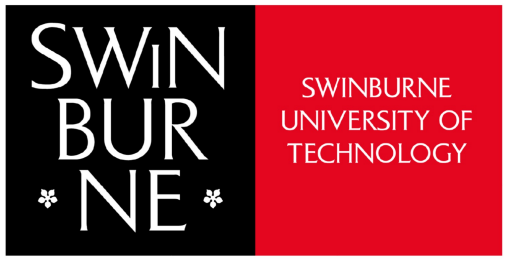
SHAPFUZZ: Efficient Fuzzing via Shapley-Guided Byte Selection

Kunpeng Zhang* (Tsinghua University), Xiaogang Zhu* (Swinburne University of Technology)

Xi Xiao (Tsinghua University), Minhui Xue (CSIRO's Data61)

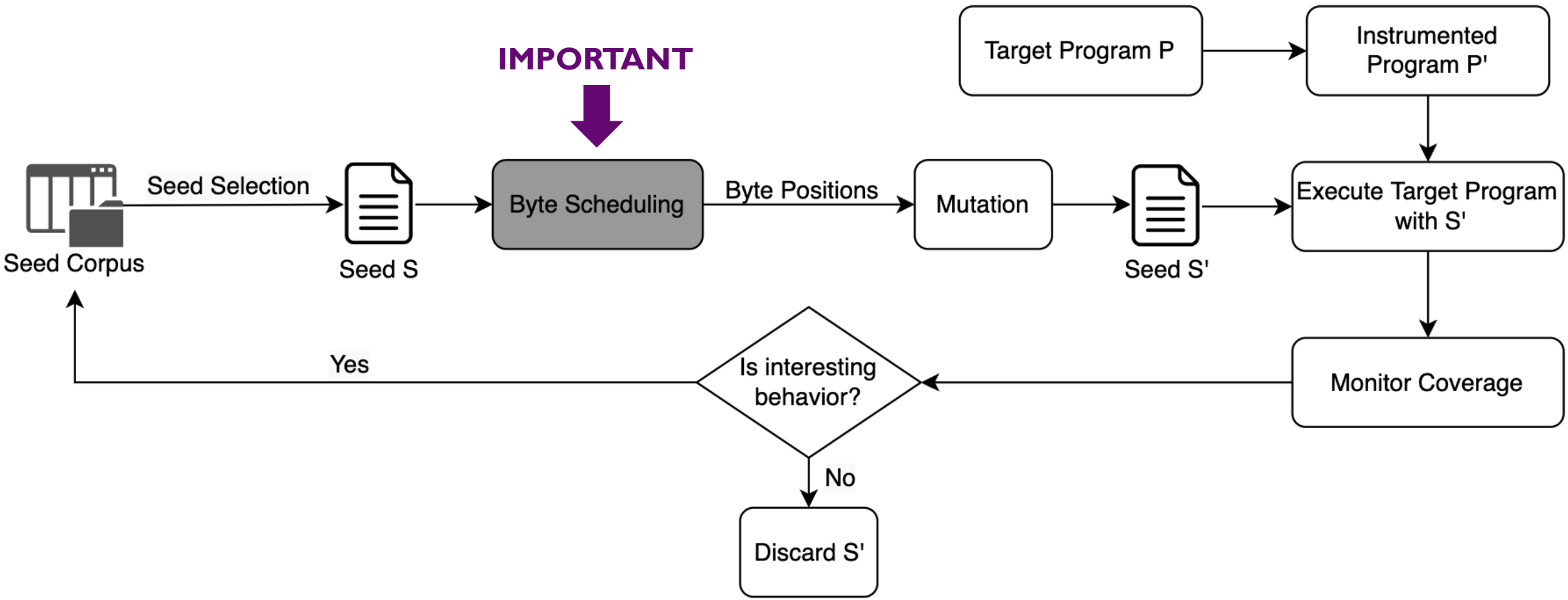
Chao Zhang (Tsinghua University), Sheng Wen (Swinburne University of Technology)



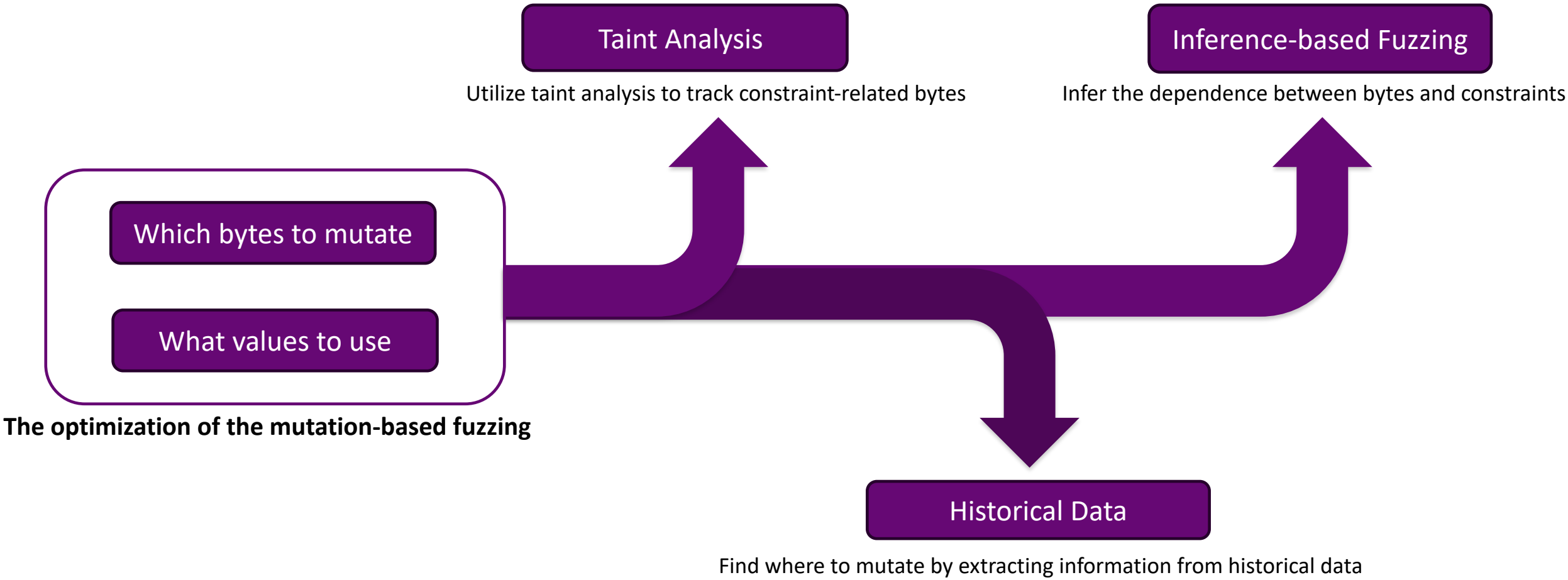
01

MOTIVATION

Background - Fuzzing



Motivation



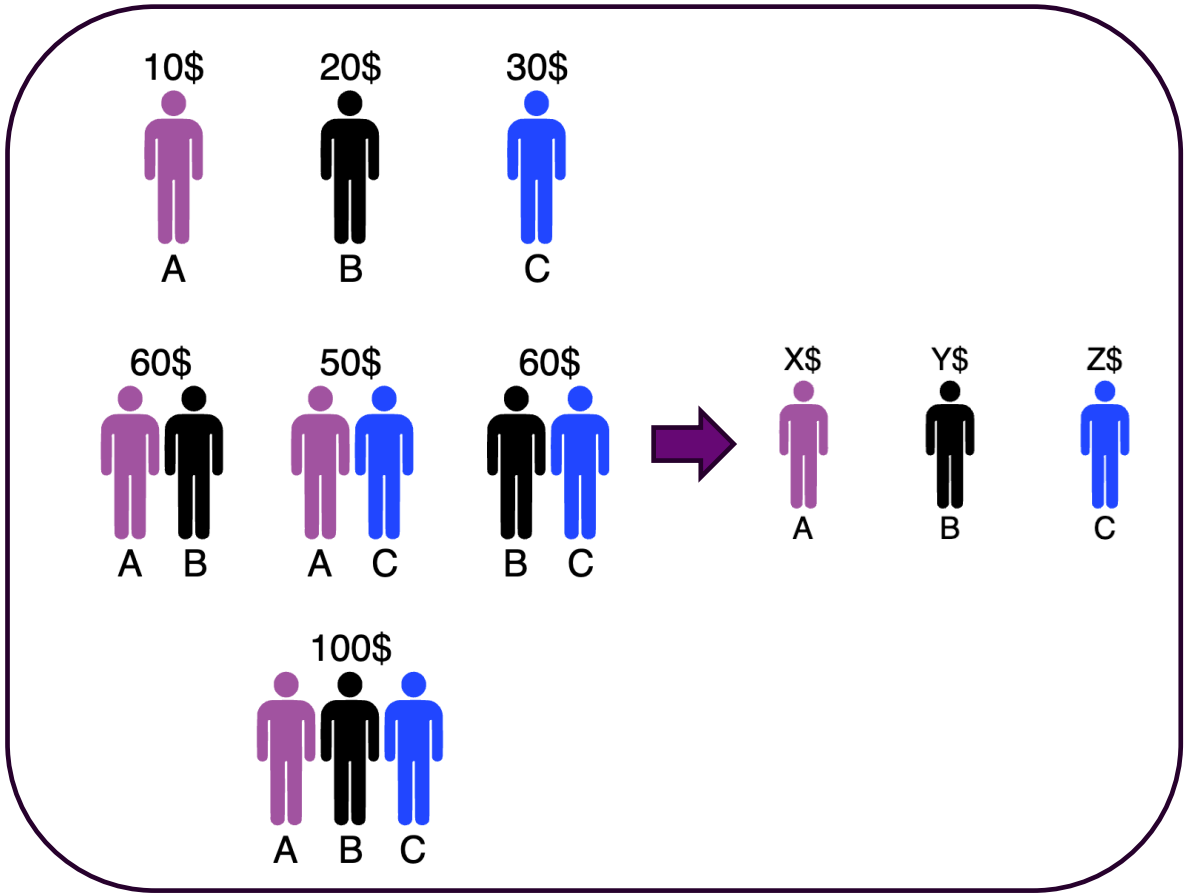
The optimization of the mutation-based fuzzing

Motivation

- Observation
  - While mutating constraint-related bytes indeed improves the efficiency of code discovery, not all constraint-related bytes are able to discover new code
  - However, existing solutions treat all constraint-related bytes equally, wasting time and energy on the ones that cannot discover new code
- Insight
  - Quantify the importance of constraint-related bytes, prioritizing the mutation of more important bytes to enhance fuzzing efficiency

Motivation

- To achieve this insight, we first need to design experiments to validate our observations.



Shapley Value

determines

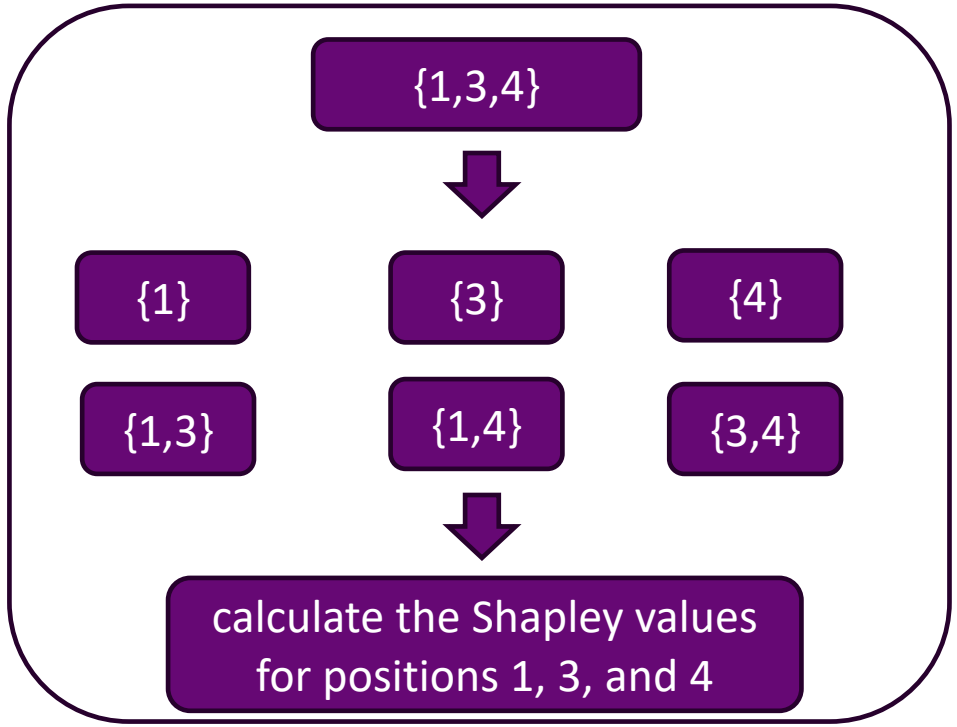
the contribution of each player participating in a collaborative work

Motivation

- Experiment Setup
  - To calculate Shapley values of bytes in a seed, we regard the number of new edges discovered by a combination as the gain
  - To obtain a relatively accurate Shapley value for a byte, we run fuzzing with random mutation for a single seed
  - Time: 48 hours (continuously mutate the initial seed)
  - Fuzzer: AFL++
- Programs
  - 18 programs: nm, tiff2bw, flvmeta, imginfo, infotocap, lame, ……
  - 9 different types of inputs: elf, tiff, pdf, text, mp4, flv, wav, jpg and mp3

Motivation

- The results indicate that a small portion of bytes contribute the most to discovering new edges.



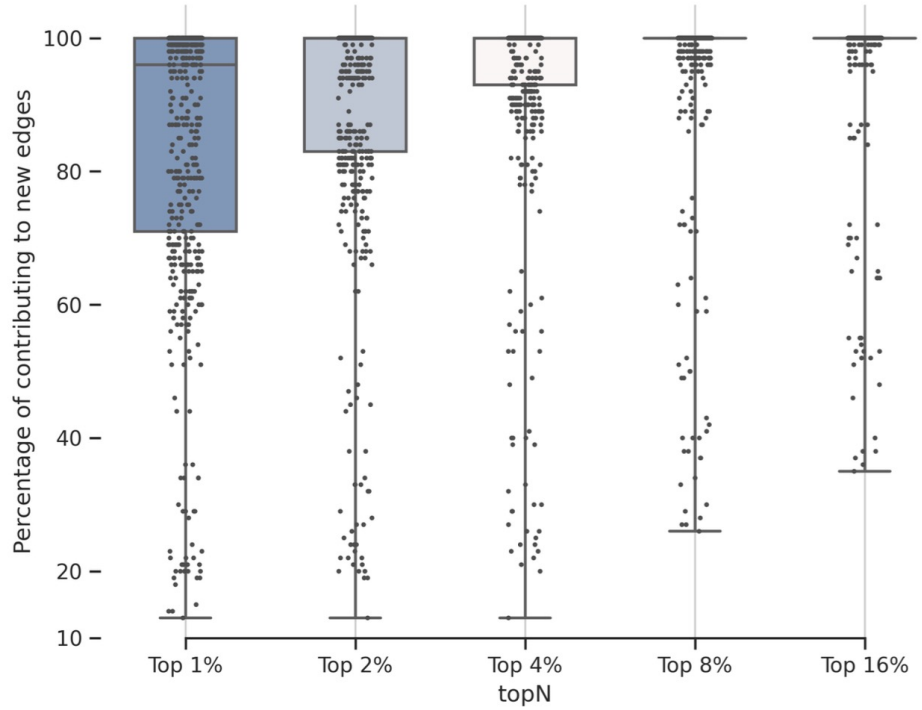
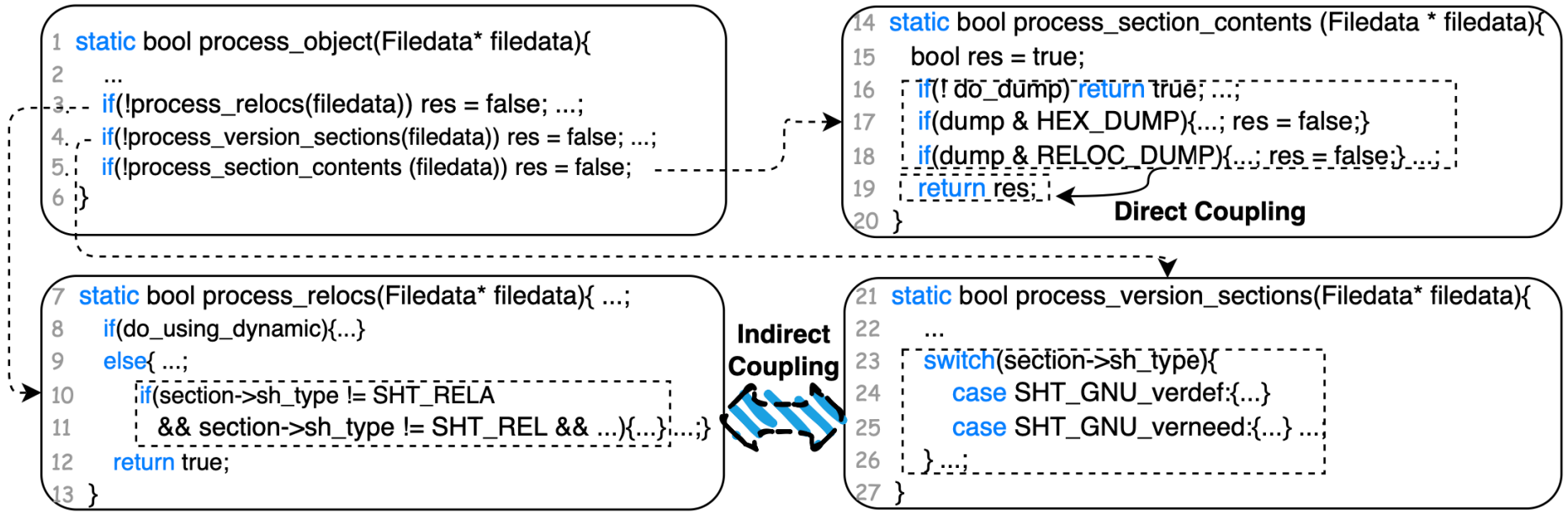


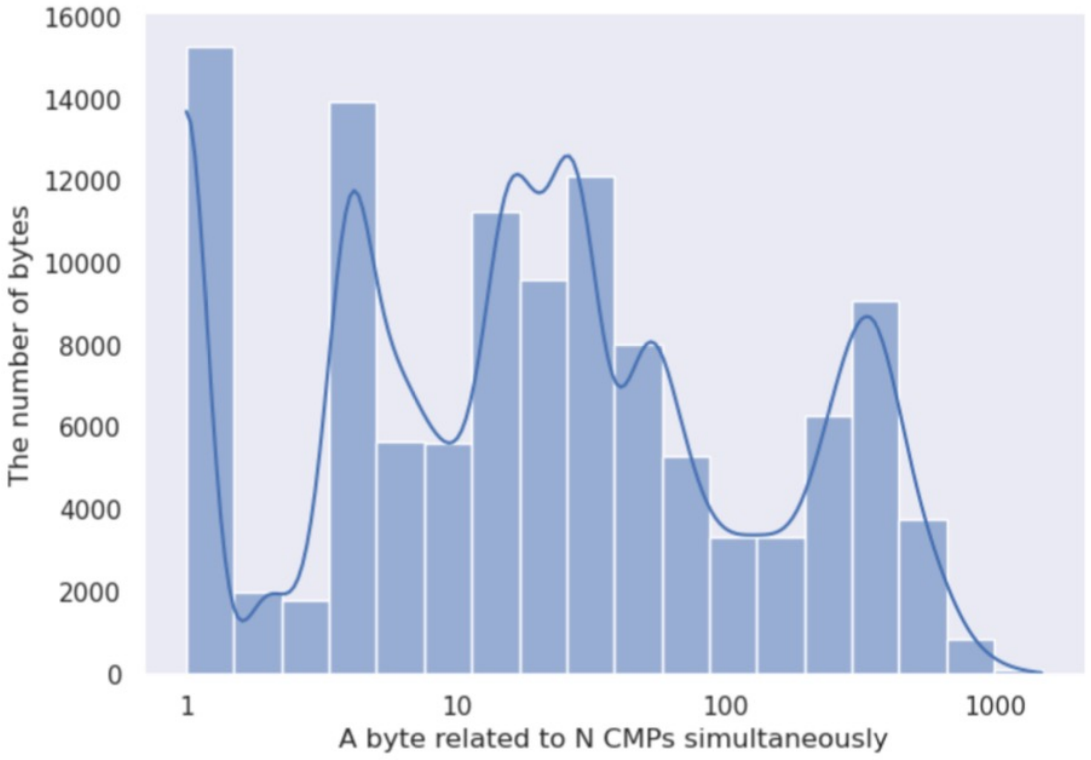
Fig. 1. The boxplot of contributions of bytes. Y% of new edges are contributed by the top X% of bytes. The top X% of bytes refers to the first X% bytes with the largest Shapley values. Each dot is a seed of a program.

Motivation



The reason for the results of the Shapley analysis. The code is extracted from readelf.

Motivation



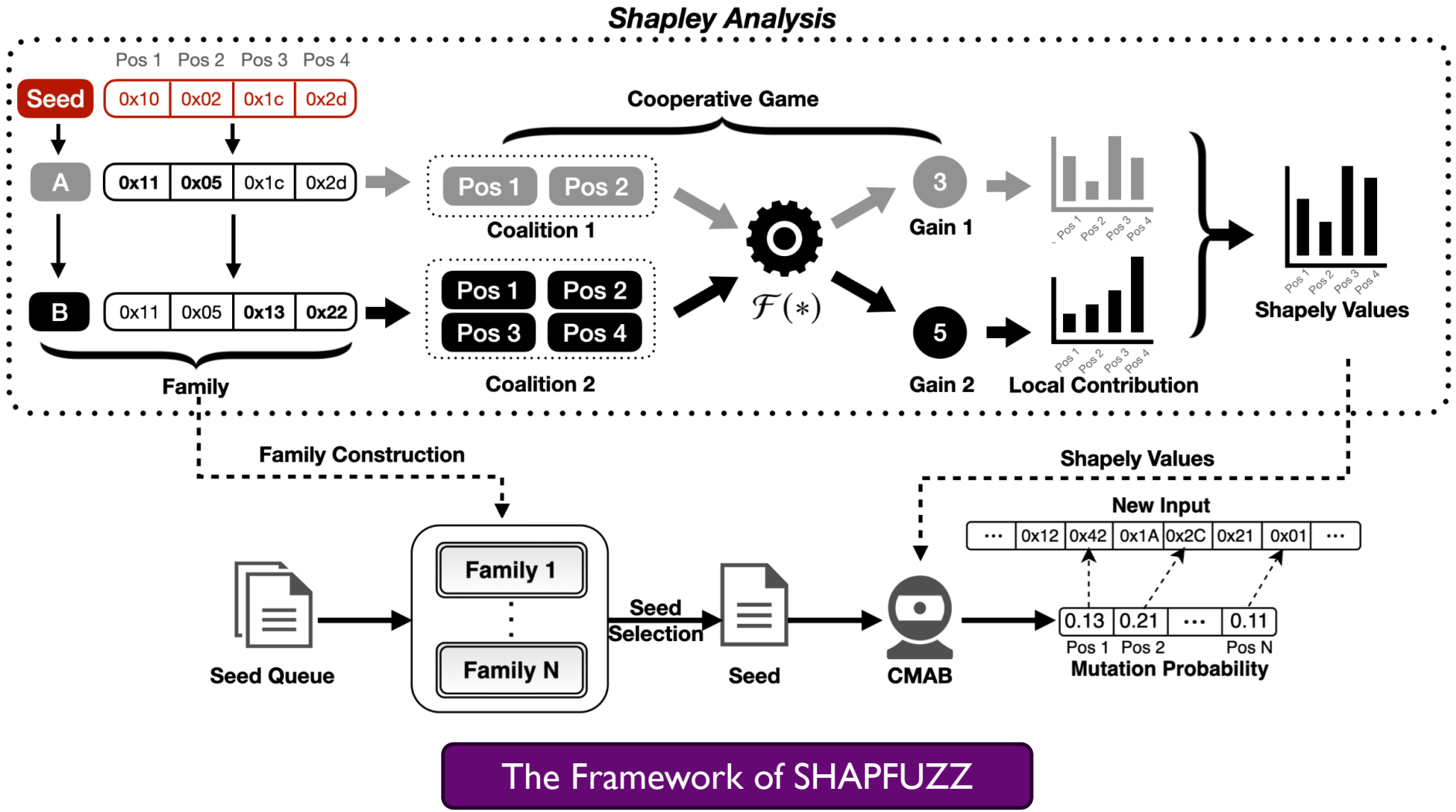
- Insight: Since only a small portion of bytes contribute the most to the discovery of new code, the Shapley analysis can be utilized to obtain those high-importance bytes, and more energy is assigned to them during fuzzing.

Statistics of the relationship between bytes and CMPs across 16 programs

02

METHOD

Framework

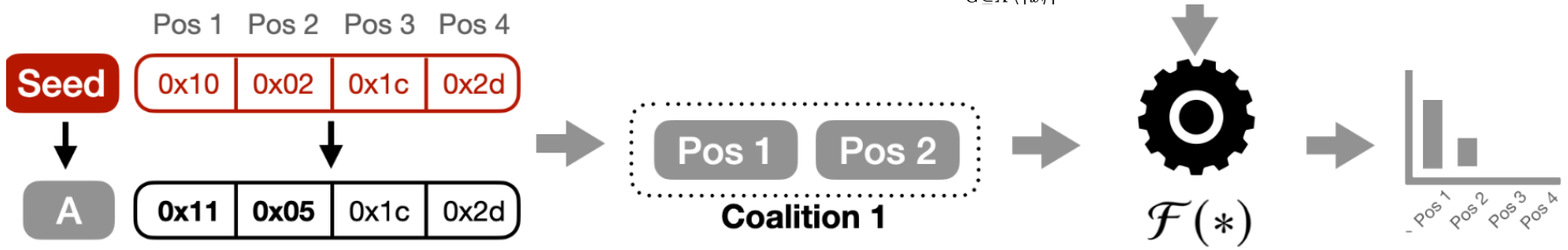


The Framework of SHAPFUZZ

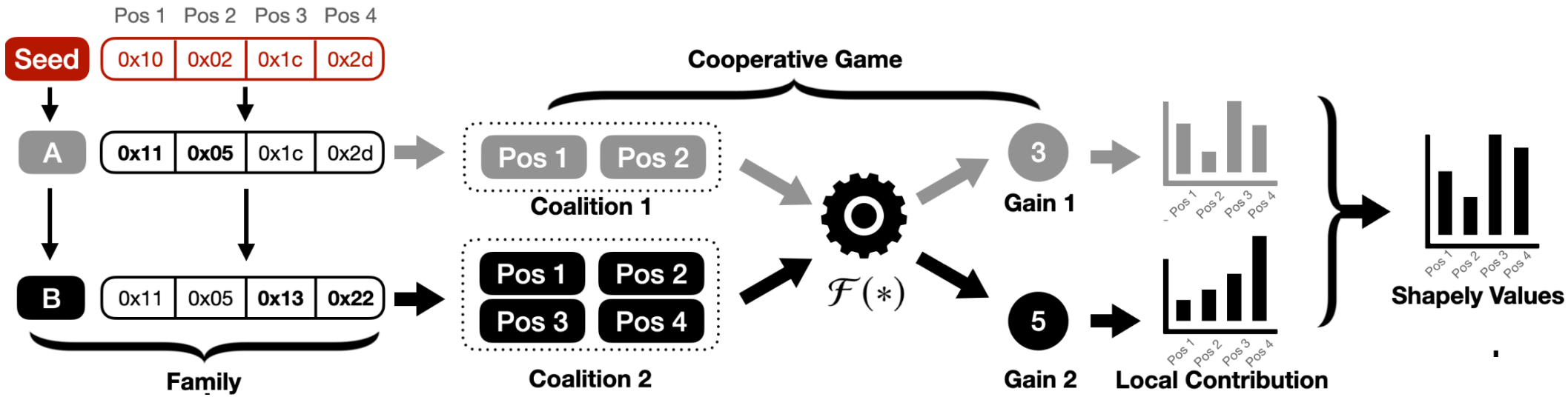
Shapley Analysis in One Seed

- Cooperative game: The schedule of bytes on the seed
- Player: A byte in the seed
- Coalition: Some certain bytes are mutated together
- Gain: The number of *self-new edges* discovered by an input i generated by mutating the seed
  - *Self-new edges* are defined as the new edges when comparing the edges discovered by the input i and the initial seed
- Characteristic function: the mapping between the collaborative mutation and the number of self-new edges

$$\phi_j = \sum_{G \subset X \setminus \{x_i\}} \frac{|G|!(N - |G| - 1)!}{N!} (\mathcal{F}(G \cup \{x_j\}) - \mathcal{F}(G)),$$

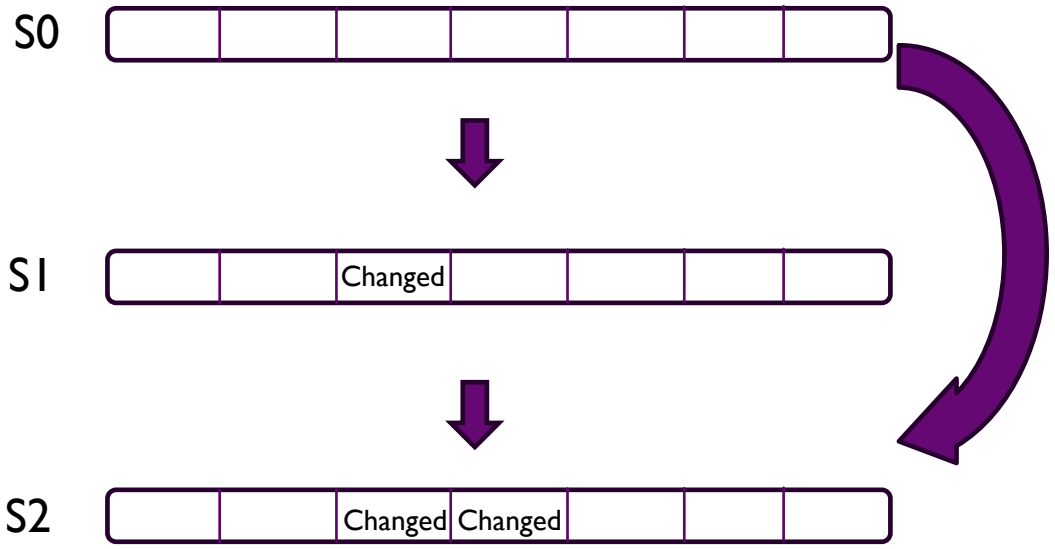


Shapley Analysis Across Seeds

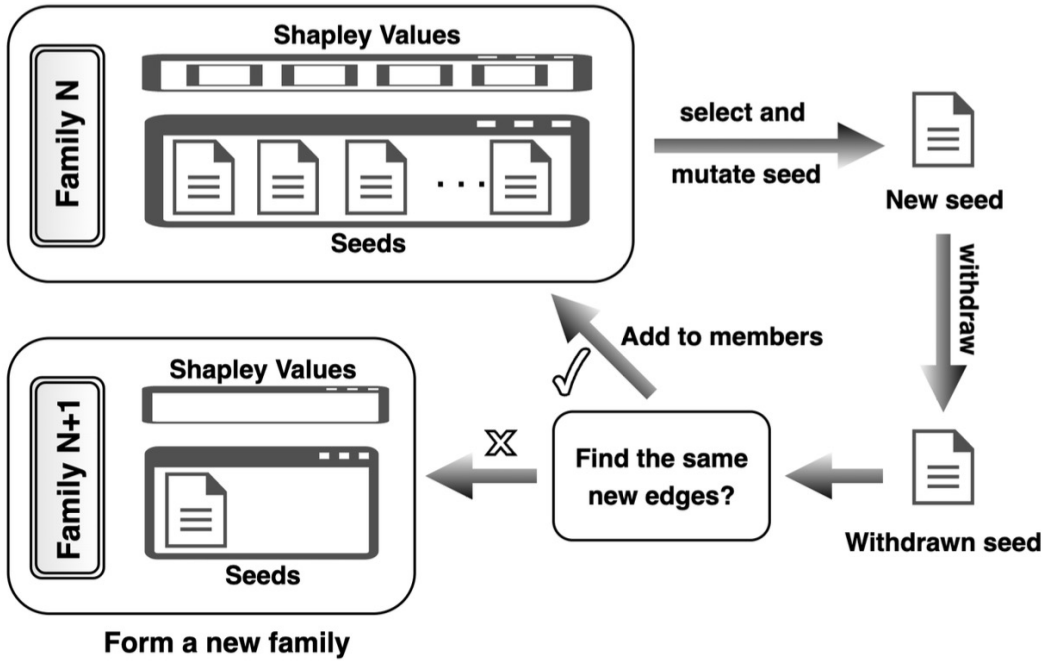


The seeds, which are retained from the same original seed and do not change the length, are part of the combinations for the original seed

Shapley Analysis Across Seeds

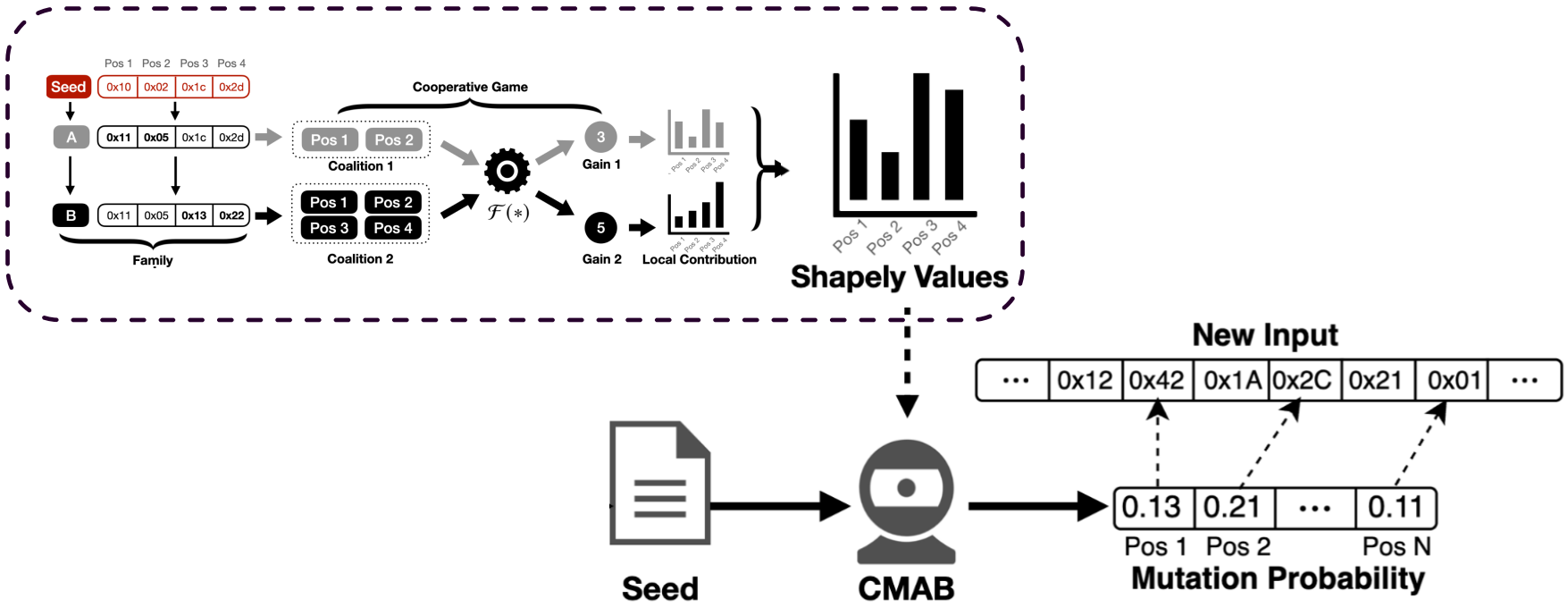


Track the mutated positions



The update of family and family members

Shapley-guided Byte Selection



02

EVALUATION

SHAPFUZZ vs. Byte-Scheduling Fuzzers

- Compared Fuzzers
  - Inference-based fuzzer: Greyone and ProFuzzer
  - Neural network-based fuzzer: PreFuzz and NEUZZ
  - Taint-based fuzzer: Angora
- Experiment Setup
  - Platform
    - UNIFUZZ
  - Experiments
    - ALL SEEDS
    - SEEDS<10000 (seeds that have less than 10,000 bytes)
    - SEEDS<1000 (seeds that have less than 1,000 bytes)

SHAPFUZZ vs. Byte-Scheduling Fuzzers

TABLE II. EDGE COVERAGE AND ANALYSIS TIME COMPARISON BETWEEN DIFFERENT FUZZERS (ON ALL SEEDS).

Programs		SHAPFUZZ			GreyOne			ProFuzzer			Angora			PreFuzz			NEUZZ		
Name	**Len**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**
tiff2pdf	448	4401	85s	0	4486	35207s	0	4578	12694s	1	2314	235s	0	3475	G	1	2832	G[1]	1
lame	13818	3656	3985s	5	3645	80225s	4	3649	70003s	4	2265	5813s	4	-	-	0	-[2]	-	0
readelf	272030	5611	1048s	4	5058	78788s	3	5168	74878s	6	5786	3758s	6	-	-	0	-	-	0
exiv2	25633	3790	135s	15	3626	35401s	5	3698	32311s	12	4291	615s	11	2866	G	0	-	-	0
flvmeta	16454	230	0s	2	230	32782s	2	230	11625s	2	230	201s	2	-	-	0	-	-	0
nm	272030	3136	248s	17	2628	83058s	9	2750	81747s	10	2657	2351s	51	-	-	0	-	-	0
tiffsplit	10032	1709	5s	7	1699	13398s	5	1719	6220s	6	890	291s	1	1189	G	2	1113	G	1
tiff2bw	10032	1839	87s	6	1870	12709s	6	1848	8702s	5	1180	310s	1	1651	G	0	1441	G	0
objdump	272030	4958	742s	14	3864	79211s	4	4106	83913s	5	3192	3672s	3	-	-	0	-	-	0
pdftotext	12465	6613	3508s	23	5794	80548s	2	5777	80591s	3	4250	14801s	2	4902	G	5	4394	G	0
mp42aac	31988	1266	19s	2	1144	75949s	0	1166	66141s	0	1120	540s	0	1017	G	0	950	G	0
tcpdump	6983	12764	328s	1	11558	50599s	1	11879	29383s	0	7615	2510s	3	4672	G	0	6019	G	0
mujs	6983	4136	0s	0	4020	18157s	0	3979	7181s	0	2364	8829s	0	2432	G	0	2393	G	0
size	272030	1860	188s	0	1667	77562s	0	1690	78671s	0	1988	2032s	0	-	-	0	-	-	0
infotocap	2519	1817	276s	7	1670	39193s	6	1530	44335s	5	940	1104s	0	1266	G	1	1071	G	0
imginfo	2519	1895	30s	0	1818	36298s	0	1744	43274s	0	1384	184s	0	-	-	0	-	-	0
Total		**59681**		**103**	54777		47	55511		59	42466		84	23470		9	20213		2

[1] G: A GPU is required to train the model. [2] -: The fuzzer fails to run on this program due to large input size.

SHAPFUZZ vs. Byte-Scheduling Fuzzers

TABLE III. EDGE COVERAGE AND ANALYSIS TIME COMPARISON BETWEEN DIFFERENT FUZZERS (ON SEEDS<10000).

Programs		SHAPFUZZ			GreyOne			ProFuzzer			Angora			PreFuzz			NEUZZ		
Name	**Len**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**
tiff2pdf	448	4461	30s	1	4450	39189s	0	4552	15278s	0	2306	50s	0	3466	G	0	2543	G[1]	0
readelf	4230	5359	77s	5	5346	39952s	6	5272	38537s	3	5554	1628s	7	4540	G	6	4208	G	4
exiv2	8437	3882	196s	7	3385	50513s	4	3691	31915s	3	3903	2365s	9	3881	G	0	3151	G	0
nm	4230	2822	17s	30	2818	29069s	20	2884	13905s	22	2409	2056s	35	2260	G	14	1749	G	0
tiffsplit	7222	1718	4s	6	1725	13541s	6	1715	4519s	7	919	219s	3	1203	G	2	1128	G	0
tiff2bw	7222	1882	42s	7	1876	12031s	6	1808	6040s	6	1186	200s	1	1594	G	0	1489	G	0
objdump	4230	4654	104s	16	4145	75874s	10	4414	50760s	13	3085	4182s	13	3560	G	4	3136	G	0
tcpdump	2305	12963	227s	1	12164	40025s	1	12497	13022s	0	7826	2119s	1	7963	G	1	6251	G	0
mujs	2305	4145	0s	0	4000	19877s	0	4015	7078s	0	2383	8677s	0	2589	G	1	2388	G	0
size	4230	1795	22s	0	1785	12623s	0	1812	9547s	0	1794	1259s	1	1458	G	0	1224	G	0
infotocap	2519	1807	457s	8	1767	38278s	5	1399	45154s	2	840	1647s	0	1159	G	1	1074	G	0
imginfo	2519	2486	9s	0	1744	26514s	0	1680	24268s	0	1413	244s	0	1464	G	0	1176	G	0
Total		**47974**		**81**	45205		58	45739		56	33618		70	35137		29	29517		4

[1] G: A GPU is required to train the model.

SHAPFUZZ vs. Byte-Scheduling Fuzzers

TABLE IV. EDGE COVERAGE AND ANALYSIS TIME COMPARISON BETWEEN DIFFERENT FUZZERS (ON SEEDS<1000).

Programs		SHAPFUZZ			GreyOne			ProFuzzer			Angora			PreFuzz			NEUZZ		
Name	**Len**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**	**Cov.**	**Time**	**#Bug**
tiff2pdf	448	4383	99s	1	4493	39407s	0	4449	12856s	1	2459	58s	0	3398	G	1	2709	G[1]	1
readelf	324	4983	34s	4	4875	13088s	3	4763	5978s	0	5131	299s	5	4144	G	0	3760	G	0
nm	324	1876	2s	0	1872	9869s	0	1891	4521s	0	1838	65s	0	1424	G	0	1103	G	0
tiffsplit	858	1721	4s	6	1712	17187s	4	1697	8743s	6	935	19s	4	1155	G	2	1134	G	0
tiff2bw	858	1882	6s	8	1829	13096s	4	1861	7278s	8	1263	13s	1	1533	G	0	1435	G	0
objdump	324	3629	25s	0	3667	19014s	0	3731	9879s	0	2422	1143s	0	2624	G	1	2275	G	0
tcpdump	451	11594	59s	0	10465	19800s	0	10827	7126s	0	7355	2077s	1	5948	G	0	4499	G	0
mujs	451	4153	0s	0	3973	15638s	0	4027	6055s	0	2421	8605s	0	2590	G	1	2418	G	0
size	324	1752	4s	0	1733	7343s	0	1747	3455s	0	1672	128s	0	1331	G	0	1225	G	0
infotocap	432	1824	131s	7	1782	33321s	6	1679	34827s	5	895	160s	0	1164	G	0	1308	G	0
imginfo	432	1625	2s	0	1494	26077s	0	1611	24070s	0	1307	43s	0	1413	G	0	1105	G	0
Total		**39422**		**26**	37895		17	38283		20	27698		11	26724		5	22971		1

[1] G: A GPU is required to train the model.

SHAPFUZZ vs. Commonly Used Fuzzers

- Compared Fuzzers
  - AFL
  - AFL++: enable Redqueen mutator and MOPT mutator
  - AFLFast
  - FairFuzz
  - MOPT: set −L = 5
- Experiment Setup
  - Platform
    - UNIFUZZ
    - MAGMA

SHAPFUZZ vs. Commonly Used Fuzzers

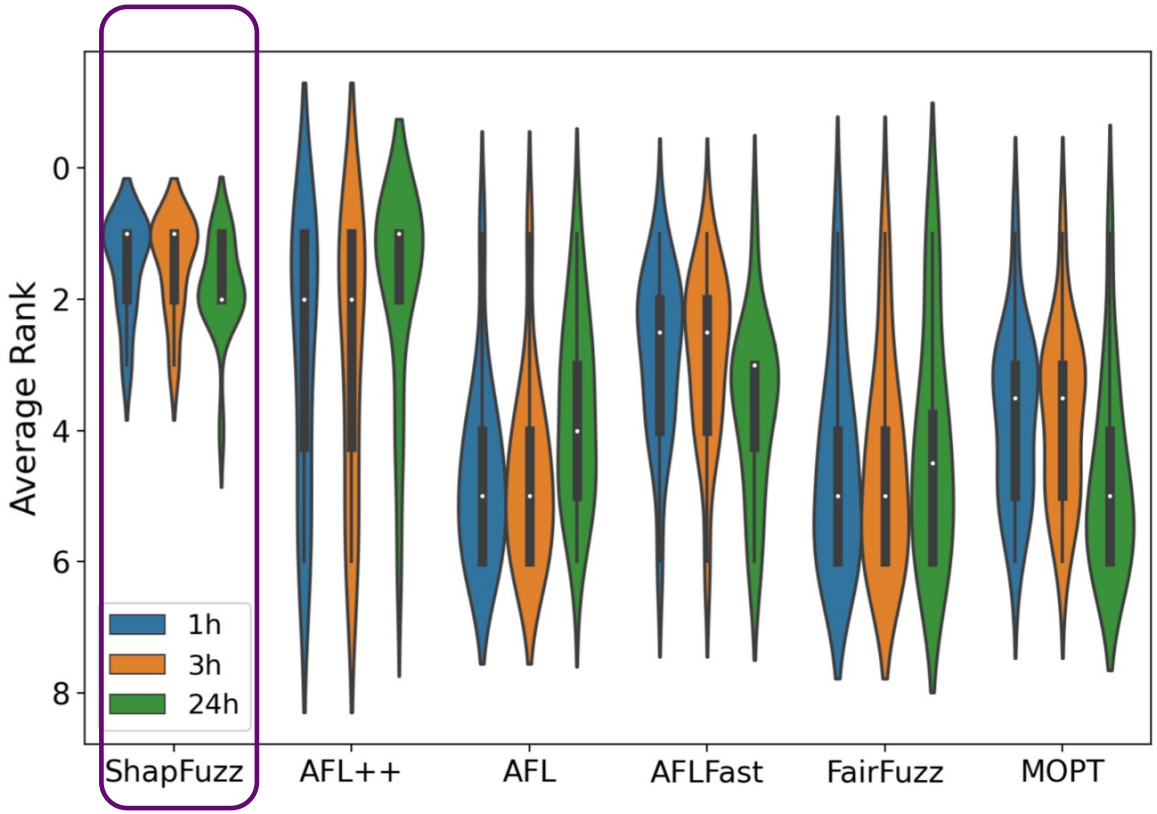


Fig. 6. The violin plot of average edge ranks across 10 trials of different fuzzers after running for 1/3/24 hours.

TABLE VI. THE NUMBER OF UNIQUE BUGS DISCOVERED BY DIFFERENT FUZZERS (ON UNIFUZZ).

Programs	SHAPFUZZ	AFL++	MOPT	AFL	AFLFast	FairFuzz
tiff2pdf	1	2	1	0	0	0
lame	4	4	4	5	4	4
readelf	6	4	4	3	5	5
exiv2	18	23	3	15	15	14
flvmeta	2	2	2	2	2	2
nm	35	31	21	22	22	30
tiffsplit	7	9	7	5	7	5
tiff2bw	10	6	5	5	5	7
objdump	29	11	8	12	10	8
pdftotext	27	24	20	25	24	23
mp42aac	0	5	0	0	0	0
tcpdump	2	1	0	2	1	2
infotocap	11	9	9	5	9	9
imginfo	0	1	0	2	0	0
Total	**152**	132	84	103	104	109

UNIFUZZ

SHAPFUZZ vs. Commonly Used Fuzzers

TABLE X. THE TIME TO BUG (TTB) OF FUZZERS (ON MAGMA). SHAPFUZZ PERFORMS THE BEST IN DISCOVERING BUGS.

Vulnerabilities	SHAPFUZZ	AFL++	MOPT	FairFuzz	AFL	AFLFast
CVE-2015-8472	17s	23s	**15s**	**15s**	**15s**	**15s**
CVE-2016-1762	47s	1m	**15s**	20s	16s	18s
CVE-2018-13988	1m	1m	**52s**	2m	1m	1m
CVE-2016-2109	3m	4m	**1m**	2m	**1m**	2m
CVE-2016-6309	4m	5m	**1m**	2m	**1m**	**1m**
CVE-2016-10270	**23s**	48s	19m	3m	23m	11m
CVE-2016-3658	**4m**	17m	1h	25m	1h	1h
CVE-2018-14883	18m	1h	3m	2h	**2m**	3m
CVE-2017-6892	**13m**	32m	33m	3h	**13m**	1h
SND017	**4m**	5m	25m	21h	38m	5m
CVE-2017-11613	**9m**	1h	1h	15h	6h	4h
CVE-2019-11034	6h	15h	**1m**	8m	2m	**1m**
PDF010	3h	1h	7h	16h	10h	**2m**
CVE-2019-7663	**3h**	8h	4h	9h	16h	6h
CVE-2019-20218	**49m**	2h	16h	11h	9h	9h
CVE-2019-9638	**41m**	17h	7h	17h	4h	44m
CVE-2015-8317	**22m**	1h	3h	-	-	12h
CVE-2019-7310	**7h**	17h	9h	15h	**7h**	8h
CVE-2020-15945	5h	**4h**	14h	15h	13h	16h
SND020	**16m**	19m	7h	21h	-	-
CVE-2015-3414	**1h**	4h	17h	16h	23h	15h
CVE-2016-10269	2h	**1h**	15h	22h	15h	-
CVE-2011-2696	**18m**	22m	17h	-	22h	23h
CVE-2017-8363	**13m**	28m	18h	22h	-	-
CVE-2017-8363	**27m**	32m	18h	22h	-	-
CVE-2017-9047	**33m**	6h	12h	-	-	-
CVE-2017-8361	**14m**	3h	18h	22h	-	-
CVE-2016-6302	9h	23h	16h	-	**3h**	6h
CVE-2017-7375	6h	**1h**	-	-	-	-
CVE-2018-10768	**5h**	**5h**	23h	-	-	23h
CVE-2016-5314	19h	19h	11h	16h	**4h**	-
CVE-2013-6954	13h	19h	16h	12h	**11h**	-
CVE-2019-19926	**5h**	19h	17h	-	21h	20h
PNG006	-	**6m**	-	-	-	-
CVE-2016-2108	**3h**	16h	22h	-	-	-
CVE-2016-10269	23h	**2h**	22h	-	-	-
CVE-2017-9865	**8h**	12h	-	-	-	-
CVE-2015-8784	22h	21h	20h	**18h**	-	20h
CVE-2016-1836	**22h**	23h	-	-	-	-
CVE-2017-14617	-	**21h**	-	-	-	-
CVE-2017-3735	20h	-	**5h**	15h	22h	10h
CVE-2017-9776	**19h**	-	-	-	-	-
CVE-2013-7443	**17h**	-	-	-	-	-
CVE-2019-19880	**18h**	-	-	-	-	-
PDF008	**20h**	-	-	-	-	-
CVE-2019-19646	**22h**	-	-	-	-	-
CVE-2017-2518	**22h**	-	-	-	-	-
CVE-2019-19317	**22h**	-	-	-	-	-
#The Fastest[1]	**29**	7	7	2	9	4

[1] #The Fastest: The number of vulnerabilities that are discovered the fastest by the fuzzer.

MAGMA

THANK YOU

T S I N G H U A U N I V E R S I T Y