

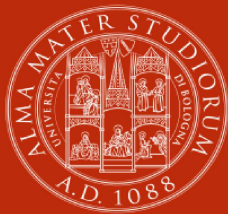
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CPS Project

Insecure Edge Counting for Control-Flow Integrity

Rocco Versace – m. 1138644

Alessandro Frullo – m. 1132704



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Main Goal

Main Goal

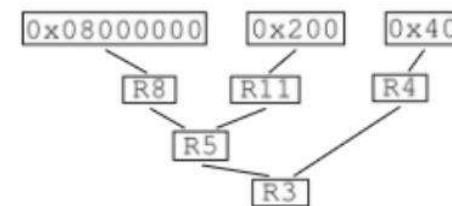
- This project aims at counting the number of indirect jumps and indirect calls to be protected in RISC-V binaries, excluding the ones which can be proven safe

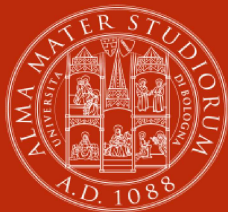
How can we verify if a jump is considered safe?



***We need to retrieve
the origin tree of the
source operand of the
target***

```
MOV R8, #1
LSL R8, R8, #27
MOV R11, 0x200
MOV R4, 0x40
ADD R5, R8, R11
ADD R3, R4, R5
BX R3
```





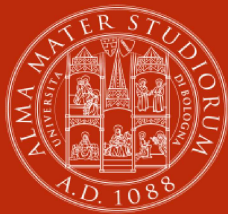
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Starting Point

Starting Point

- This project starts where previous research left off in *"PROLEPSIS: Binary Instrumentation Tool for Control-Flow Integrity in ARM and RISC-V"*
<https://webthesis.biblio.polito.it/secure/24598/1/tesi.pdf>
- While the previous work focused on **static** analysis, there's room for enhancement in **dynamic** conditions in order to retrieve the targets of indirect jumps.
- For this project some benchmark have been run:
<https://github.com/embench/embench-iot.git>





ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Instrumentation

Instrumentation

- Benchmarks were compiled and run with a **Qemu RISC-V32** environment, obtaining the static assembly code of each program, together with their program counter dynamic record



- The data obtained from the benchmarks were then processed with **Python** scripts



Obtaining assembly

```
riscv64-unknown-elf-objdump -D -S -M numeric main > main_DSnum.log
```

```
Disassembly of section .text:

00010074 <main>:
10074:      1101          addi    x2,x2,-32
10076:      ce06          sw      x1,28(x2)
10078:      2a25          jal     101b0 <initialise_board>
1007a:      2a39          jal     10198 <initialise_benchmark>
1007c:      4505          li      x10,1
1007e:      2a31          jal     1019a <warm_caches>
10080:      2a15          jal     101b4 <start_trigger>
10082:      2a29          jal     1019c <benchmark>
10084:      c62a          sw      x10,12(x2)
10086:      2a0d          jal     101b8 <stop_trigger>
10088:      4532          lw      x10,12(x2)
1008a:      2a21          jal     101a2 <verify_benchmark>
1008c:      40f2          lw      x1,28(x2)
1008e:      00153513      seqz    x10,x10
10092:      6105          addi    x2,x2,32
10094:      8082          ret

00010096 <register_fini>:
10096:      00000793      li      x15,0
1009a:      c791          beqz    x15,100a6 <register_fini+0x10>
1009c:      00000517      auipc   x10,0x0
100a0:      33450513      addi    x10,x10,820 # 103d0 <__libc_fini_array>
100a4:      a695          j       10408 <atexit>
100a6:      8082          ret
```



Obtaining dynamic program counter

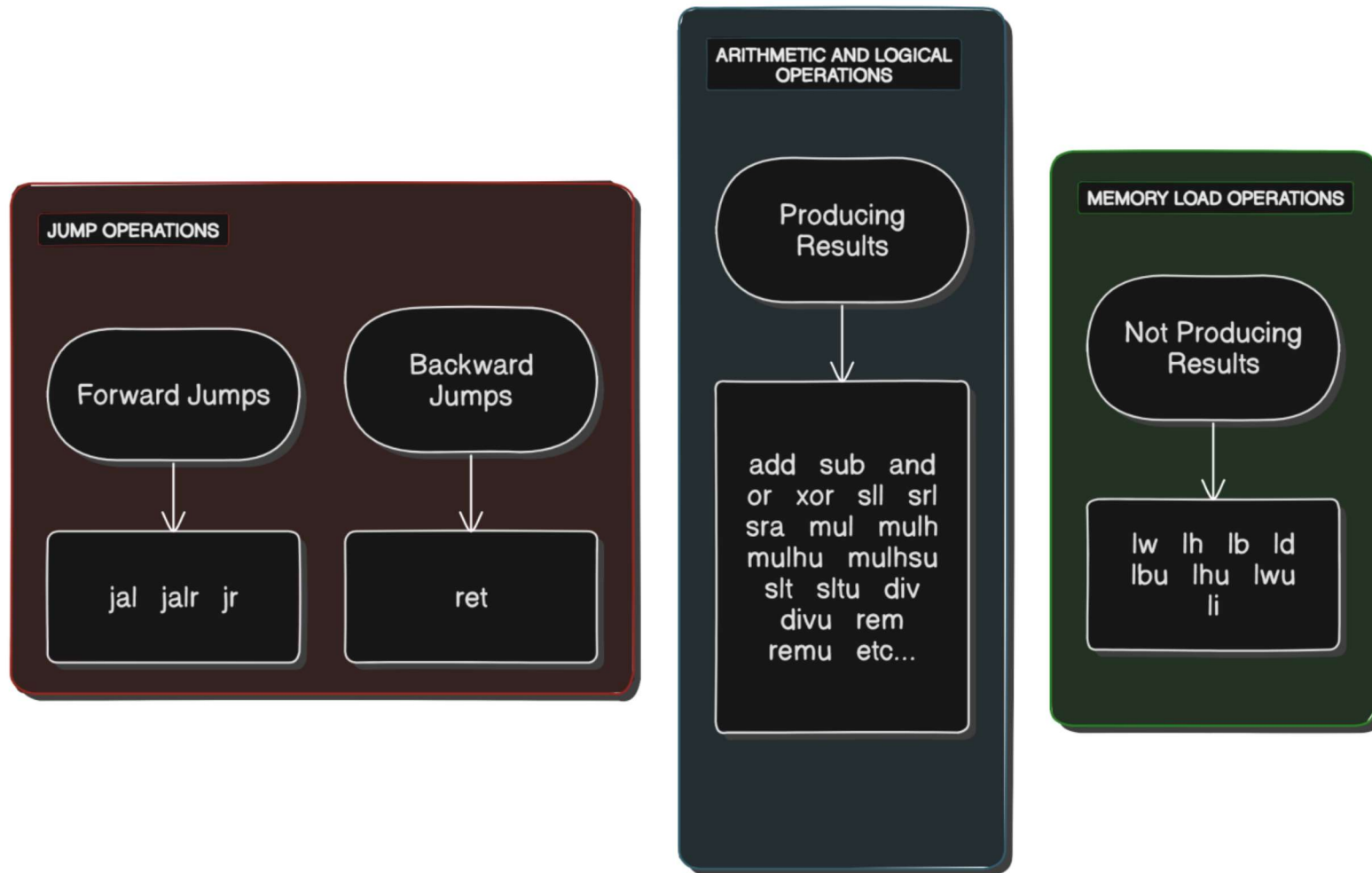
```
qemu-riscv32 -singlestep -d nochain,cpu main 2>main.log  
grep -o 'pc\s*[0-9a-fA-F]\{8\}' main.log >main_pc.log
```

```
pc      000100a8  
x0/zero 00000000 x1/ra      00000000 x2/sp      407fff30 x3/gp      00000000  
x4/tp    00000000 x5/t0      00000000 x6/t1      00000000 x7/t2      00000000  
x8/s0    00000000 x9/s1      00000000 x10/a0     00000000 x11/a1     00000000  
x12/a2   00000000 x13/a3     00000000 x14/a4     00000000 x15/a5     00000000  
x16/a6   00000000 x17/a7     00000000 x18/s2     00000000 x19/s3     00000000  
x20/s4   00000000 x21/s5     00000000 x22/s6     00000000 x23/s7     00000000  
x24/s8   00000000 x25/s9     00000000 x26/s10    00000000 x27/s11    00000000  
x28/t3   00000000 x29/t4     00000000 x30/t5     00000000 x31/t6     00000000  
pc      000100ac  
x0/zero 00000000 x1/ra      00000000 x2/sp      407fff30 x3/gp      000120a8  
x4/tp    00000000 x5/t0      00000000 x6/t1      00000000 x7/t2      00000000  
x8/s0    00000000 x9/s1      00000000 x10/a0     00000000 x11/a1     00000000  
x12/a2   00000000 x13/a3     00000000 x14/a4     00000000 x15/a5     00000000  
x16/a6   00000000 x17/a7     00000000 x18/s2     00000000 x19/s3     00000000  
x20/s4   00000000 x21/s5     00000000 x22/s6     00000000 x23/s7     00000000  
x24/s8   00000000 x25/s9     00000000 x26/s10    00000000 x27/s11    00000000  
x28/t3   00000000 x29/t4     00000000 x30/t5     00000000 x31/t6     00000000  
pc      000100b0  
x0/zero 00000000 x1/ra      00000000 x2/sp      407fff30 x3/gp      000120c0  
x4/tp    00000000 x5/t0      00000000 x6/t1      00000000 x7/t2      00000000  
x8/s0    00000000 x9/s1      00000000 x10/a0     00000000 x11/a1     00000000  
x12/a2   00000000 x13/a3     00000000 x14/a4     00000000 x15/a5     00000000  
x16/a6   00000000 x17/a7     00000000 x18/s2     00000000 x19/s3     00000000  
x20/s4   00000000 x21/s5     00000000 x22/s6     00000000 x23/s7     00000000  
x24/s8   00000000 x25/s9     00000000 x26/s10    00000000 x27/s11    00000000  
x28/t3   00000000 x29/t4     00000000 x30/t5     00000000 x31/t6     00000000  
pc      000100b4
```

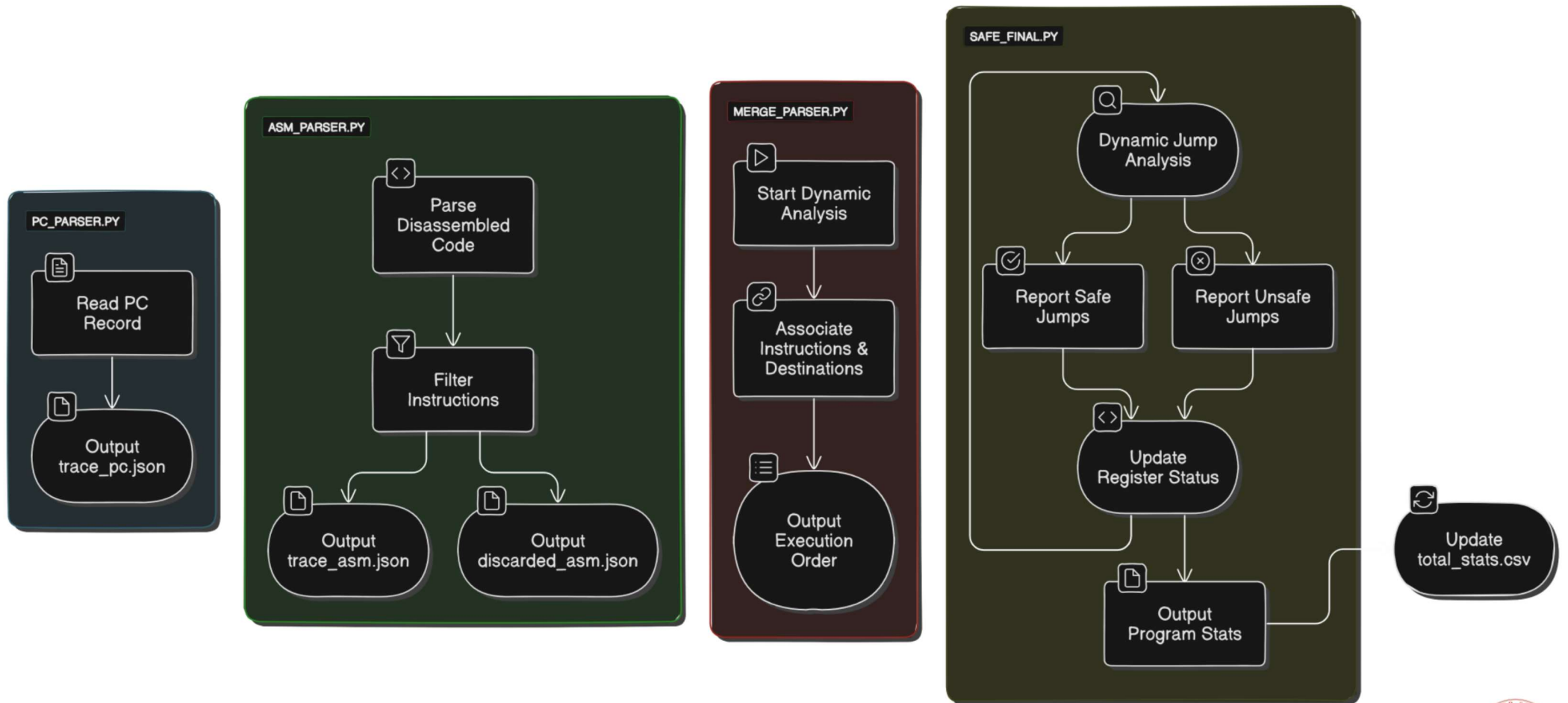
```
1 pc      000100a8  
2 pc      000100ac  
3 pc      000100b0  
4 pc      000100b4  
5 pc      000100b8  
6 pc      000100ba  
7 pc      000100bc  
8 pc      0001026a  
9 pc      0001026c  
10 pc     0001026e  
11 pc     00010272  
12 pc     00010276  
13 pc     000102f4  
14 pc     000102f8  
15 pc     000102fc  
16 pc     000102fe  
17 pc     00010300  
18 pc     000102c0  
19 pc     000102c4  
20 pc     000102c8  
21 pc     000102cc  
22 pc     000102d0  
23 pc     000102d4  
24 pc     000102d8  
25 pc     000102dc  
26 pc     000102e0  
27 pc     00010304  
28 pc     00010306  
29 pc     00010308
```



Instructions classification



Program Flow

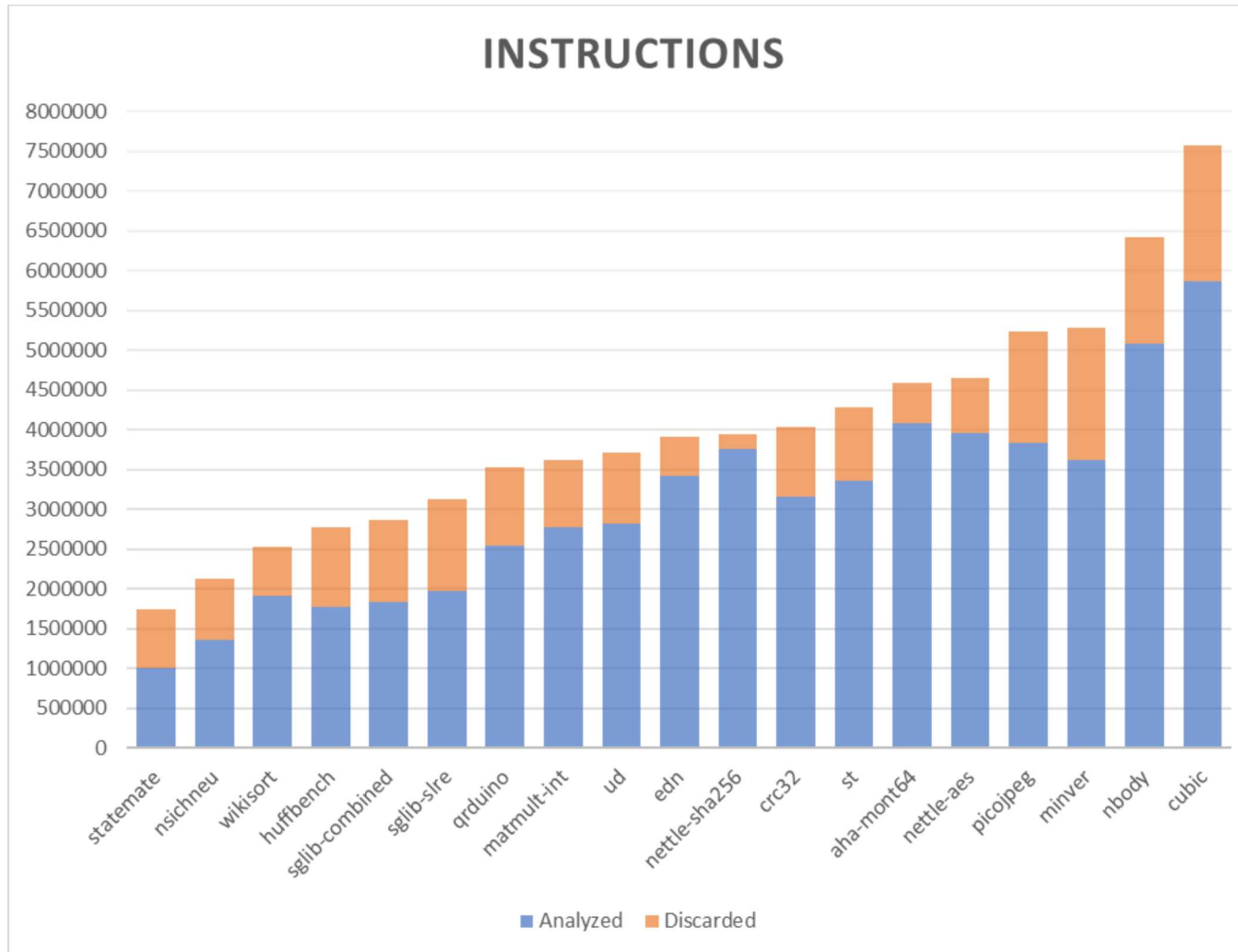




ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Results

Results

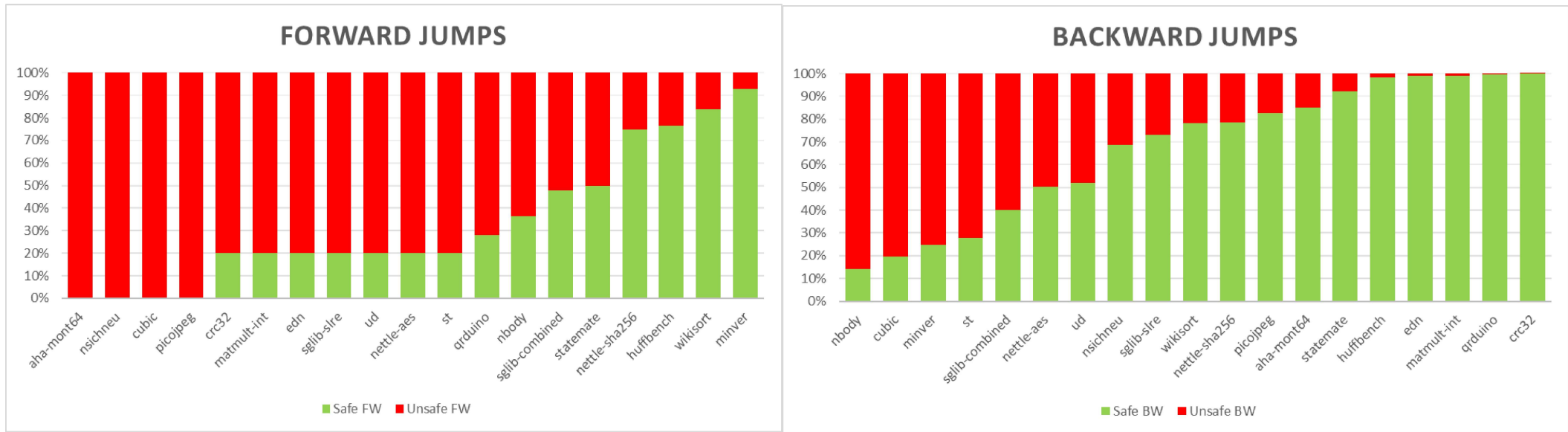


On average, 75% of instructions are analyzed and 25% are discarded:

- **nettle-sha256**: 95% of instructions are analyzed due to the need for many result-producing instructions
- **statemate**: Only 58% of instructions are analyzed, with a large portion being non-result-producing instructions



Results



- **Forward jumps:** On average, 94% of jumps are **unsafe**, and only 6% are **safe**.
- **Backward jumps:** The majority (94%) are **safe**.

The predominance of backward jumps is attributed to the frequent use of "*ret*" instructions (used to return from routines)



Results - *statemate*

```
Total jumps:      29498

Total fw jumps:   3935 (13.3%)
Safe fw:          1967 (50.0%)
Unsafe fw:        1968 (50.0%)
```

Balance between forward **safe**
and **unsafe** jumps

```
Enter the PC value:
112aa
The PC value '112aa' occurs 1965 times in 'statemate/merge_list.json'.
```

}, { "Index": 287, "PC": "112a2" }, { "Index": 288, "PC": "112a4" }, { "Index": 289, "PC": "112a8" }, { "Index": 290, "PC": "112aa" }, { "Index": 291, "PC": "112ca" }, { "Index": 292, "PC": "112ce" }, { "Index": 293, "PC": "112d2" }, {	1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481	00011274 <memset>: 11274: 433d 11276: 872a 11278: 02c37363 1127c: 00f77793 11280: efbdb 11282: e5ad 11284: ff067693 11288: 8a3d 1128a: 96ba 1128c: c30c 1128e: c34c 11290: c70c 11292: c74c 11294: 0741 11296: fed76be3 1129a: e211 1129c: 8082 1129e: 40c306b3 112a2: 068a 112a4: 00000297 112a8: 9696 112aa: 00a68067 112ae: 00b70723 112b2: 00b706a3 112b6: 00b70623 112ba: 00b705a3 112be: 00b70523 112c2: 00b704a3	li x6,15 mv x14,x10 bgeu x6,x12,1129e <memset+0x2a> andi x15,x14,15 bnez x15,112fe <memset+0x8a> bnez x11,112ec <memset+0x78> andi x13,x12,-16 andi x12,x12,15 add x13,x13,x14 sw x11,0(x14) sw x11,4(x14) sw x11,8(x14) sw x11,12(x14) addi x14,x14,16 bltu x14,x13,1128c <memset+0x18> bnez x12,1129e <memset+0x2a> ret sub x13,x6,x12 slli x13,x13,0x2 auipc x5,0x0 add x13,x13,x5 jr 10(x13) sb x11,14(x14) sb x11,13(x14) sb x11,12(x14) sb x11,11(x14) sb x11,10(x14) sb x11,9(x14)



Results – *statemate*

```
Enter the PC value:
1130a
The PC value '1130a' occurs 1966 times in 'statemate/merge_list.json'.
```

"Index": 12, "PC": "11280"	1471 1129e: 40c306b3 sub x13,x6,x12
{	1472 112a2: 068a slli x13,x13,0x2
"Index": 13, "PC": "112fe"	1473 112a4: 00000297 auipc x5,0x0
},	1474 112a8: 9696 add x13,x13,x5
{	1475 112aa: 00a68067 jr 10(x13)
"Index": 14, "PC": "11302"	1476 112ae: 00b70723 sb x11,14(x14)
},	1477 112b2: 00b706a3 sb x11,13(x14)
{	1478 112b6: 00b70623 sb x11,12(x14)
"Index": 15, "PC": "11306"	1479 112ba: 00b705a3 sb x11,11(x14)
},	1480 112be: 00b70523 sb x11,10(x14)
{	1481 112c2: 00b704a3 sb x11,9(x14)
"Index": 16, "PC": "11308"	1482 112c6: 00b70423 sb x11,8(x14)
},	1483 112ca: 00b703a3 sb x11,7(x14)
{	1484 112ce: 00b70323 sb x11,6(x14)
"Index": 17, "PC": "1130a"	1485 112d2: 00b702a3 sb x11,5(x14)
},	1486 112d6: 00b70223 sb x11,4(x14)
{	1487 112da: 00b701a3 sb x11,3(x14)
"Index": 18, "PC": "112ca"	1488 112de: 00b70123 sb x11,2(x14)
},	1489 112e2: 00b700a3 sb x11,1(x14)
{	1490 112e6: 00b70023 sb x11,0(x14)
"Index": 19, "PC": "112ce"	1491 112ea: 8082 ret
},	1492 112ec: 0ff5f593 zext.b x11,x11
{	1493 112f0: 00859693 slli x13,x11,0x8
"Index": 20, "PC": "112d2"	1494 112f4: 8dd5 or x11,x11,x13
},	1495 112f6: 01059693 slli x13,x11,0x10
{	1496 112fa: 8dd5 or x11,x11,x13
"Index": 21, "PC": "112d6"	1497 112fc: b761 j 11284 <memset+0x10>
},	1498 112fe: 00279693 slli x13,x15,0x2
{	1499 11302: 00000297 auipc x5,0x0
"Index": 22, "PC": "112d0"	1500 11306: 9696 add x13,x13,x5
},	1501 11308: 8286 mv x5,x1
{	1502 1130a: fa8680e7 jalr -88(x13)
"Index": 23, "PC": "112d4"	1503 1130e: 8096 mv x1,x5
},	1504 11310: 17c1 addi x15,x15,-16
{	1505 11312: 8f1d sub x14,x14,x15
"Index": 24, "PC": "112d8"	1506 11314: 963e add x12,x12,x15
},	1507 11316: f8c374e3 bgeu x6,x12,1129e <memset+0x2a>
{	1508 1131a: b7a5 j 11282 <memset+0xe>
"Index": 25, "PC": "112dc"	1509 1131e: 8096 mv x1,x5
},	1510 11322: 17c1 addi x15,x15,-16
{	1511 11324: 8f1d sub x14,x14,x15
"Index": 26, "PC": "112de"	1512 11326: 963e add x12,x12,x15
},	1513 11328: f8c374e3 bgeu x6,x12,1129e <memset+0x2a>
{	1514 1132a: b7a5 j 11282 <memset+0xe>
"Index": 27, "PC": "112e0"	1515 1132c: 8096 mv x1,x5
},	1516 11330: 17c1 addi x15,x15,-16
{	1517 11332: 8f1d sub x14,x14,x15
"Index": 28, "PC": "112e4"	1518 11334: 963e add x12,x12,x15
},	1519 11336: f8c374e3 bgeu x6,x12,1129e <memset+0x2a>
{	1520 11338: b7a5 j 11282 <memset+0xe>

The program's execution path depends on the outcome of branches

112aa and 1130a instruction s both use **x13** as their destination register



Results – Static vs Dynamic Analysis

```
statemate_dSnum.log
00011274 <memset>:
11302: 00000297      auipc x5,0x0
11306: 9696         add  x13,x13,x5
11308: 8286         mv  x5,x1
1130a: fa8680e7     jalr -88(x13)
```

```
statemate_dSnum.log
0001012a <frame_dummy>:
10130: 00000013     addi x10,x10,0x0
1013c: 00000317     auipc x6,0x0
10140: 00000067     jr  x0 # 0 <main-0x10074>
```

A static analysis would have reported **2 safe jumps** and **9 unsafe jumps**, not highlighting the safeness balance observed in dynamic analysis of program execution.



Results - Forward jumps

In *wikisort*, program the forward jumps turn out to be 84% **safe** compared to 16% **unsafe** forward jumps

```
105c8: cc2e      sw x11,24(x2)
105ca: c836      sw x13,16(x2)
105cc: ca3a      sw x14,20(x2)
105ce: c642      sw x16,12(x2)
105d0: 40d70c33  sub x24,x14,x13
105d4: 8abe      mv x21,x15
105d6: 5406      lw x8,96(x2)
105d8: 8b2a      mv x22,x10
105da: 8a46      mv x20,x17
105dc: 89b6      mv x19,x13
105de: 40f807b3  sub x15,x16,x15
105e2: 09864463  blt x12,x24,1066a <WikiMerge+0xc4>
105e6: 00369493  slli x9,x13,0x3
105ea: 003c1993  slli x19,x24,0x3
105ee: 94aa      add x9,x9,x10
105f0: 99a2      add x19,x19,x8
105f2: 02f05c63  blez x15,1062a <WikiMerge+0x84>
105f6: 03805a63  blez x24,1062a <WikiMerge+0x84>
105fa: 003a9913  slli x18,x21,0x3
105fe: 080e      slli x16,x16,0x3
10600: 992a      add x18,x18,x10
10602: 01050b33  add x22,x10,x16
10606: 4010      lw x12,0(x8)
10608: 4054      lw x13,4(x8)
1060a: 00092503  lw x10,0(x18)
1060e: 00492583  lw x11,4(x18)
10612: 04a1      addi x9,x9,8
10614: 9a02      jalr x20
10616: ed15      bnez x10,10652 <WikiMerge+0xac>
10618: 401c      lw x15,0(x8)
1061a: feF4ac23  sw x15,-8(x9)
```

```
102680      "PC": "1060e"
102681      },
102682      {
102683          "Index": 25671,
102684          "PC": "10612"
102685      },
102686      {
102687          "Index": 25672,
102688          "PC": "10614"
102689      },
102690      {
102691          "Index": 25673,
102692          "PC": "1014a"
102693      },
102694      {
102695          "Index": 25674,
102696          "PC": "1014c"
102697      },
102698      {
102699          "Index": 25675,
102700          "PC": "10150"
102701      },
102702      {
102703          "Index": 25676,
102704          "PC": "10152"
102705      },
102706      {
102707          "Index": 25677,
102708          "PC": "10616"
102709      },
102710      {
```

The *jalr* (forward safe jump case) is executed about 7000 times, and a total of 47000 *jalr* are executed in the program with a similar construct



Results - *Forward jumps*

In the *nsichneu* program the forward jumps turn out to be 100% unsafe.

Address	Disassembly	Comment
13be8: 40295913	srai x18,x18,0x2	
13bec: 00090963	beqz x18,13bfe <__libc_init_array+0x34>	
13bf0: 4481	li x9,0	
13bf2: 401c	lw x15,0(x8)	
13bf4: 0485	addi x9,x9,1	
13bf6: 0411	addi x8,x8,4	
13bf8: 9782	jalr x15	
13bfa: fe991ce3	bne x18,x9,13bf2 <__libc_init_array+0x28>	
13bfe: 00000417	auipc x8,0x0	
13c02: 40640413	addi x8,x8,1030 # 14004 <__init_array_start>	

There are standard constructs, such as the one above, that are repeated several times in the code and lead to an unsafe jump

Results - *Backward jumps*

In *nbody*, **unsafe** backward jumps represent 86% of the total backward jumps

```
10166: 033a1063      bne x20,x19,10186 <offset_momentum+0x4e>
1016a: 50b2          lw x1,44(x2)
1016c: 5422          lw x8,40(x2)
1016e: 5492          lw x9,36(x2)
10170: 5902          lw x18,32(x2)
10172: 49f2          lw x19,28(x2)
10174: 4a62          lw x20,24(x2)
10176: 4ad2          lw x21,20(x2)
10178: 4b42          lw x22,16(x2)
1017a: 4bb2          lw x23,12(x2)
1017c: 4c22          lw x24,8(x2)
1017e: 4c92          lw x25,4(x2)
10180: 4d02          lw x26,0(x2)
10182: 6145          addi x2,x2,48
10184: 8082          ret
10186: 0384ac03      lw x24,56(x9)
1018a: 03c4ac83      lw x25,60(x9)
1018e: 02000413      li x8,32
10192: 008487b3      add x15,x9,x8
10196: 4390          lw x12,0(x15)
10198: 43d4          lw x13,4(x15)
1019a: 8562          mv x10,x24

28990: {
28991:   "Index": 7248,
28992:   "PC": "10184"
28993: },
28994: {
28995:   "Index": 7249,
28996:   "PC": "103b4"
28997: },
28998: {
28999:   "Index": 7250,
29000:   "PC": "103b8"
29001: },
29002: {
29003:   "Index": 7251,
29004:   "PC": "103ba"
29005: },
29006: {
29007:   "Index": 7252,
29008:   "PC": "103bc"
29009: },
29010: {
29011:   "Index": 7253,
```

Register **x1** (*return address*) is modified with a load (unsafe operation) and before the execution of the "ret" instruction, x1 is loaded with potential unsafe content and the corresponding backward jump is unsafe



Results - *Backward jumps*

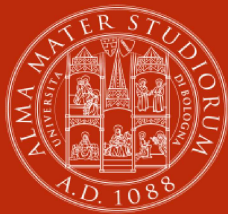
In *crc32*, backward **safe** jumps correspond to almost all backward jumps (99.9%)

```
000101bc <rand_beebs>:
 101bc: c3818793      addi x15,x3,-968 # 11cf8 <seed>
 101c0: 4388          lw x10,0(x15)
 101c2: 41c65737      lui x14,0x41c65
 101c6: e6d70713      addi x14,x14,-403 # 41c64e6d <__global_pointer$+0x41c
 101ca: 02e50533      mul x10,x10,x14
 101ce: 670d          lui x14,0x3
 101d0: 03970713      addi x14,x14,57 # 3039 <main-0xd03b>
 101d4: 953a          add x10,x10,x14
 101d6: 0506          slli x10,x10,0x1
 101d8: 8105          srli x10,x10,0x1
 101da: c388          sw x10,0(x15)
 101dc: 8141          srli x10,x10,0x10
 101de: 8082          ret

676      "PC": "101dc"
677      },
678      {
679      "Index": 170,
680      "PC": "101de"
681      },
682      {
683      "Index": 171,
684      "PC": "10146"
685      },
686      {
687      "Index": 172,
688      "PC": "1014a"
689      },
690      {
```

The instruction highlighted is executed many times (about 175000 out of a total of 175500 "ret") and the return address is never overwritten, leading to a backward safe jump





ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Conclusions

Conclusions

Reached objectives:

- Successfully compiled *Embench IoT* benchmark programs to retrieve the dynamic traces
- Successfully counted the number of insecure edges in executed RISC-V benchmark programs
- Results confirmed by patterns identified in the program trace
- Processed data stored in a CSV file, useful for further analysis
- First experience with Python code design

Future developments:

- Extend process to *Mibench* benchmark programs
- Refine the criteria chosen for defining safe edge and unsafe edge

