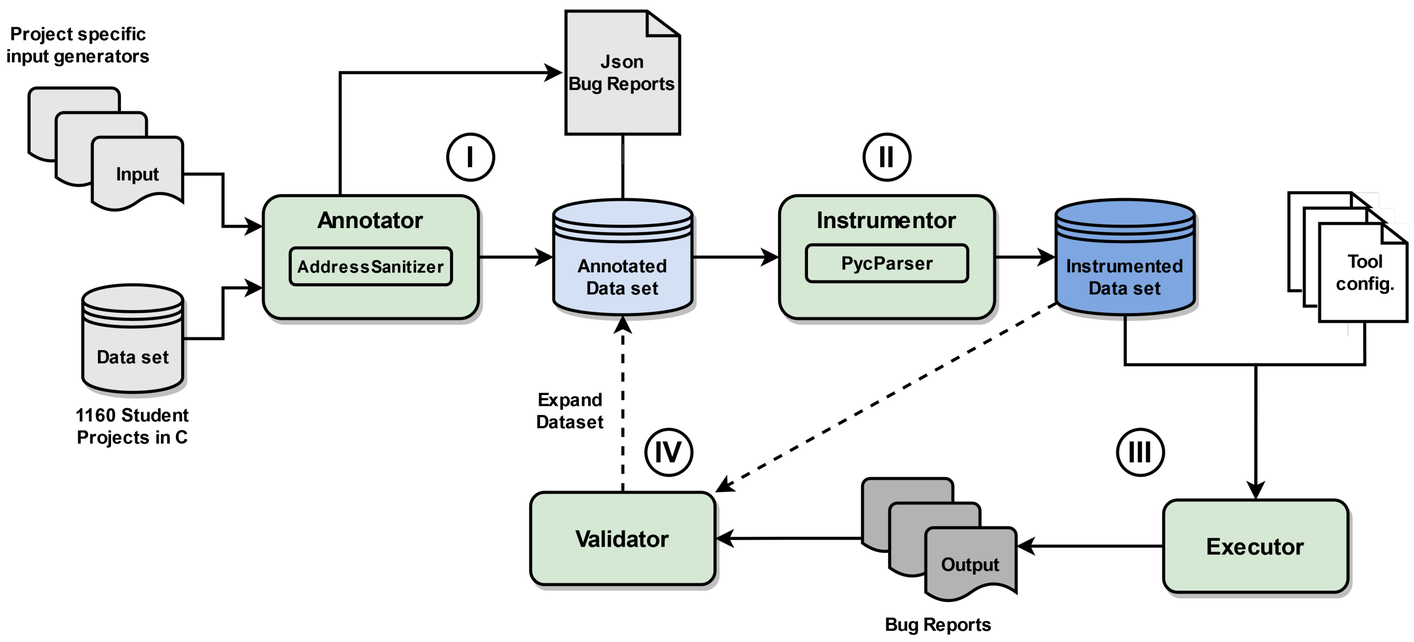
Empirical Study on Applying Program Analysis and Testing Tools to Student Code

Frederico Ramos, Filipe Marques, Pedro Adão, Nuno Santos, and José Fragoso Santos
INESC-ID & Instituto Superior Técnico, University of Lisbon

Overview



Abstract

We perform the first empirical study on the application of program analysis and software testing tools to student code. To conduct this study, we set up a large curated dataset consisting of 1,160 student projects implemented in the C programming language, totalling 405k LoC and averaging 349 LoC per project. These projects were anonymised and annotated with the exact locations where memory errors occur. We are in the process of evaluating the best-performing tools submitted to the SV-Comp 2022 competition against the constructed dataset. For this evaluation, we performed a non-trivial instrumentation of the dataset. Preliminary results indicate that these tools perform well and that they should play a more prominent role in undergraduate Computer Science curricula.

Annotated Dataset (I)

Original Dataset. The table below presents the benchmark suite characterisation. For each project, we show the number of lines of code of the sample project solution developed by the faculty (S_{LoC}), the number of student projects (n_{proj}), the total number of lines of code of the student projects (T_{LoC}), and the average number of lines of code per student project (Avg_{LoC}).

Project	S_{LoC}	n_{proj}	T_{LoC}	Avg_{LoC}
P1	256	398	140,349	352.64
P2	529	349	176,547	505.86
P3	166	84	17,697	210.68
P4	307	67	21,849	326.10
P5	416	117	17,145	146.54
P6	208	77	20,278	263.35
P7	100	27	4,048	149.93
P8	304	15	2,691	179.40
P9	204	15	2,399	159.93
P10	108	11	1,938	176.18
Total	2,598	1,160	404,941	349.09

Dataset Bugs. We considered the following types of memory bugs: heap-overflows (HO), stack-overflows (SO), data-overflows (DO), allocation-size-to-big (ASTB), segmentation-violation (SEGV), free-errors (FE), and other bugs (OTHER).

Project	HO	SO	DO	ASTB	SEGV	FE	OTHER
P1	1	117	13	0	47	0	6
P2	19	6	1	0	23	21	0
P3	175	38	1	64	154	7	1
P4	220	43	0	51	174	6	1
P5	143	78	10	91	180	10	2
P6	143	61	4	64	208	3	43
P7	40	13	0	22	45	2	1
P8	23	10	0	10	32	1	2
P9	5	1	0	0	1	0	0
P10	31	0	0	9	26	5	0
Total	800	367	29	311	890	55	56

To collect the bugs of each project, we: **(i)** generated 1,000 inputs using project-specific input generators; **(ii)** compiled and ran student solutions using *AddressSanitizer* on the generated inputs; and **(iii)** collected all reported errors.

Vulnerability Report. We anonymised and annotated the student projects: for each project, we created a JSON filed with a description of its memory bugs. For each bug, we stored: the bug type, the line number in which the bug occurs, the function in which the bug was triggered, and file name of the program. An example is given below:

```
[
  {
    "bug_type"  : "stack-overflow",
    "line"      : "121",
    "procedure" : "main",
    "file"      : "alunos/al011/p1.c",
    "witness"   : "random_20.in"
  },{
    "bug_type"  : "stack-buffer-overflow",
    "line"      : "147",
    "procedure" : "main",
    "file"      : "alunos/al001/p1.c",
    "witness"   : "random_60.in"
  }
]
```

Instrumentor (II)

Code Instrumentation. We instrumented the targeted C code, replacing calls to libc functions that read formatted input with tool-specific calls responsible for generating the appropriate symbolic values, as illustrated by the instrumentation of the *scanf* function below:

```
struct A { int value; char name[SIZE]; }
...
scanf("\%s:\%d", obj.name, &obj.value);
```

↓

```
for (int i = 0; i < SIZE; ++i)
    obj.name[i] = __VERIFIER_nondet_char();
obj.name[SIZE-1] = '\0';
obj.value = __VERIFIER_nondet_int();
```

Executor (III)

Tool Selection Criteria. We selected the evaluated tools according to the following criteria:

	Criteria	Tools
1.	5 best-performing tools in the *Cover-Error* Category of Test-Comp 2022.	FuSeBMC LibKluzzer VeriFuzz KLEE Symbiotic
2.	Winners of *Memsafety* from SV-Comp 2022.	Symbiotic CPAchecker
3.	Other static analysis tools	Infer & Pulse

If you would like us to consider your tool, come talk to us!

Validator (IV)

Bug Reports. We will extend the dataset with the bugs identified by the tools that do not report false positives. The job of the validator component is to confirm the bugs reported by the tested tools when they come with a concrete model and to extend the dataset with the confirmed bugs.

```
ktest file : 'test000007.ktest'
args       : ['Projects/1/preprocessed/1.bc',
↪  '-sym-stdin', '10' ]
num objects: 7
...
object 4: name: 'sym_int'
object 4: size: 4
object 4: data: b'\x00\x00\x00\x80'
object 4: hex : 0x00000080
object 4: int : -2147483648
object 4: uint: 2147483648
...
```

Preliminary Results

The table on the right presents the preliminary results obtained when running the analysed tools against our curated dataset. For each project, the table shows the total and average number of errors, e.g. Total/Average.

- The static analysis tools report ≈ 137% more bugs than symbolic execution tools — this is expected as these tools may also report false positives.
- Symbolic execution tools uncover ≈ 216% more bugs than the project-specific fuzzers.
- A final analysis of precision and recall for the analysed tools will be done after the bugs identified by symbolic execution tools are validated and integrated in the curated dataset.

Project	Random	Infer	Pulse	Symbiotic	KLEE
P1	184/1	1,456/4	1,504/4	1,595/5	1,303/3
P2	70/1	836/9	1,270/14	621/2	616/2
P3	440/5	390/5	780/9	235/3	365/4
P4	495/8	441/7	862/13	389/6	452/7
P5	514/4	467/4	858/7	348/3	493/4
P6	526/7	385/5	733/9	267/4	427/6
P7	123/5	112/4	214/9	70/3	151/6
P8	78/5	41/3	89/6	48/4	68/5
P9	7/1	106/7	150/10	10/1	11/1
P10	71/7	42/4	118/11	48/4	51/5
Total	2,508/4	4,276/5	6,579/9	3,631/4	3,936/4

Conclusions

We presented ongoing work on the first empirical study on the application of state-of-the-art program analysis and testing tools to student projects. So far, we have: **(i)** set up a curated dataset of anonymised student projects; **(ii)** annotated the dataset with memory error locations; and, **(iii)** ran KLEE, Symbiotic, Infer, and Pulse against our dataset. The results indicate that all tools perform well, uncovering more memory bugs than those present in the original dataset.

Acknoledgments. The authors were supported by Portuguese funds through Fundação para a Ciência e a Tecnologia (UIDB/50021/2020, INESC-ID multi-annual funding program) and projects INFOCOS (PTDC/CCI-COM/32378/2017) and DIVINA (CMU/TIC/0053/2021).