WEIZZ: Automatic Grey-Box Fuzzing for Structured Binary Formats

Andrea Fioraldi, Daniele Cono D’Elia and Emilio Coppa

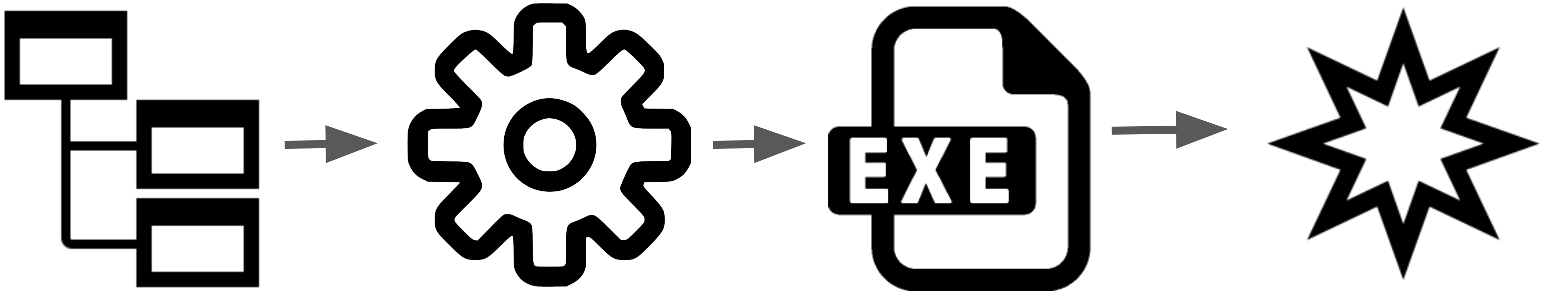
@andreafioraldi

andreafioraldi@gmail.com

SAPIENZA
UNIVERSITÀ DI ROMA

ISSTA
LAX 2020

Format-aware Fuzzing



Input Format Model

Input Generation

Program Under Test

Crashes

Format-aware Fuzzing

- LangFuzz
- Peach
- Spike
- CSmith
- ...

Problems

- Impossible if the input structure is unknown

Problems

- Impossible if the input structure is unknown
- May fail to find bugs related to syntactically invalid inputs in parsers

Problems

- Impossible if the input structure is unknown
- May fail to find bugs related to syntactically invalid inputs in parsers
- Parser implementations do not always closely mirror format specifications

Problems

- Impossible if the input structure is unknown
- May fail to find bugs related to syntactically invalid inputs in parsers
- Parser implementations do not always closely mirror format specifications
- Models take some time to be written by a human (and contains simplifications)

Problems

- Impossible if the input structure is unknown
- May fail to find bugs related to syntactically invalid inputs in parsers
- Parser implementations do not always closely mirror format specifications
- Models take some time to be written by a human (and contain simplifications)
- Wrong models make fuzzing ineffective

Solutions?

- Automatically learn the model from the actual implementation of the parser

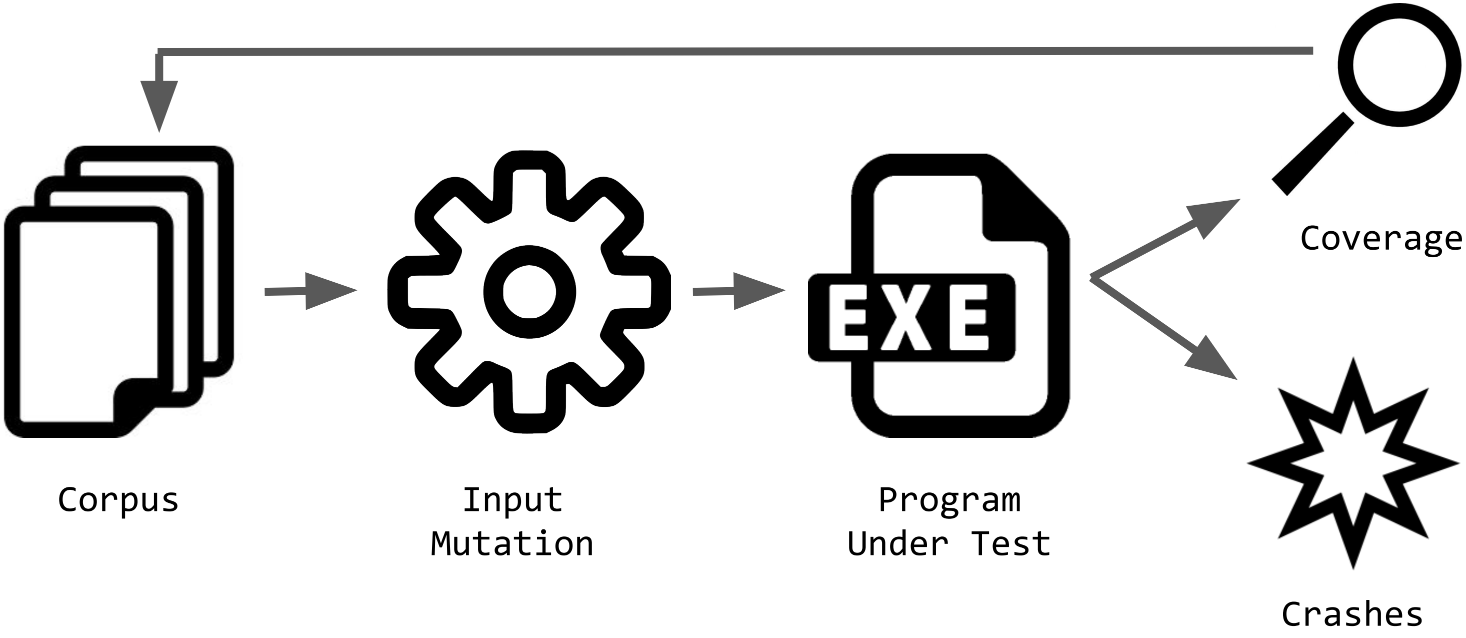
Solutions?

- Automatically learn the model from the actual implementation of the parser
- Generate not always syntactically valid inputs

Solutions?

- Automatically learn the model from the actual implementation of the parser
  - (Approximation of) Taint Tracking
    - [Tupni] [Autogram] [Polyglot] [Grimoire]
  - Machine Learning
    - [Learn&Fuzz] [REINAM]
  - Oracle based
    - [GLADE]
- Generate not always syntactically valid inputs

Coverage-guided Fuzzing



Problems

- Fail to explore deep paths behind parsers

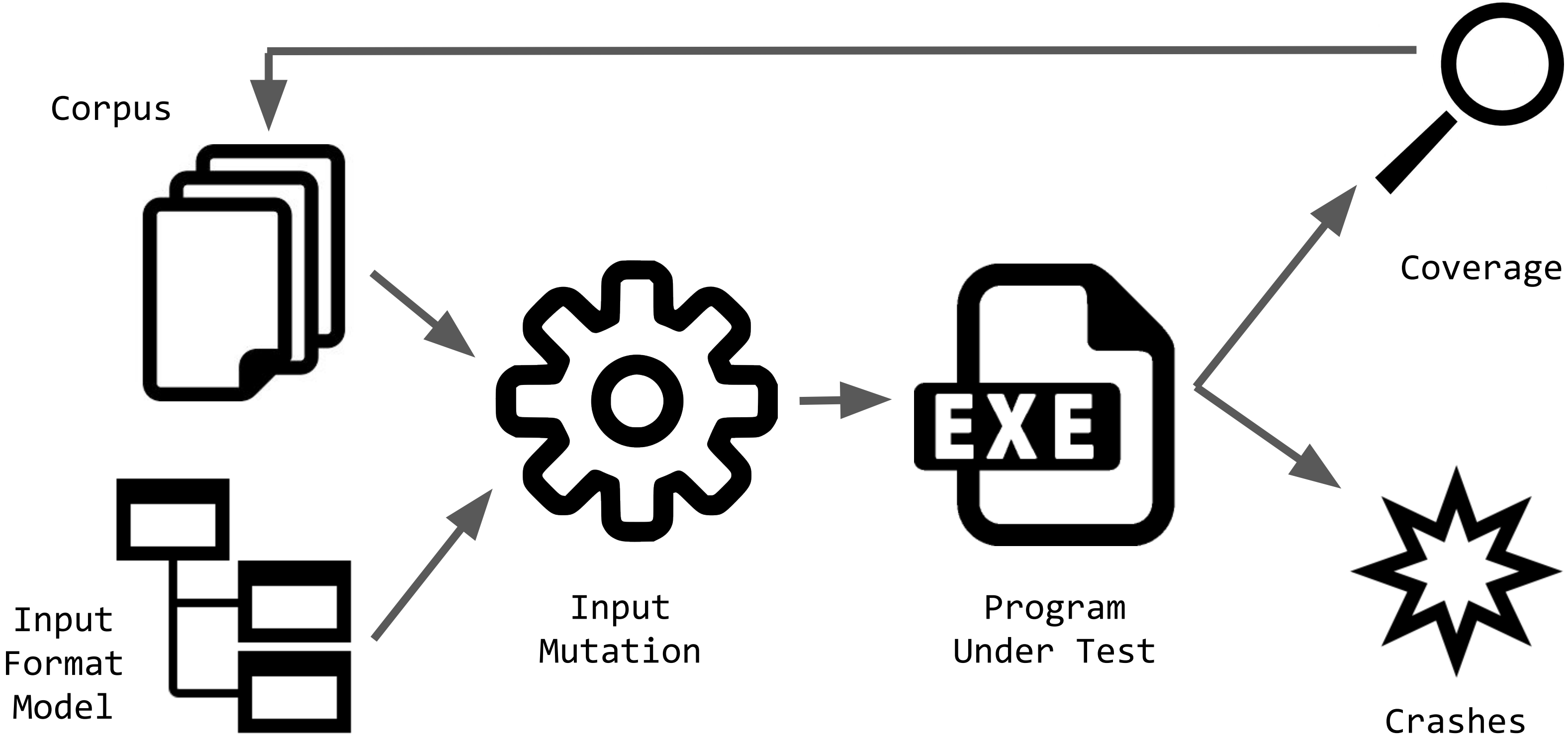
Problems

- Fail to explore deep paths behind parsers

- Affected by roadblocks (multi-byte comparisons, checksums, hashes, …)

```
if (hash(input[0:8]) != input[8:12]) exit(1)

if (input[12:16] == 0xABADCAFE) bug()
```

Structured Fuzzing



Structured Fuzzing

- AFLSmart
- Nautilus
- Superion
- Libprotobuf-Mutator
- Zest
- ...

Bypass Roadblocks

- Concolic Fuzzing
    - [Driller] [QSYM] [Eclipser]
- (Approximation of) Taint Tracking
    - [TaintScope] [Vuzzer] [Angora] [Redqueen]
- Sensitive feedbacks
    - [LAF-Intel] [CompareCoverage] [FuzzFactory] [IJON]

Bypass Roadblocks

- Concolic Fuzzing
  - [Driller] [QSYM] [Eclipser]
- (Approximation of) Taint Tracking
  - [TaintScope] [Vuzzer] [Angora] [Redqueen]
- Sensitive feedbacks
  - [LAF-Intel] [CompareCoverage] [FuzzFactory] [IJON]

Idea #1

- Reuse expensive analysis to bypass roadblocks previously explored in past works to enable Structure-aware mutations

Bypass Roadblocks [Redqueen]

- Mutations targeting magic byte comparisons (Input-To-State)

Bypass Roadblocks [Redqueen]

- Mutations targeting magic byte comparisons (Input-To-State)

```
input: AAAABBBBCCCCBBBB

cmp eax, FFFF → eax = BBBB
```

Bypass Roadblocks [Redqueen]

- Mutations targeting magic byte comparisons (Input-To-State)

  input: AAAABBBBDDCCDDCC (equivalent in coverage)

  cmp eax, FFFF → eax = BBBB

Bypass Roadblocks [Redqueen]

- Mutations targeting magic byte comparisons (Input-To-State)

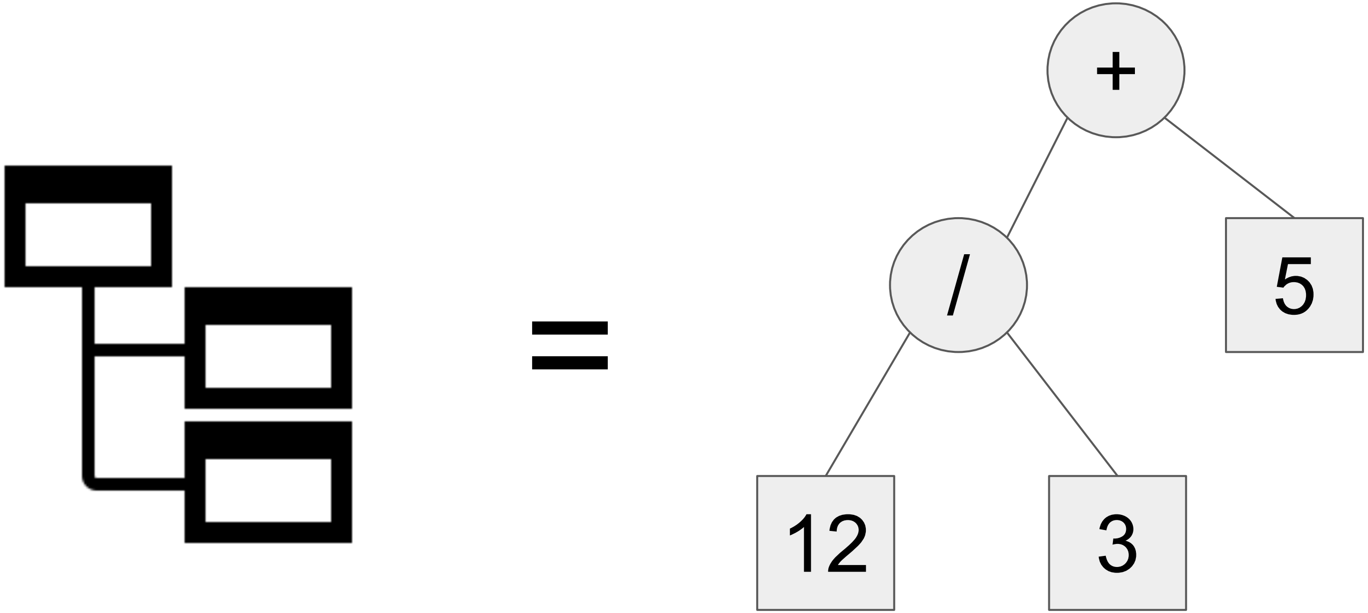
```
input: AAAABBBBDDCCDDCC (equivalent in coverage)

cmp eax, FFFF → eax = BBBB

new input: AAAAFFFFDDCCDDCC
```

Bypass Roadblocks [Redqueen]

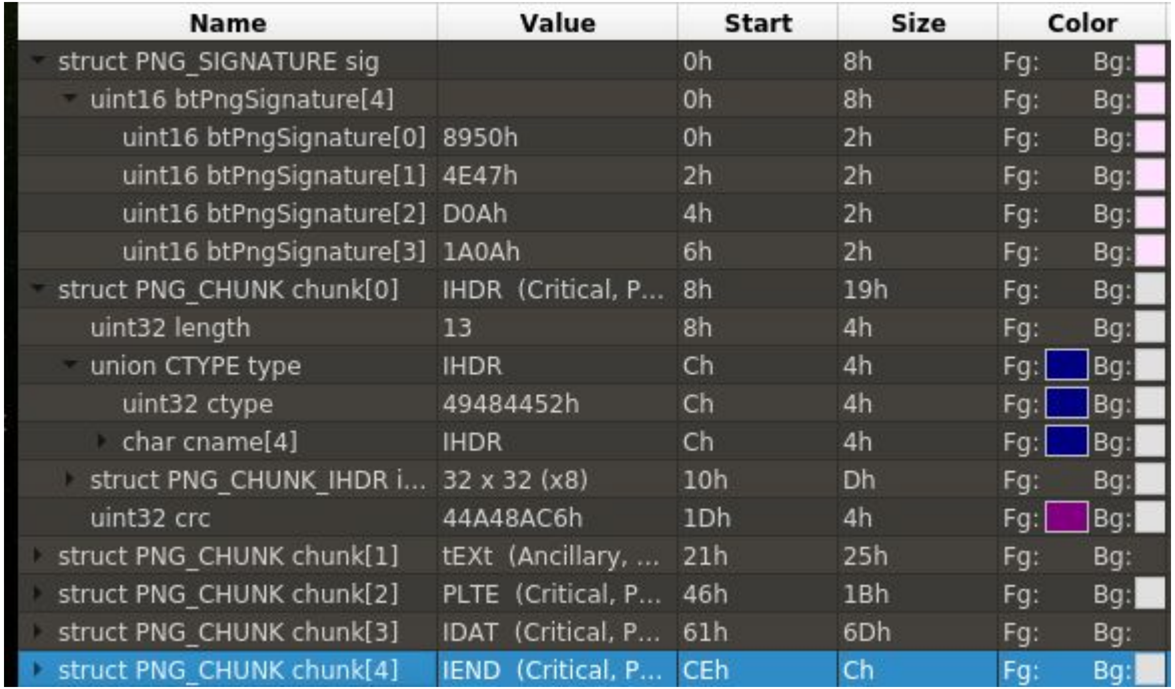
- Mutations targeting magic byte comparisons (Input-To-State)

- Patch out checksum checks

Formats as an AST [Grimoire]



Not all formats are parsed into an AST



Name	Value	Start	Size	Color
struct PNG_SIGNATURE sig		0h	8h	Fg: Bg: □
uint16 btPngSignature[4]		0h	8h	Fg: Bg: □
uint16 btPngSignature[0]	8950h	0h	2h	Fg: Bg: □
uint16 btPngSignature[1]	4E47h	2h	2h	Fg: Bg: □
uint16 btPngSignature[2]	D0Ah	4h	2h	Fg: Bg: □
uint16 btPngSignature[3]	1A0Ah	6h	2h	Fg: Bg: □
struct PNG_CHUNK chunk[0]	IHDR (Critical, P...	8h	19h	Fg: Bg: □
uint32 length	13	8h	4h	Fg: Bg: □
union CTYPE type	IHDR	Ch	4h	Fg: 🟦 Bg: □
uint32 ctype	49484452h	Ch	4h	Fg: 🟦 Bg: □
char cname[4]	IHDR	Ch	4h	Fg: 🟦 Bg: □
struct PNG_CHUNK_IHDR i...	32 x 32 (x8)	10h	Dh	Fg: Bg: □
uint32 crc	44A48AC6h	1Dh	4h	Fg: 🟪 Bg: □
struct PNG_CHUNK chunk[1]	tEXt (Ancillary, ...	21h	25h	Fg: Bg:
struct PNG_CHUNK chunk[2]	PLTE (Critical, P...	46h	1Bh	Fg: Bg: □
struct PNG_CHUNK chunk[3]	IDAT (Critical, P...	61h	6Dh	Fg: Bg:
struct PNG_CHUNK chunk[4]	IEND (Critical, P...	CEh	Ch	Fg: Bg: □

Comparisons for validation

```
if (chunk->size_field > SIZE_MAX)
    error(“Invalid Chunk Size”);
```

Idea #2

- Instead of using memory accesses to reconstruct the format ([Tupni] [Autogram]) use the comparisons instructions that are likely validation checks

Idea #3

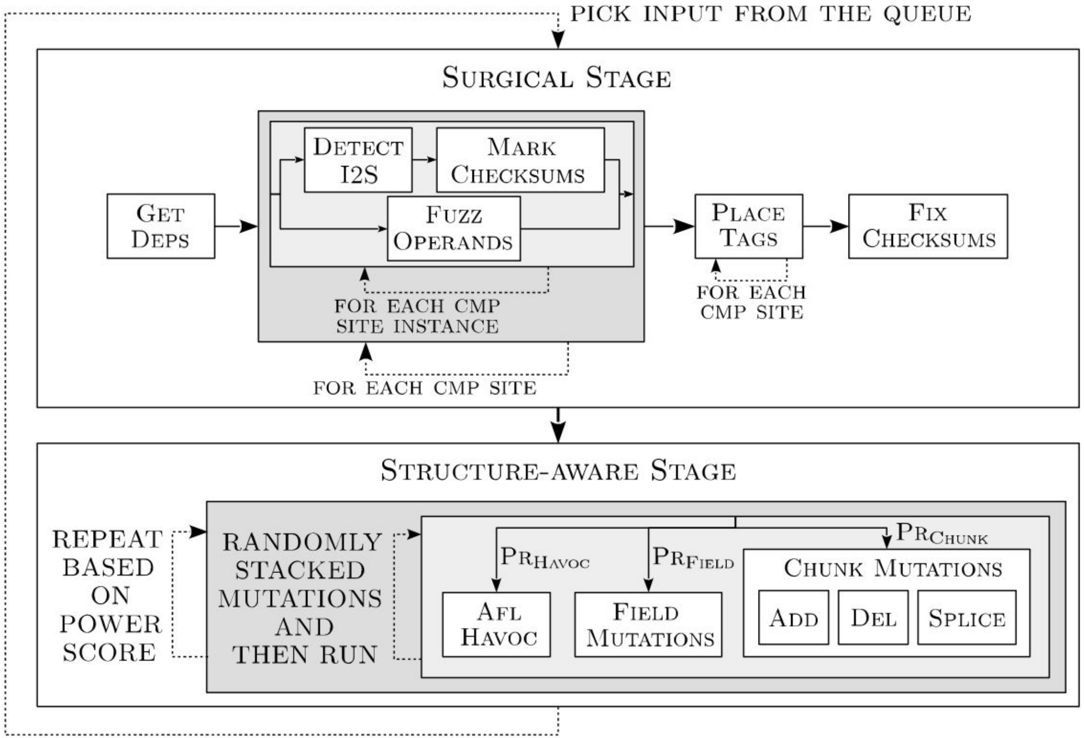
- Don’t learn a model and use it to guide the fuzzer, but reconstruct each time the structure and apply mutations.

  This avoids the problem of having errors in the learning process.

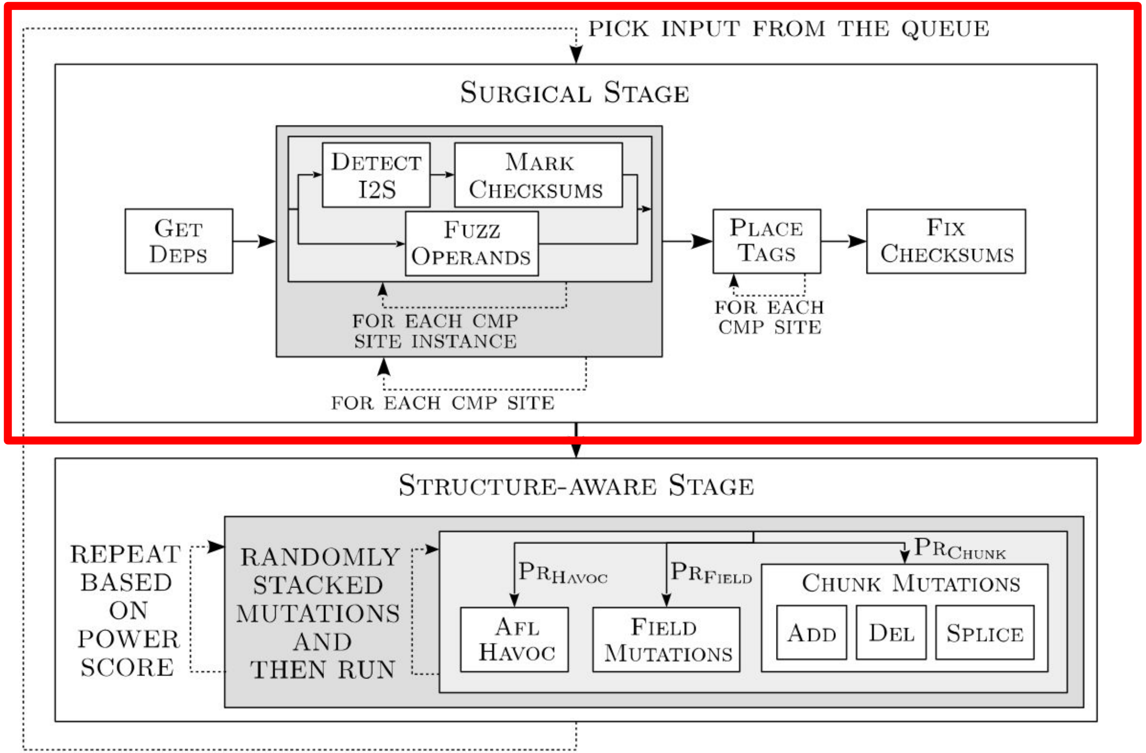
Weizz

- Based on AFL 2.52b
- Binary-only (QEMU)
- Approximate Taint to bypass Roadblocks and learn information about validation checks
- Structural mutations based on that information (inspired by [AFLSmart])

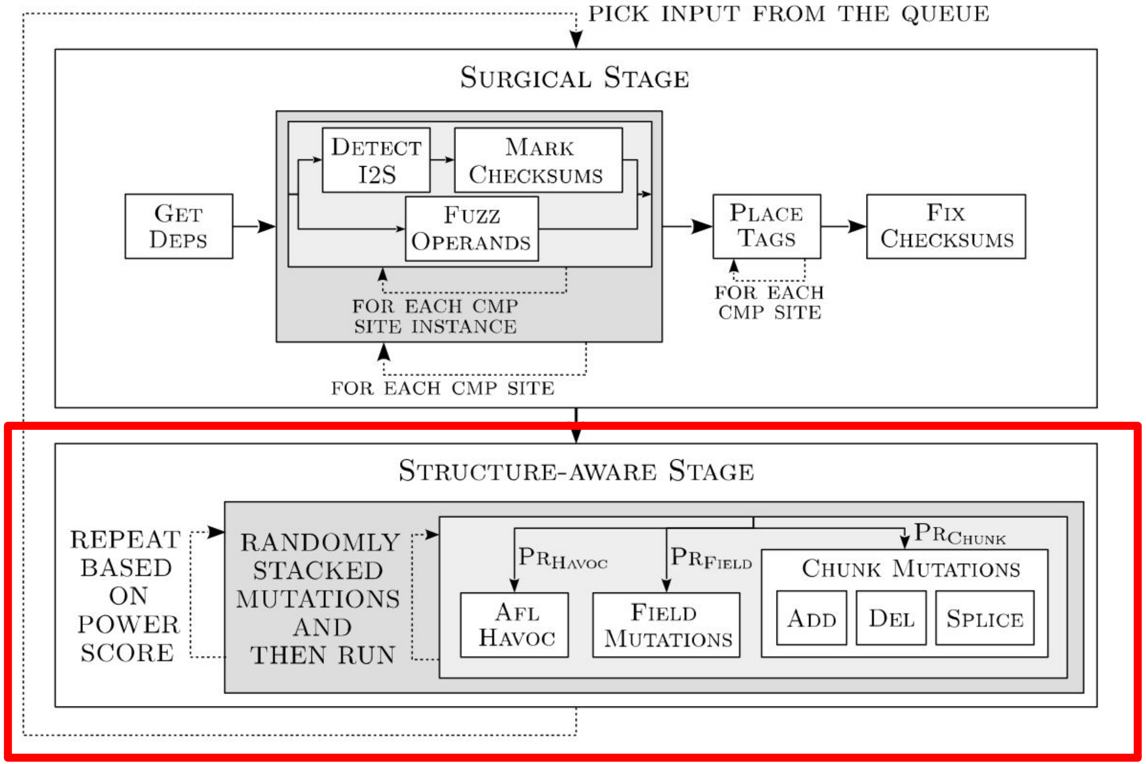
Architecture



Architecture



Architecture



GetDeps: Approximating Taint Tracking

Input: AAAABBBBCCCCDDDD

```
cmp eax, FFFF → eax = AAAA
```

GetDeps: Approximating Taint Tracking

```
Input: AAAABBBBCCCCDDDD
```

```
cmp eax, FFFF → eax = AAAA
```

```
Bitflip #1: BAAABBBBCCCCDDDD
```

```
cmp eax, FFFF → eax = BAAA
```

Detect Checksum Checks

- One operand is I2S
- The other operand is not I2S and GetDeps revealed dependencies on some input bytes
- The sets of their byte dependencies are disjoint

Input Tags

- Comparison ID
- Timestamp
- Parent ID
- Number of tags with the same ID
- The Comparison ID of the inner checksum that guard this byte
- Flags (which CMP operand, if this is a checksum field, …)

Many Comparisons affected by the same byte

1. Prioritize Checksum fields

Many Comparisons affected by the same byte

1. Prioritize Checksum fields
2. Prioritize comparisons appeared earlier in time (possible validation checks)

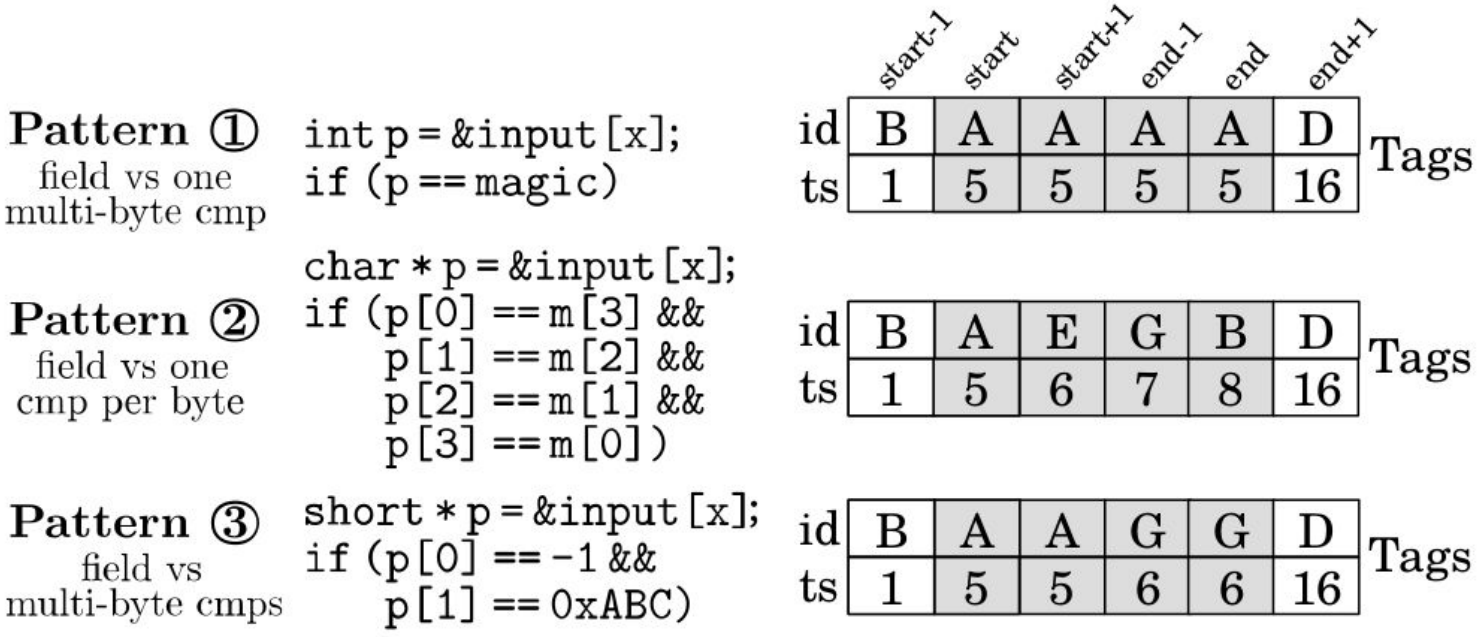
Many Comparisons affected by the same byte

1. Prioritize Checksum fields
2. Prioritize comparisons appeared earlier in time (possible validation checks)
3. Prioritize if the number of bytes influencing the comparison are low

Fixing Checksum

- Late-stage repair
- Topological Sort (Tags have the info for this)
- Unpatch false positives

Locating Fields



Pattern ①
field vs one
multi-byte cmp

```
int p = &input [x];
if (p == magic)
```

	start-1	start	start+1	end-1	end	end+1	
id	B	A	A	A	A	D	Tags
ts	1	5	5	5	5	16	

Pattern ②
field vs one
cmp per byte

```
char * p = &input [x];
if (p[0] == m[3] &&
     p[1] == m[2] &&
     p[2] == m[1] &&
     p[3] == m[0])
```

id	B	A	E	G	B	D	Tags
ts	1	5	6	7	8	16	

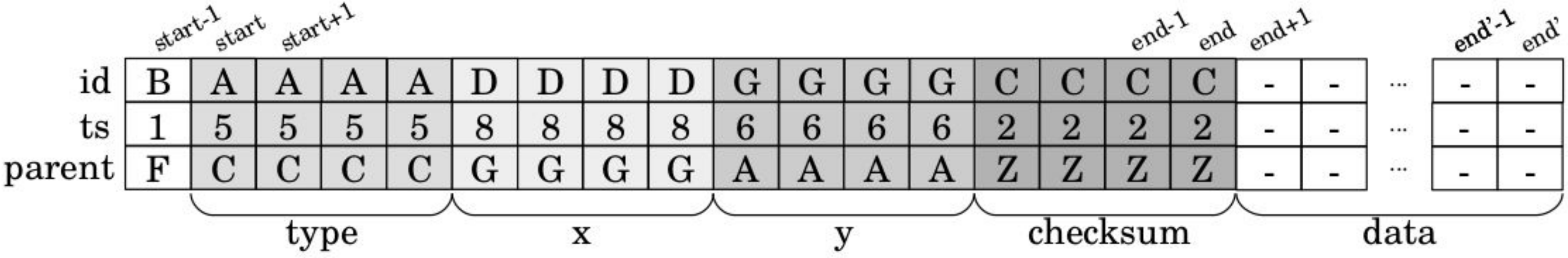
Pattern ③
field vs
multi-byte cmps

```
short * p = &input [x];
if (p[0] == -1 &&
     p[1] == 0xABC)
```

id	B	A	A	G	G	D	Tags
ts	1	5	5	6	6	16	

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

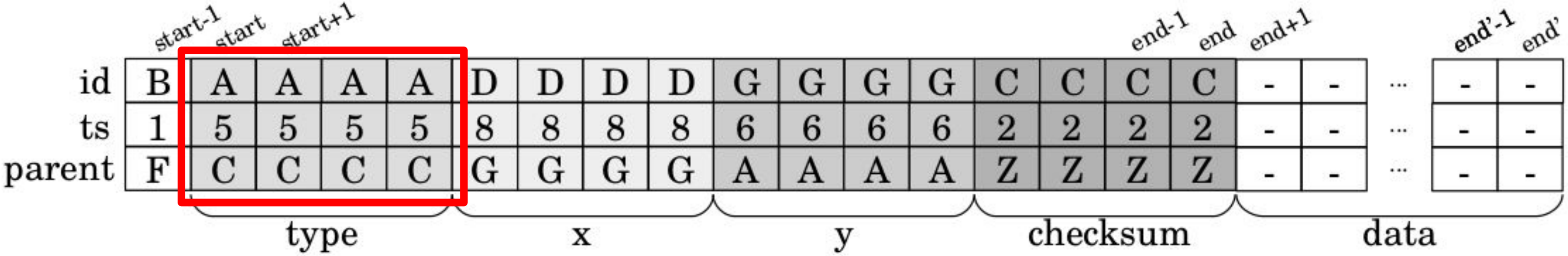


	start-1	start	start+1												end-1	end	end+1			end'-1	end'
id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

1. Pick a tag type

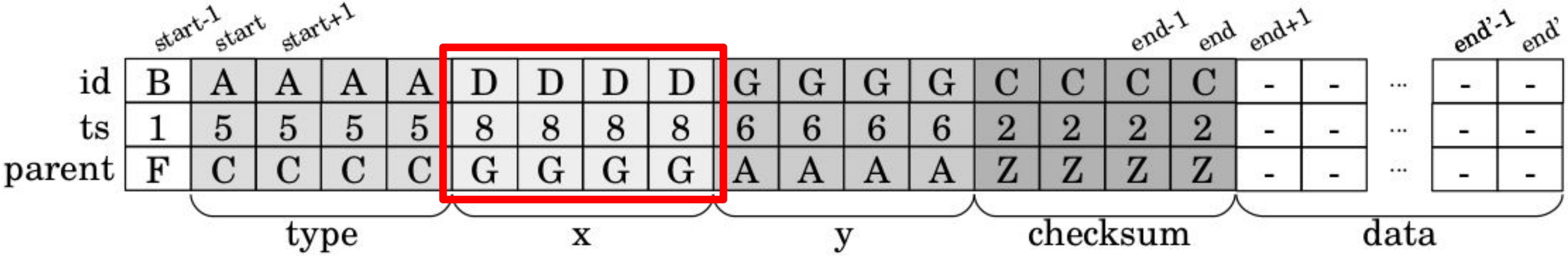


	start-1	start	start+1													end-1	end	end+1			end'-1	end'
id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

1. Pick a tag type
2. Recurse if next Timestamp (ts) > current

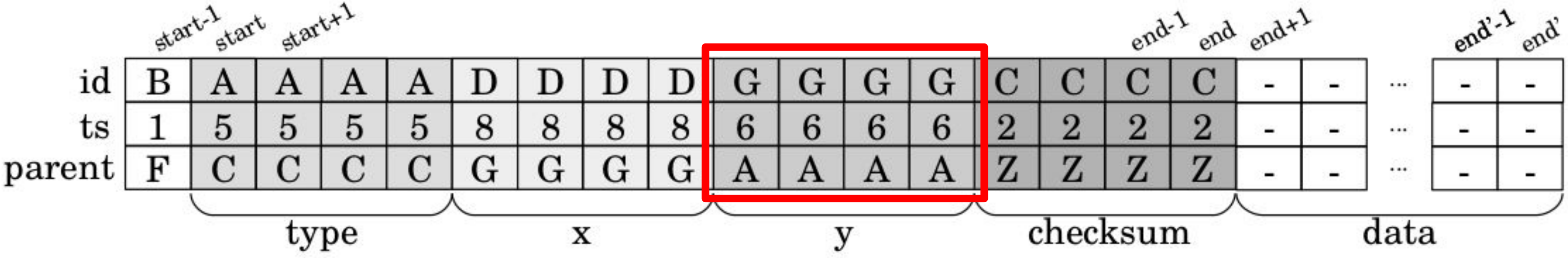


id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

1. Pick a tag type
2. Recurse if next Timestamp (ts) > current

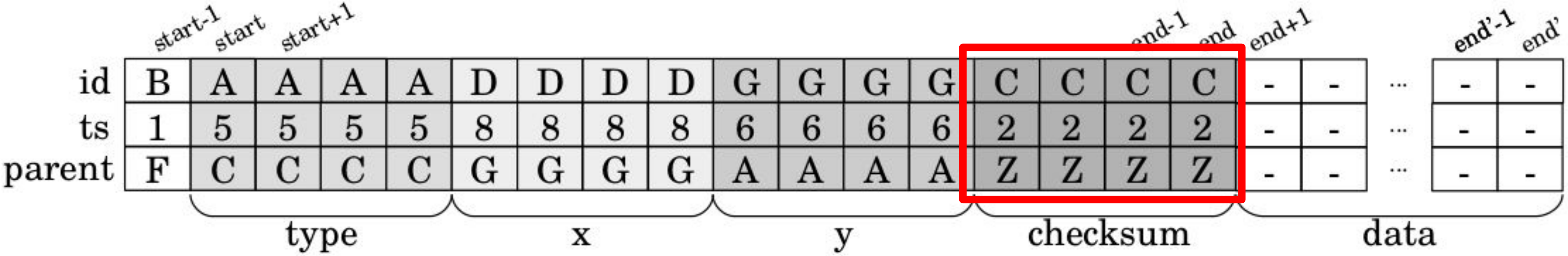


	start-1	start	start+1													end-1	end	end+1			end'-1	end'
id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-
		type				x				y				checksum				data				

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

1. Pick a tag type
2. Recurse if next Timestamp (ts) > current
3. Go forward if next ID = current Parent

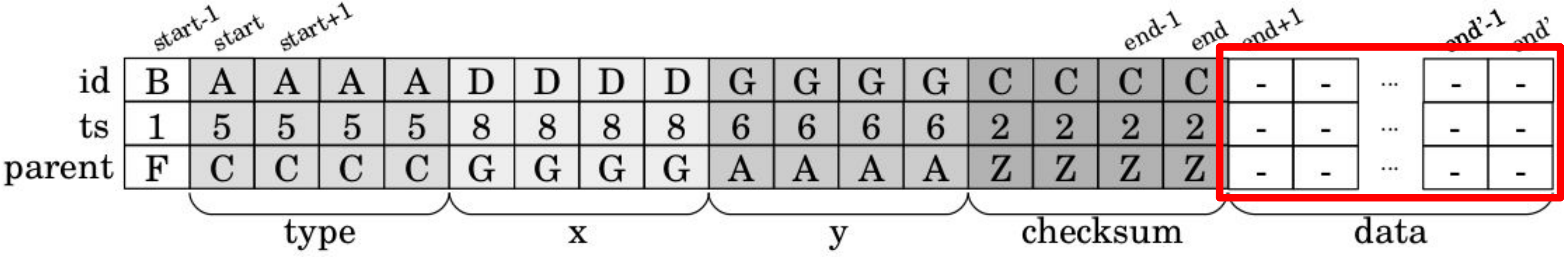


	start-1	start	start+1												end-1	end	end+1				end'-1	end'
id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-
		type				x				y				checksum				data				

Locating Chunks

```
struct {
  int type;
  int x , y;
  int cksm;
};
```

1. Pick a tag type
2. Recurse if next Timestamp (ts) > current
3. Go forward if next ID = current Parent
4. With a probability take untagged part and recurse again



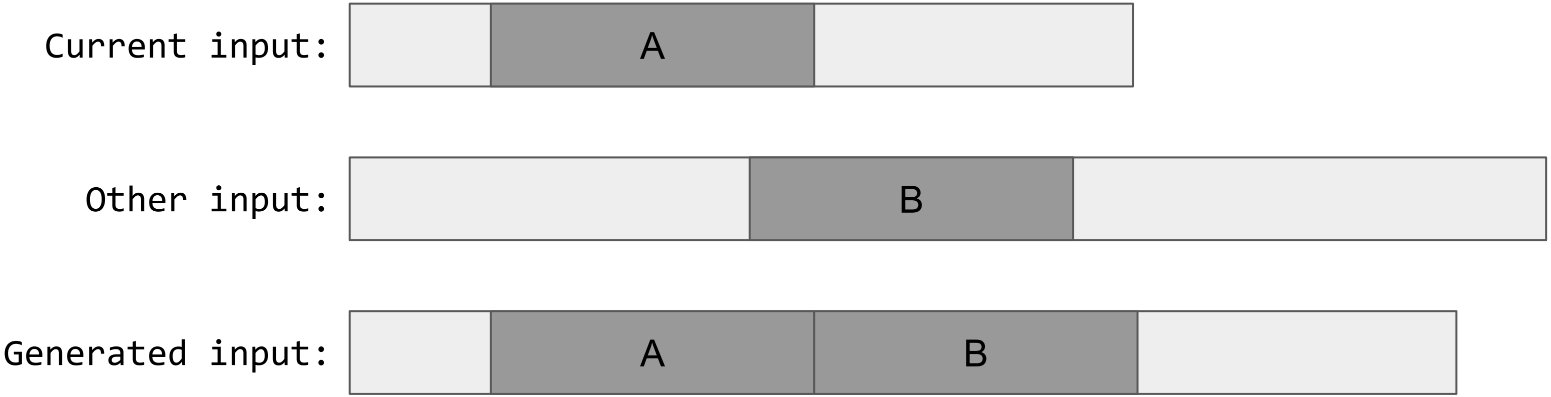
	start-1	start	start+1													end-1	end	end+1			end'-1	end'
id	B	A	A	A	A	D	D	D	D	G	G	G	G	C	C	C	C	-	-	...	-	-
ts	1	5	5	5	5	8	8	8	8	6	6	6	6	2	2	2	2	-	-	...	-	-
parent	F	C	C	C	C	G	G	G	G	A	A	A	A	Z	Z	Z	Z	-	-	...	-	-

Mutating Chunks [AFLSmart]

- Addition
- Deletion
- Splicing

Mutating Chunks [Weizz]

- Addition
    - Select a chunk A and adds a chunk from another input in the queue with the same parent ID in the first tag of A before or after A



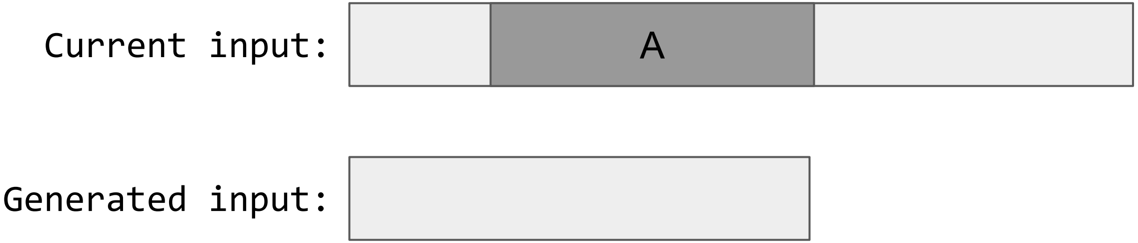
Current input: A

Other input: B

Generated input: A B

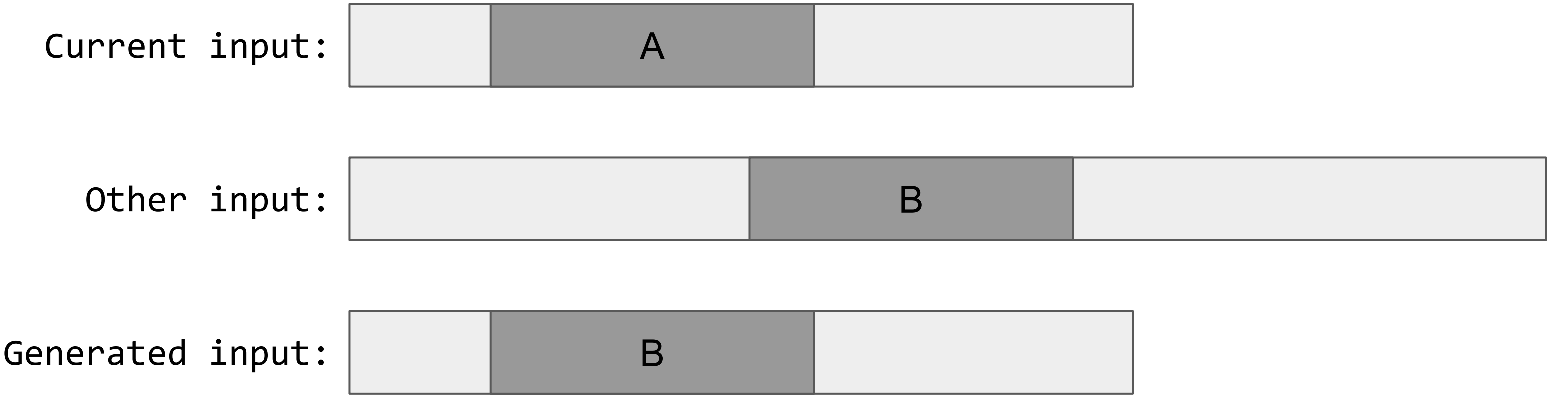
Mutating Chunks [Weizz]

- Deletion
    - Select a chunk and removes it



Mutating Chunks [Weizz]

- Splicing
  - Select a chunk A and replaces it with a chunk from another input in the queue with the same comparison ID in the first tag



Current input: A

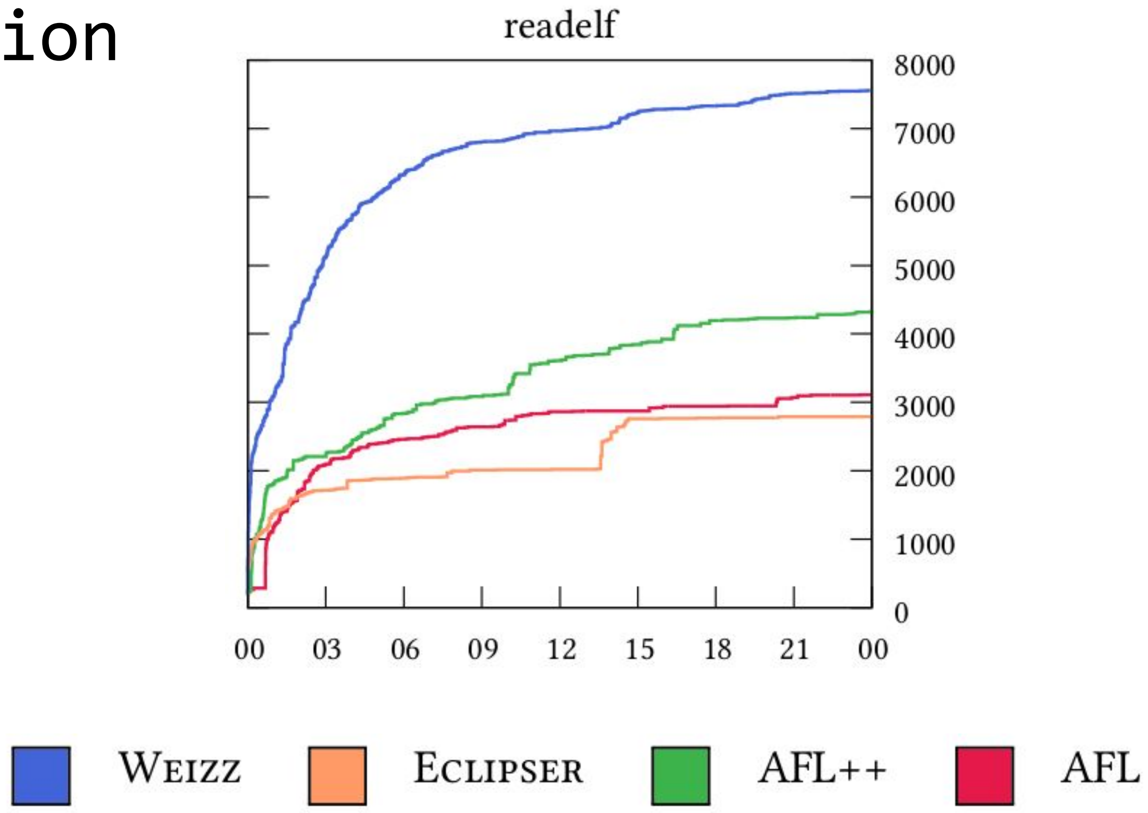
Other input: B

Generated input: B

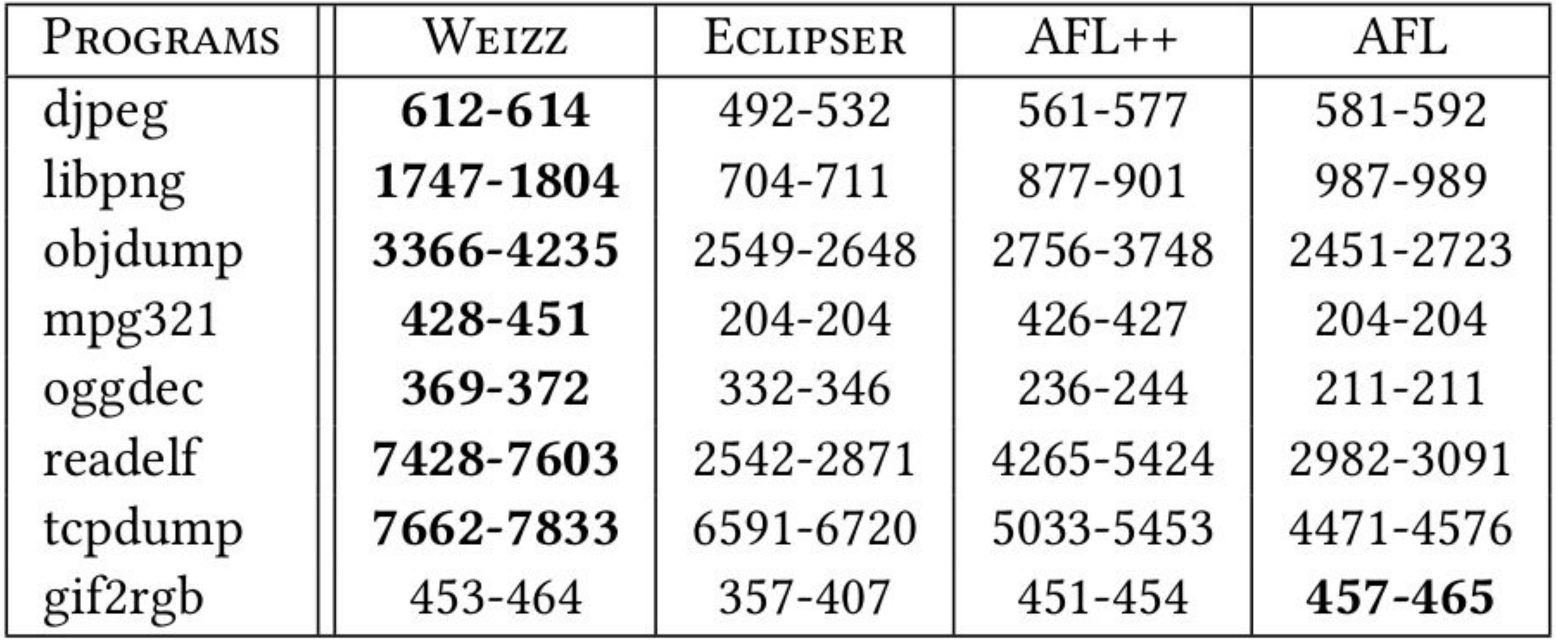
Evaluation

1. Comparison with popular fuzzers over chunk-oriented programs
2. New bugs found by Weizz
3. Role of structural mutations and roadblock bypassing?

Evaluation

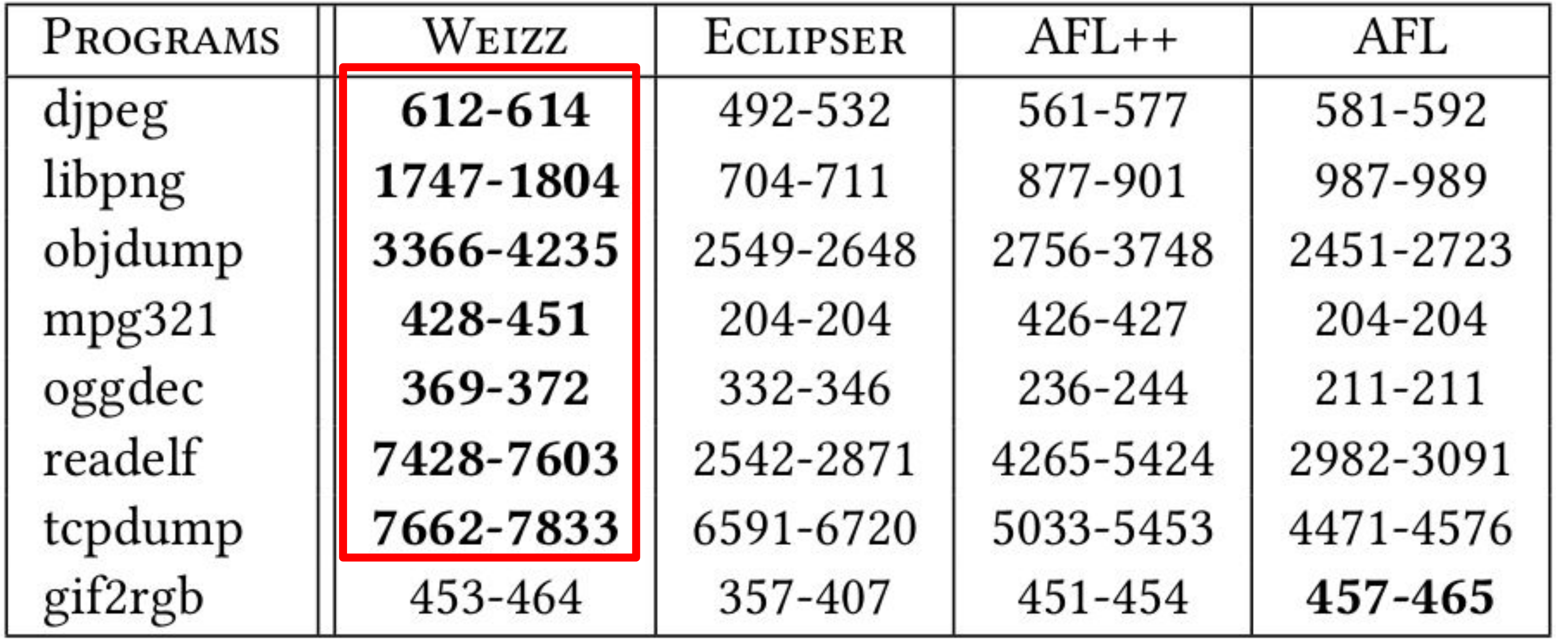

readelf
8000
7000
6000
5000
4000
3000
2000
1000
0
00
03
06
09
12
15
18
21
00
WEIZZ
ECLIPSER
AFL++
AFL

Evaluation (60% conf. intervals)



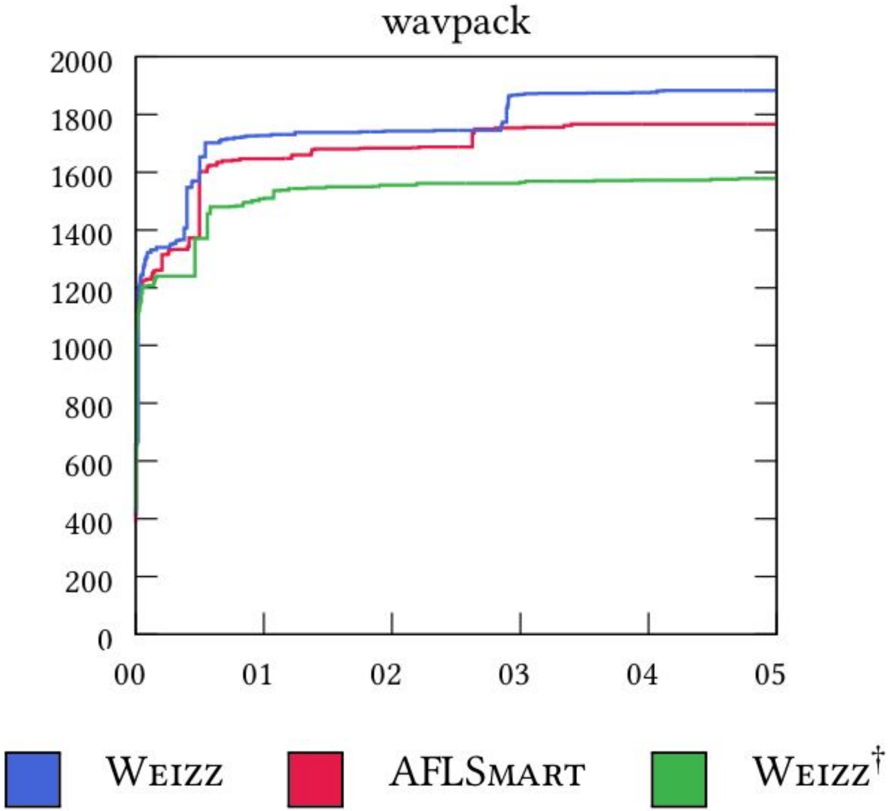
Programs	Weizz	Eclipser	AFL++	AFL
djpeg	**612-614**	492-532	561-577	581-592
libpng	**1747-1804**	704-711	877-901	987-989
objdump	**3366-4235**	2549-2648	2756-3748	2451-2723
mpg321	**428-451**	204-204	426-427	204-204
oggdec	**369-372**	332-346	236-244	211-211
readelf	**7428-7603**	2542-2871	4265-5424	2982-3091
tcpdump	**7662-7833**	6591-6720	5033-5453	4471-4576
gif2rgb	453-464	357-407	451-454	**457-465**

Evaluation (60% conf. intervals)



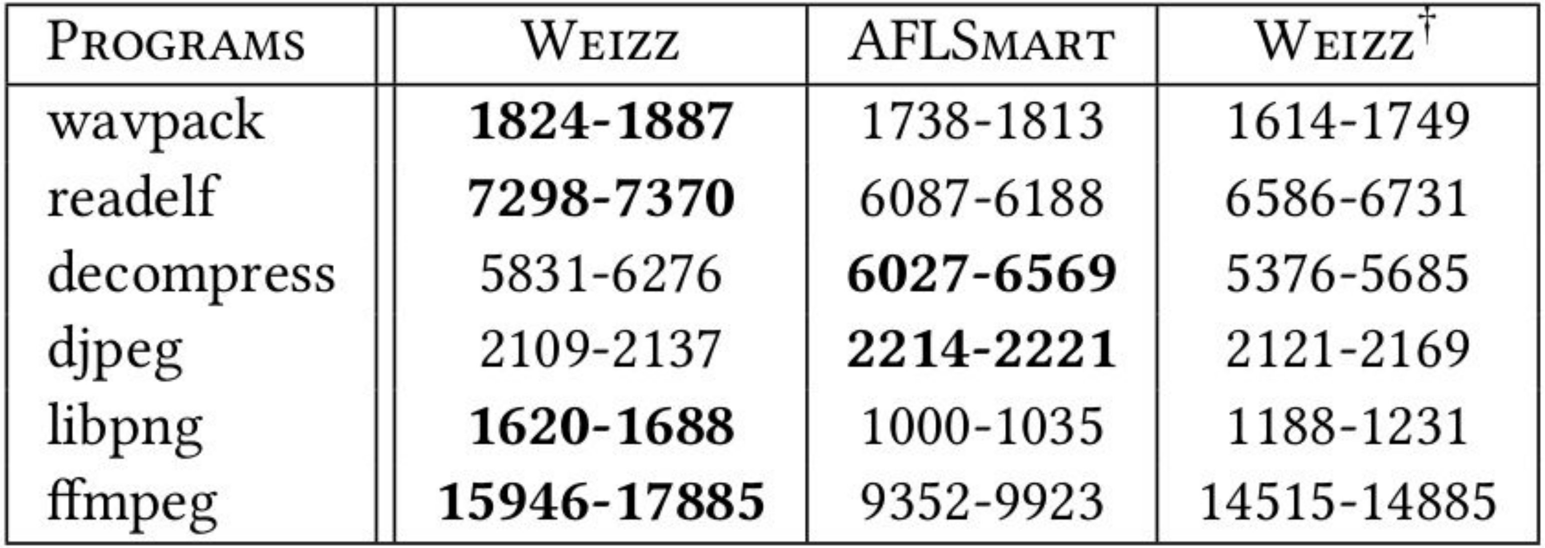
PROGRAMS	WEIZZ	ECLIPSER	AFL++	AFL
djpeg	**612-614**	492-532	561-577	581-592
libpng	**1747-1804**	704-711	877-901	987-989
objdump	**3366-4235**	2549-2648	2756-3748	2451-2723
mpg321	**428-451**	204-204	426-427	204-204
oggdec	**369-372**	332-346	236-244	211-211
readelf	**7428-7603**	2542-2871	4265-5424	2982-3091
tcpdump	**7662-7833**	6591-6720	5033-5453	4471-4576
gif2rgb	453-464	357-407	451-454	**457-465**

Evaluation



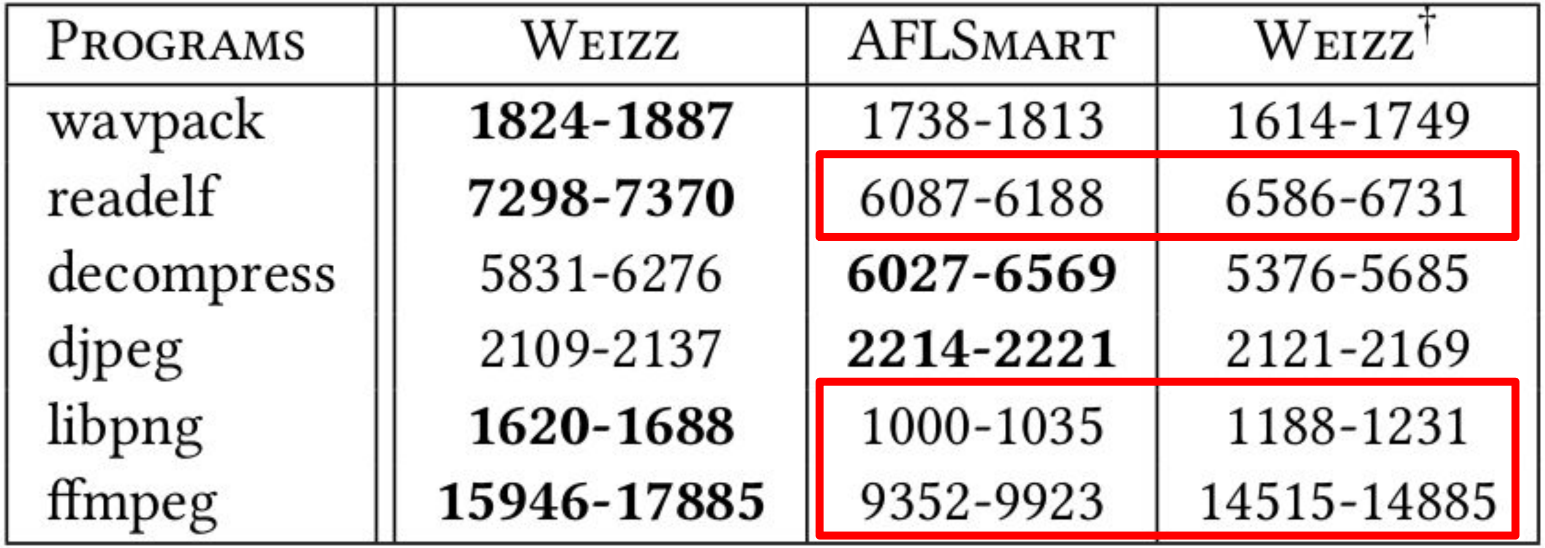
w/o I2S

Evaluation (60% conf. intervals)



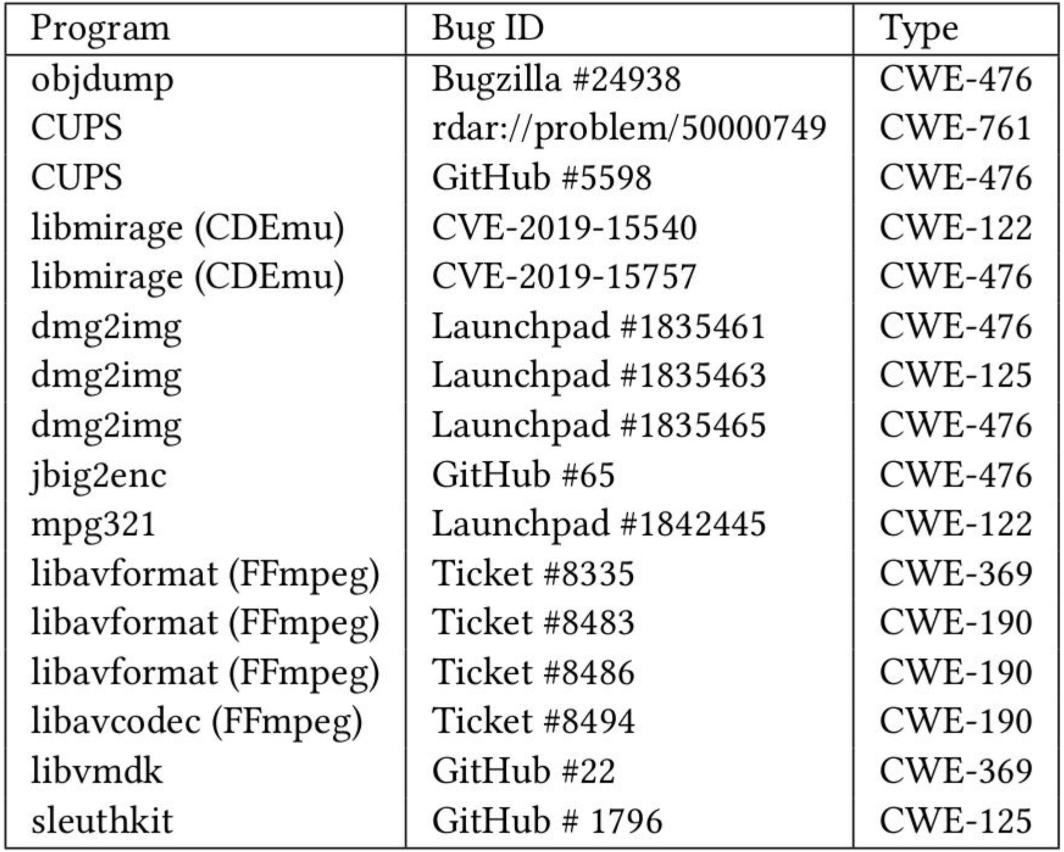
Programs	Weizz	AFLSmart	Weizz†
wavpack	**1824-1887**	1738-1813	1614-1749
readelf	**7298-7370**	6087-6188	6586-6731
decompress	5831-6276	**6027-6569**	5376-5685
djpeg	2109-2137	**2214-2221**	2121-2169
libpng	**1620-1688**	1000-1035	1188-1231
ffmpeg	**15946-17885**	9352-9923	14515-14885

Evaluation (60% conf. intervals)



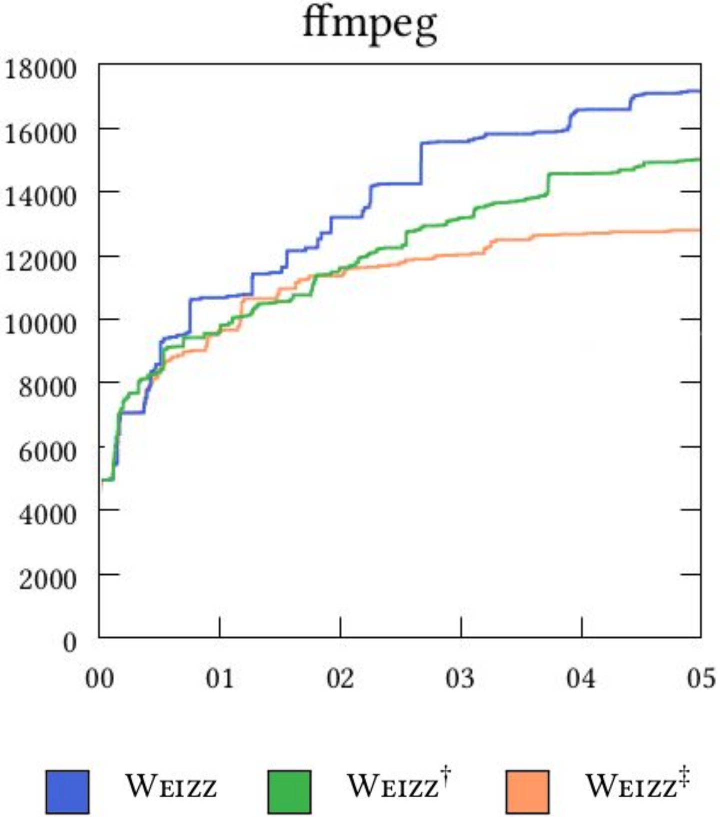
Programs	Weizz	AFLSmart	Weizz†
wavpack	**1824-1887**	1738-1813	1614-1749
readelf	**7298-7370**	6087-6188	6586-6731
decompress	5831-6276	**6027-6569**	5376-5685
djpeg	2109-2137	**2214-2221**	2121-2169
libpng	**1620-1688**	1000-1035	1188-1231
ffmpeg	**15946-17885**	9352-9923	14515-14885

Bugs



Program	Bug ID	Type
objdump	Bugzilla #24938	CWE-476
CUPS	rdar://problem/50000749	CWE-761
CUPS	GitHub #5598	CWE-476
libmirage (CDEmu)	CVE-2019-15540	CWE-122
libmirage (CDEmu)	CVE-2019-15757	CWE-476
dmg2img	Launchpad #1835461	CWE-476
dmg2img	Launchpad #1835463	CWE-125
dmg2img	Launchpad #1835465	CWE-476
jbig2enc	GitHub #65	CWE-476
mpg321	Launchpad #1842445	CWE-122
libavformat (FFmpeg)	Ticket #8335	CWE-369
libavformat (FFmpeg)	Ticket #8483	CWE-190
libavformat (FFmpeg)	Ticket #8486	CWE-190
libavcodec (FFmpeg)	Ticket #8494	CWE-190
libvmdk	GitHub #22	CWE-369
sleuthkit	GitHub # 1796	CWE-125

Evaluation



w/o I2S

w/o struct. mut.

Future Directions

- Taint Tracking for large inputs
- More chunk location heuristics
  - Exclude types of tags as starting point for a chunk
  - Apply traditional file-format reverse engineering algorithms based on memory accesses to tags
- Port to other OSes

Thank You

https://github.com/andreafioraldi/weizz-fuzzer