         if ( m )
242             *m++ = '\0';
```

  - This probably qualifies as shotgun parsing?

- *Was this is the latest release of svg2ass?*
- *If so, did you report the bug / the fix?*

Group 4 – exiv – older versions using Radamsa

Version	Instrumentation	Number	Inputs	Time	No of mutations[1]	Crashes
v0.16	None	testOther	Other	00:15:04	97 * 100	29
		testJPEG	JPEG	00:00:08	4 * 100	0
		testPNG	PNG	00:00:08	4 * 100	0
		testTYPES	Types	00:00:21	13 * 100	0
		runOther1	Other	02:33:40	97 * 1000	212
		runJPEG1	JPEG	00:13:19	4 * 10000	1
		runPNG1	PNG	00:13:02	4 * 1000	0
		runTypes1	Types	00:36:35	13 * 10000	173
v0.20	None	runOther2	Other	00:19:14	97 * 100	1
		runJPEG2	JPEG	00:13:57	4 * 10000	0
		runPNG2	PNG	00:14:35	4 * 10000	0
		runTypes2	Types	00:37:40	13 * 10000	0
v0.27.4-RC2	None	runOther3	Other	02:22:58	97 * 1000	0
		runJPEG3	JPEG	00:11:22	4 * 10000	0
		runPNG3	PNG	00:09:06	4 * 10000	0
		runTypes3	Types	00:22:55	13 * 10000	0

Group 4 – exiv – latest versions using Radamsa

v0.27.5	None	runOther4	Other	00:13:56	97 * 100	0
		runJPEG4	JPEG	00:08:25	4 * 10000	0
		runPNG4	PNG	00:07:44	4 * 10000	0
		runTypes4	Types	00:20:10	13 * 10000	0
	ASAN + UBSAN	instrOther	Other	00:42:07	97 * 100	3
		instrJPEG	JPEG	01:19:25	4 * 10000	0
		instrPNG	PNG	01:19:42	4 * 10000	0
		instrTypes	Types	04:07:13	13 * 10000	0

Even in the most recent version, Radamsa managed to find some crashes that AFL++ did not manage to find.

It is also quite notable how, in the most recent versions, instrumentation is basically required in order to find crashes for Exiv2, while this was not necessary for older versions. This shows that Exiv2 definitely improved security with regards to memory safety over the years.

Does fuzzing find different bugs for different operations?

For the fuzzing of LibGD we used many different input formats, such as BMP, PNG, GIF, TGA. We ran with this input format for many hours and performed a few different operations on these images, such as rotating; scaling; negation and Gaussian blur. And with these operations we provided various arguments to the functions of the library.

We mentioned that we were curious whether using LibGD's transformations in our experiments would affect their outcome. We found this not to be the case for the experiments that we performed.

group 21 had different experience

Experiment number	Libvips module	Initial seed	Tools used	Duration in minutes	No. of test cases	Test/min	Issues found
19	rot	blank.jpeg	AFL++ & ASan	46	193.000	4.195	None
20	gamma	blank.jpeg	AFL++ & ASan	109	351.000	3.220	None
21	affine	blank.jpeg	AFL++ & ASan	170	1.130.000	6.647	3 crashes, 26 hangs
22	draw_rect	blank.jpeg	AFL++ & ASan	241	1.700.000	7.053	115 hangs

Group 13 - Sox

No.	Tool	Sanitizer	Time	Test cases	Crashes	Timeouts	Crashes/million	Cases/hour
1	AFL++	None	12 hrs	59 245 651	32	3	0.54	4 937 137
2	AFL++	ASAN	12 hrs	19 772 819	30	6	1.52	1 647 734
3	AFL++	MSAN	12 hrs	2 779 440	26	11	9.35	231 620
4	AFL++	UBSAN	12 hrs	139 110 621	75	3	0.54	11 592 551
5	AFL++	CFISAN	12 hrs	55 350 939	24	4	0.43	4 612 578
6	Radamsa	None	12 hrs	7 583 000	0	0	0	631 916
7	Radamsa	ASAN	12 hrs	4 359 500	0	0	0	363 291
8	Radamsa	MSAN	12 hrs	5 062 000	1	0	0.71	421 833
9	Radamsa	UBSAN	12 hrs	5 786 500	896047	0	567 837.14	482 208
10	Radamsa	CFISAN	12 hrs	7 579 500	0	0	0	631 625
11	HonggFuzz	None	18 hrs	1 202 474	0	451	0	66 804
12	HonggFuzz	ASAN	18 hrs	1 094 811	277	195	46.58	60 822
13	HonggFuzz	MSAN	18 hrs	1 088 257	51	130	0	60 458
14	HonggFuzz	UBSAN	18 hrs	1 063 982	0	14	0	59 110
15	HonggFuzz	CFISAN	18 hrs	1 206 133	0	484	0	67 007

- **Radamsa finds many more duplicates**
- ***Was this the latest version & did you report the bugs?***

Group 13 – SoX – root place & cause analysis

Source file	Sanitizer	Issue Type	Line number	Column number
src/wav.c	MSAN	use-of-uninitialized-value	508	11
src/wav.c	MSAN	FPE	848	80
src/wav.c	MSAN	FPE	1353	34
src/wav.c	MSAN	use-of-uninitialized-value	527	5
src/wav.c	MSAN	use-of-uninitialized-value	773	5
src/wav.c	MSAN	use-of-uninitialized-value	518	11
src/xa.c	ASAN	SEGV	219	26
src/wav.c	ASAN	FPE	1353	34
src/wav.c	ASAN	FPE	848	80
src/adpcm.c	ASAN	heap-buffer-overflow	154	13

Running AFL++ with MSAN and ASAN two same issues were found (see Table 2). Comparing AFL++ to Radamsa no similar issues were found even if issues were already found in the same source file.

Group 17 - Lepton

Number	Tool	Time	No of test cases	Issues found
1	zzuf compression	15 min	50,000 .jpg	49,901 crashes, 0 hangs
2	zzuf decompression	25 min	100,000 .lep	99,953 crashes, 0 hangs
3	zzuf compression with Asan	33 min	50,000 .jpg	49,812 crashes, 0 hangs
4	zzuf decompression with Asan	1 hour 11 min	100,000 .lep	99,932 crashes, 0 hangs
5	AFL++ instrumented compression	2 hours	619,000	43 crashes, 1 hang
6	AFL++ non-instrumented compression	31 min	500,000	6767 crashes, 0 hangs
7	AFL++ instrumented decompression	2 hours 24 min	321,000	5 crashes, 19 hangs
8	AFL++ non-instrumented decompression	2 hours 21 min	1,960,000	0 crashes, 0 hangs
9	Hongfuzz compression	1.5 hours	1,820,000 .jpg	0 crashes, 0 hangs
10	Hongfuzz decompression	1 hour	2,330.000 .lep	0 crashes, 0 hangs

But crashes are gentle crashes, ie. crash with error messages. No memory corruption bugs found

Were hangs found new & in the latest version? Did you report them?

Group 17 - Lepton

Both zzuf and AFL++ ended up finding a lot of gentle crashes, but AFL++ was the only fuzzer which ended up finding files on which Lepton hangs indefinitely. These hangs are much more impactful than gentle crashes for Lepton when used as a component for Dropbox, so we think AFL++ has the best results, even though zzuf finds more gentle crashes.

Different next year?

- Brightspace forum & class discussion of potential targets
  - to steer clear off uninteresting targets
    - *Suggestions of projects to avoid? Eg mplayer, mpv, FFmpeg? Others?*

- Demo & discussion session early November

- *Other suggestions?*