IDFuzz: Intelligent Directed Grey-box Fuzzing

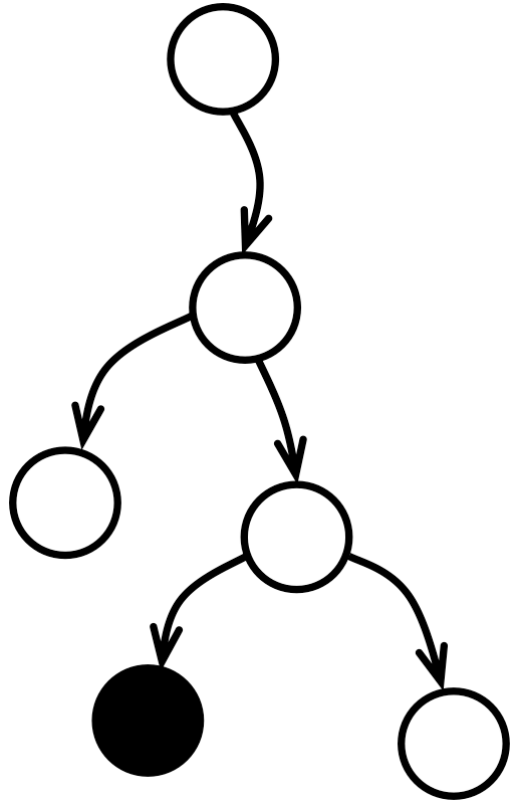
Yiyang Chen, Chao Zhang*, Long Wang*, Wenyu Zhu, Changhua Luo, Nuoqi Gui, Zheyu Ma, Xingjian Zhang, and Bingkai Su

Tsinghua University

Directed Fuzzing

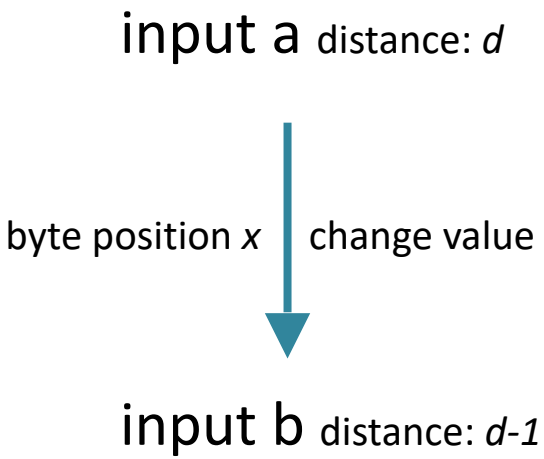
- Directed fuzzing aims to test specific target code sites in programs
  - ➢ crash reproduction
  - ➢ candidate vulnerability confirmation
  - ➢ patch testing

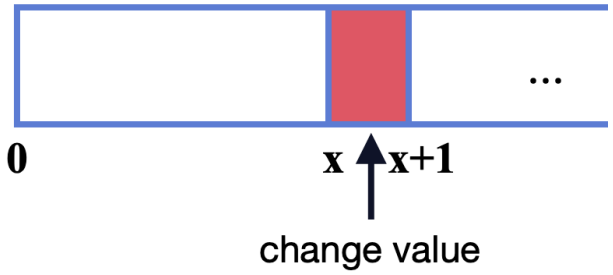
Reach target code more quickly!!!



Motivation

- We can identify ***critical fields*** for mutation based on ***past mutations***





- ***critical field*** of input a: offset *x*, length *1*
- Mutate the critical field directly rather than making blind attempts!!!

Motivation

- ***Q1: Applicable to future mutations?***

  ➢ critical field of input a --> critical field of input x?

- ***Q2: Reach unexplored code regions?***

  ➢ distance: *d* to *d-1* --> *d-1* to 0?

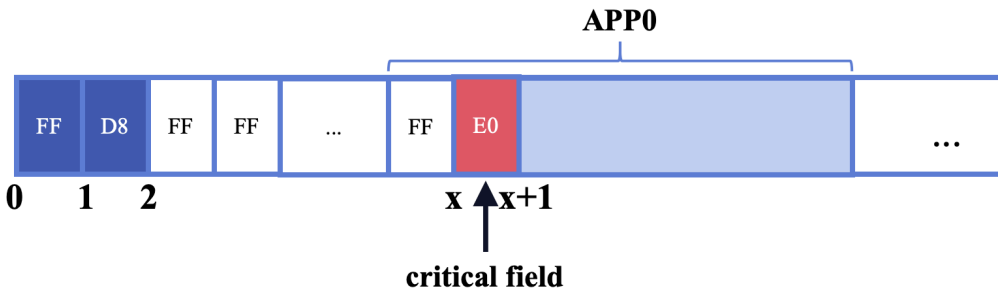
- ***A1:*** Learn the high-level pattern through **a neural network model**
- ***A2:*** learn to reach an unexplored branch from **covered sibling branches**

An example


```
1  static int SectionsRead;
2  int ReadJpegSections(FILE* infile, ReadMode_t ReadMode) {
3      int a = fgetc(infile);
4      if (a != 0xff || fgetc(infile) != M_SOI) return FALSE;
5      for(;;) {
6          int prev = 0;
7          int marker = 0;
8          for (a=0;;a++) {
9              marker = fgetc(infile);
10             if (marker != 0xff && prev == 0xff) break;
11             if (marker == EOF) {
12                 ErrFatal(''Unexpected end of file'');
13             }
14             prev = marker;
15         }
16         Sections[SectionsRead].Type = marker;
17         SectionsRead += 1;
18         switch(marker) {
19             case M_SOS: ...
20             case M_DQT: ...
21             case M_DHT: ...
22             ...
23             case M_COM:
24                 // target
25             ...
26             case M_SOF15: ...
27             default: ...
28         }
29     }
30     return TRUE;
31 }
```

Listing 1: A motivating example.

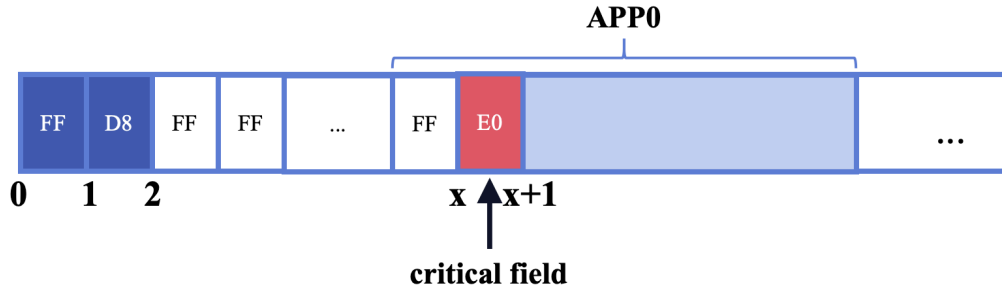
- Part of the *jhead* program for reading a JPEG format input file
- Mutate the **marker** byte to **M_COM**
  - ➢ offset: the 1st non-0xff byte
  - ➢ length: 1



An example


```
1  static int SectionsRead;
2  int ReadJpegSections(FILE* infile, ReadMode_t ReadMode) {
3      int a = fgetc(infile);
4      if (a != 0xff || fgetc(infile) != M_SOI) return FALSE;
5      for(;;) {
6          int prev = 0;
7          int marker = 0;
8          for (a=0;;a++) {
9              marker = fgetc(infile);
10             if (marker != 0xff && prev == 0xff) break;
11             if (marker == EOF) {
12                 ErrFatal(''Unexpected end of file'');
13             }
14             prev = marker;
15         }
16         Sections[SectionsRead].Type = marker;
17         SectionsRead += 1;
18         switch(marker) {
19             case M_SOS: ...
20             case M_DQT: ...
21             case M_DHT: ...
22             ...
23             case M_COM:
24                 // target
25             ...
26             case M_SOF15: ...
27             default: ...
28         }
29     }
30     return TRUE;
31 }
```

Listing 1: A motivating example.



I. Compute input distance as NN output

① do not reach *switch* $d = 1$

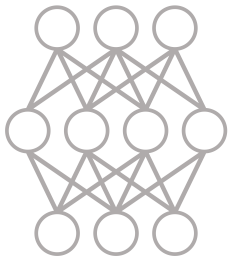
② reach *switch* and take case *M_COM* $d = 0$

③ reach *switch* but take other cases $d = 0.1, 0.2, 0.3, \ldots$

II. Model the relationship between **input bytes** and **distance**

➢ NN input: values of input bytes $<x_1, x_2, \ldots, x_n>$

NN model

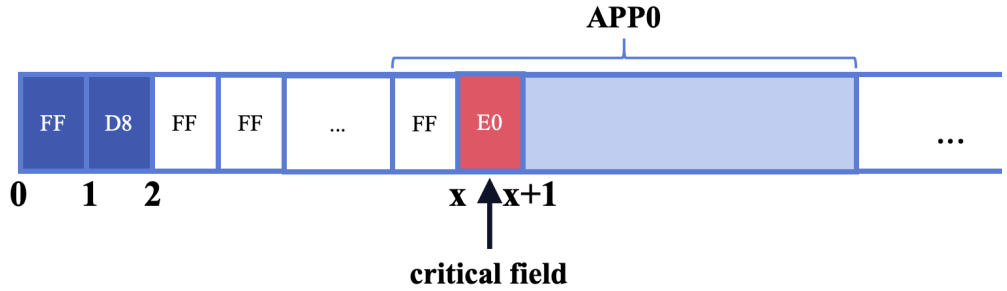


➢ NN output: input distance d

An example


```
1  static int SectionsRead;
2  int ReadJpegSections(FILE* infile, ReadMode_t ReadMode) {
3      int a = fgetc(infile);
4      if (a != 0xff || fgetc(infile) != M_SOI) return FALSE;
5      for(;;) {
6          int prev = 0;
7          int marker = 0;
8          for (a=0;;a++) {
9              marker = fgetc(infile);
10             if (marker != 0xff && prev == 0xff) break;
11             if (marker == EOF) {
12                 ErrFatal(''Unexpected end of file'');
13             }
14             prev = marker;
15         }
16         Sections[SectionsRead].Type = marker;
17         SectionsRead += 1;
18         switch(marker) {
19             case M_SOS: ...
20             case M_DQT: ...
21             case M_DHT: ...
22             ...
23             case M_COM:
24                 // target
25             ...
26             case M_SOF15: ...
27             default: ...
28         }
29     }
30     return TRUE;
31 }
```

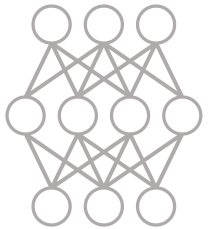
Listing 1: A motivating example.



III. Compute **model gradients** to identify marker byte

- 🤔Changing marker byte has the greatest impact on input distance
- 🤔Marker byte is theoretically associated with the largest absolute model gradient value
- 💡We can identify the marker byte by finding the largest absolute model gradient value!!!

values of input bytes $\langle x_1, x_2, \ldots, x_n \rangle$



input distance d

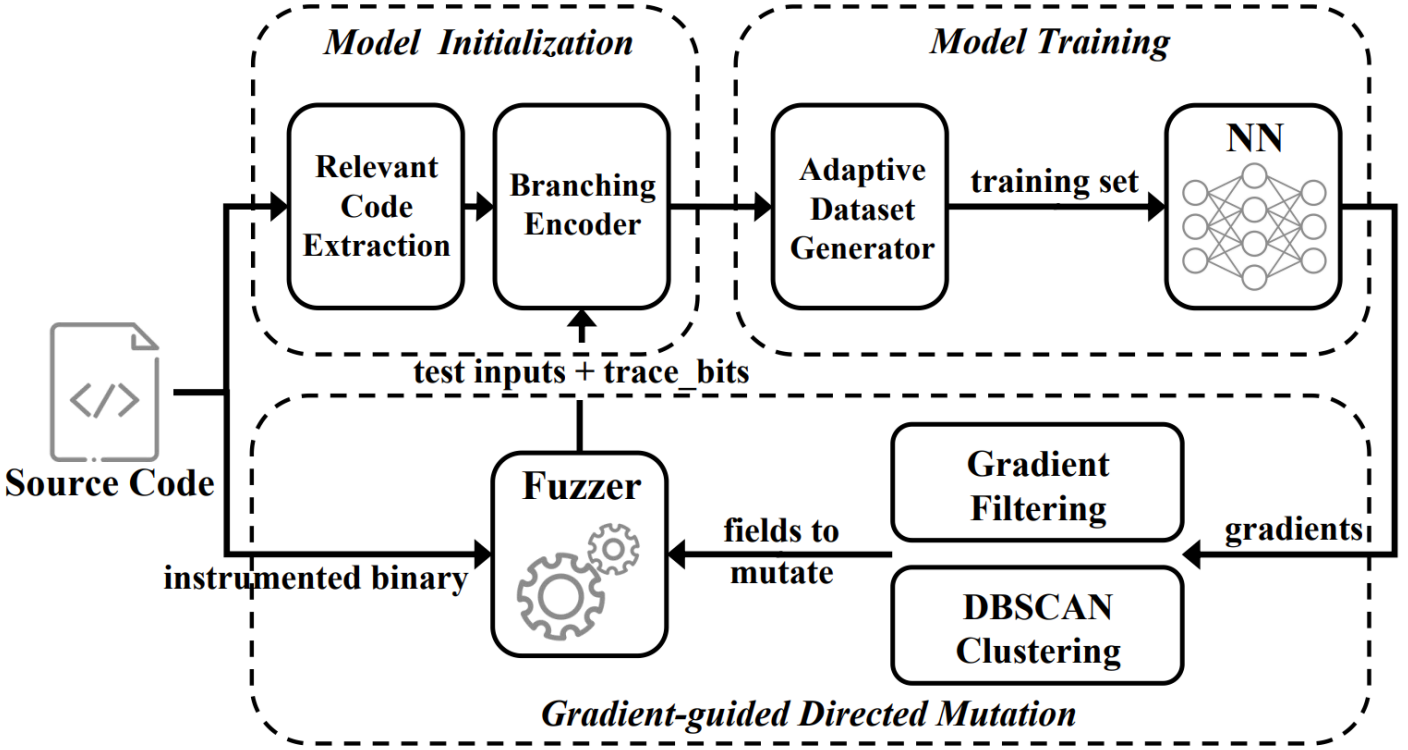
$$\langle \frac{\partial d}{\partial x_1}, \frac{\partial d}{\partial x_2}, \ldots, \frac{\partial d}{\partial x_n} \rangle$$

model gradients

IDFuzz: Key Idea

- Model the relationship between input bytes and input distance using an NN
- Identify the critical bytes/fields by locating the maximum model gradients

IDFuzz Workflow

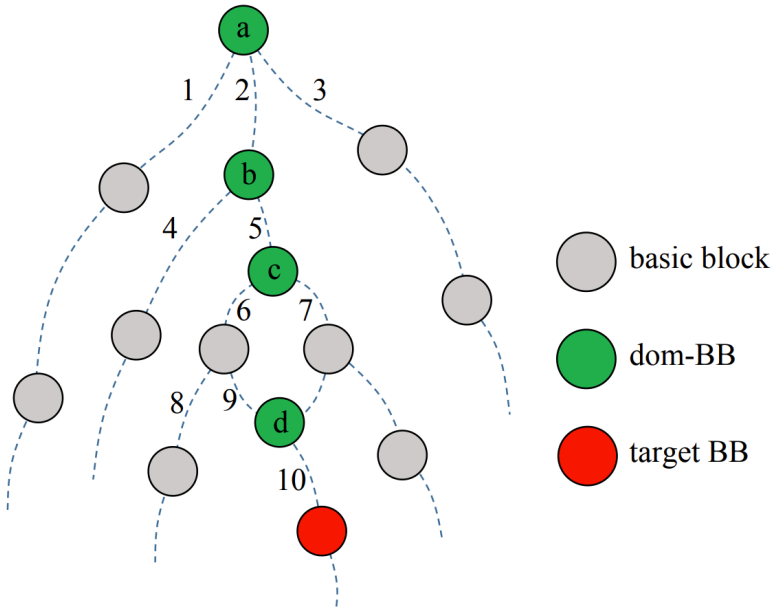


- **Model Initialization**
- **Model Training**
- **Gradient-guided Directed Mutation**

Model Initialization

- Model Input: Input byte values
- Model Output: ???
  - ➢ distance value -> inaccurate **X**
- Relevant Code Extraction
  - ➢ modeling the coverage of only **dominating basic blocks**
- Branching Encoding
  - ➢ 0 for uncovered, 1 for covered
  - ➢ A proper fraction between 0 and 1 for covered sibling branches

"1" : < 1 ,0.25, 0 ,0 ,0 >
"2→4" : < 1 , 1 ,0.5, 0 , 0 >
"2→5→6→8" : < 1 , 1 , 1 , 0 , 0 >



Model Training

- Adaptive Dataset Generation

➢ To obtain a small yet sufficient dataset for model training

- seed queue -> insufficient
- fuzzing inputs generated by every seed -> too big

① Prune early accumulated inputs that cover very few dom-BBs

② Reduce inputs by leveraging the similarity in byte distributions among relative seeds

Gradient-guided Directed Mutation

- Gradient Filtering

  ➢ To eliminate real-world noise caused by uneven input byte distributions

  - magic bytes
  - critical bytes for neighboring dom-BBs

- DBSCAN clustering
  - critical bytes -> critical fields

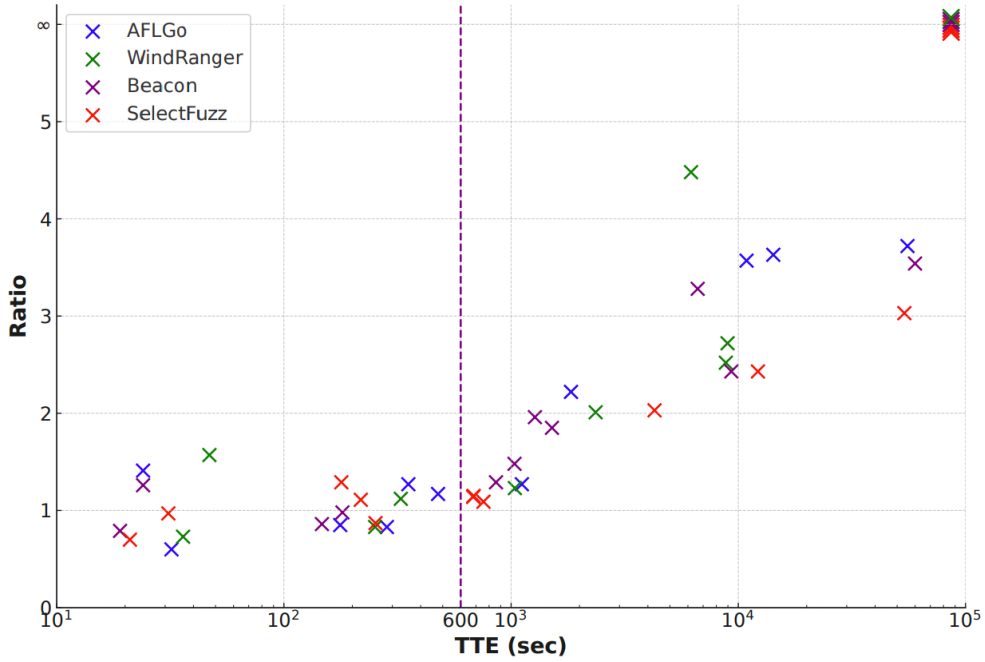
$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Evaluation

- RQ1: How efficiently can IDFuzz accelerate existing directed fuzzers in triggering known vulnerabilities?
- RQ2: How does each component affect IDFuzz's performance?
- RQ3: What is the runtime overhead of IDFuzz?

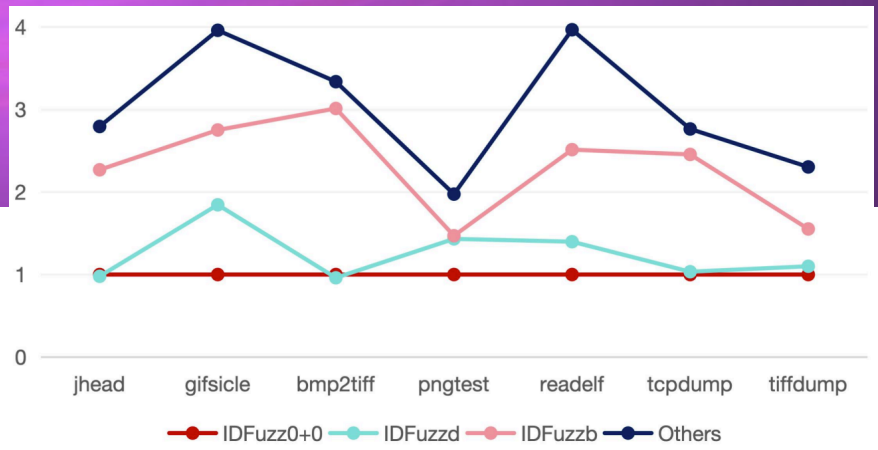
Evaluation

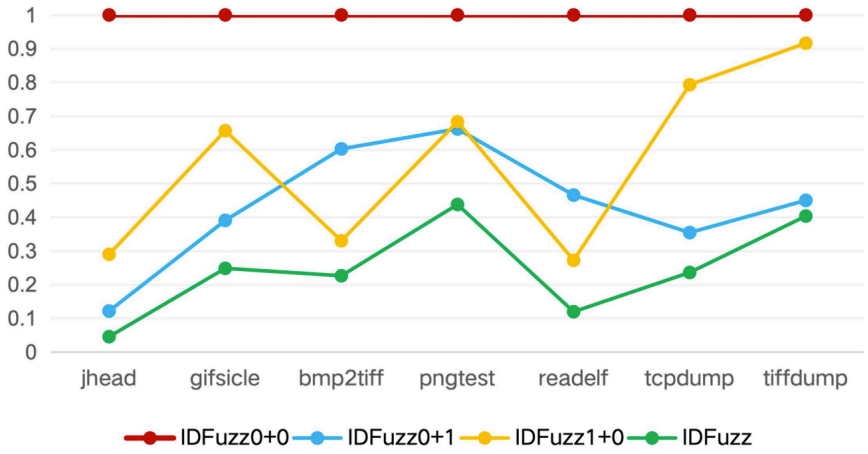
- RQ1
  - ➢ Benchmark: Google Fuzzer Test Suite
  - ➢ Baselines: AFLGo, WindRanger, Beacon, SelectFuzz
  - ➢ We implement IDFuzz as an **input mutation module** and equip the baselines with IDFuzz
  - ➢ Without IDFuzz vs. with IDFuzz -- speedups
    - **AFLGo 2.88x**
    - **WindRanger 2.63x**
    - **Beacon 2.38x**
    - **SelectFuzz 2.01x**

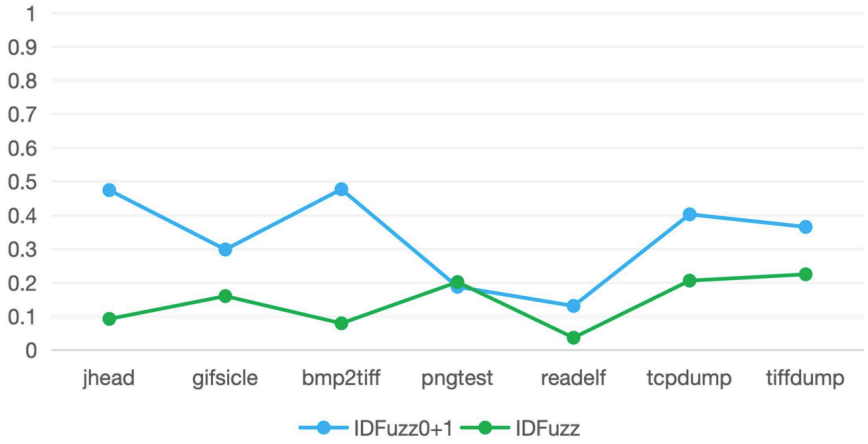


Evaluation

- RQ2
  - ➢ branching encoding
    - **53%** ineffective mutations reduced
  - ➢ adaptive dataset generation
    - **44%** imr
  - ➢ gradient filtering
    - **56%** imr
  - ➢ together
    - **92%** imr







Evaluation

- RQ3
  - ➢ only **6%** increase in overhead

Program	Generation	Training	Analysis	Total Trainings
jhead	18 (0.02%)	4381 (5.07%)	11 (0.01%)	19.8
gifsicle	35 (0.04%)	6040 (6.99%)	37 (0.04%)	18.2
bmp2tiff	17 (0.02%)	4597 (5.32%)	18 (0.02%)	15.8
pngtest	91 (0.11%)	1355 (1.57%)	5 (0.01%)	6.5
readelf	137 (0.16%)	5562 (6.44%)	325 (0.38%)	15.1
tcpdump	39 (0.05%)	6800 (7.87%)	21 (0.02%)	17.9
tiffdump	23 (0.03%)	4478 (5.18%)	29 (0.03%)	13.7
Avg.	51 (0.06%)	4745 (5.49%)	64 (0.07%)	15.3

Evaluation

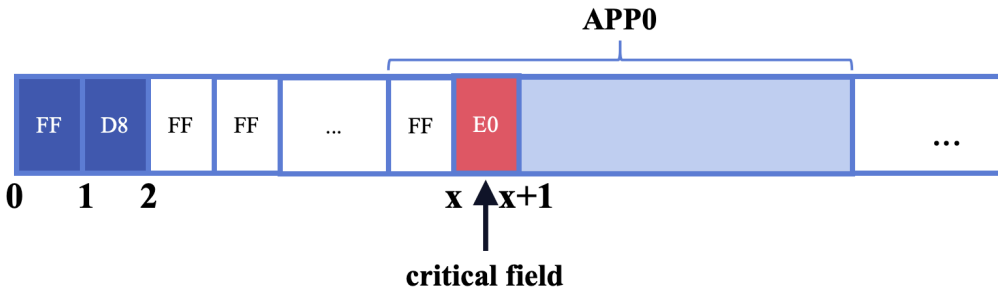
- Case Study

```
1  static int SectionsRead;
2  int ReadJpegSections(FILE* infile, ReadMode_t ReadMode) {
3      int a = fgetc(infile);
4      if (a != 0xff || fgetc(infile) != M_SOI) return FALSE;
5      for(;;) {
6          int prev = 0;
7          int marker = 0;
8          for (a=0;;a++) {
9              marker = fgetc(infile);
10             if (marker != 0xff && prev == 0xff) break;
11             if (marker == EOF) {
12                 ErrFatal(''Unexpected end of file'');
13             }
14             prev = marker;
15         }
16         Sections[SectionsRead].Type = marker;
17         SectionsRead += 1;
18         switch(marker) {
19             case M_SOS: ...
20             case M_DQT: ...
21             case M_DHT: ...
22             ...
23             case M_COM:
24                 // target
25             ...
26             case M_SOF15: ...
27             default: ...
28         }
29     }
30     return TRUE;
31 }
```

Listing 1: A motivating example.

Average file length: **488.36**

Average attempts to hit the marker byte: **6.49!!!**



Evaluation

- **6** new vulnerabilities + **1** incomplete fix

  (**4** CVEs, **2** high-severity vulnerabilities)

Project	ID	Vuln. Type	Status	CVE ID
tcpreplay	#1	NULL Pointer Dereference	patched	CVE-2023-43279
	#2	Infinite Loop	patched	CVE-2024-22654
gifsicle	#3	Floating Point Exception	patched	CVE-2023-46009 (High Severity)
yasm	#4	Memory Leak	patched	
	#5	NULL Pointer Dereference	patched	CVE-2024-22653
w3m	#6	OOB Read	reported	
	#7	OOB Write	patched	Incomplete fix for CVE-2022-38223 (High Severity)

Conclusion

- New Finding:
  - ➢ We can identify critical fields for mutation based on past mutations in directed fuzzing
- Prototype:
  - ➢ IDFuzz
    - https://doi.org/10.5281/zenodo.13753907
    - https://github.com/vul337/IDFuzz
- Strong Results:
  - ➢ 2.48× speedup and 91.86% reduction in ineffective mutations

Thanks!

- Contact me

➢chenyy23@mails.tsinghua.edu.cn