Background

Program bugs remain omnipresent. Today, there are many approaches to finding program bugs, with code reviews, unit tests, and program analysis tools being the most commonly used. While code reviews and unit tests are by their nature adapted to the program that they analyse, program analysis tools are typically more general.

In this project, you will develop or extend a static program analysis tool to detect memory errors and/or API contract errors (your choice). Your project will focus on analysing the GSTreamer framework (which is scheduled for inclusion in the WARA for Software) and may be adapted specifically to GSTreamer’s needs. You can find the project and its dependencies here:

- GSTreamer:
  [https://gstreamer.freedesktop.org/](https://gstreamer.freedesktop.org/)
- GSTreamer git packages:
  [https://gitlab.freedesktop.org/gstreamer](https://gitlab.freedesktop.org/gstreamer)
- glib, a library used by GSTreamer:
  [https://gitlab.gnome.org/GNOME/glib](https://gitlab.gnome.org/GNOME/glib)

Participants

- Baldvin Bern, Axis Communications AB
- Christoph Reichenbach, Lund University

Challenges to investigate

You will work with a list of known bugs in past versions of GSTreamer. You will focus on bugs that cause memory errors or API contract violations (e.g., in the GSTreamer pipeline system or the glib library).

1. RQ1 Which bugs from your list can the existing tools find? Which ones can in principle be detected statically?
2. RQ2 What systematic static program analyses can capture additional bugs not detected by the previous tools?
3. RQ3 How effective are these analyses at detecting bugs in code that are not part of the bug list that you explored for RQ1?
4. RQ4 How do your analysis extensions affect the rate of false positives?

Due to the limited amount of time, you will focus your efforts on a narrow class of bugs.
Deliverables

- A research report, 6-12 pages, in acmart sigplan format, that provides answers to the above research questions. After RQ2, you will select only one of the program analysis ideas that you have identified.

- Presentation of your results at the WASP Winter Conference, including demonstration videos: http://wasp-sweden.org/graduate-school/courses/wasp-project-course-fall-2019/

References

As resources, you may use any existing Open Source framework for static program analysis of C/C++ code. You must compare the performance of your system against the performance of these existing frameworks (which you can also uses as a basis for your own work):

- Clang Static Analyzer: https://clang-analyzer.llvm.org/
- CPPCheck: http://cppcheck.sourceforge.net/

You may use or introduce any auxiliary systems that you find useful, from custom preprocessors to Datalog processing engines to API contract description languages.

You may find the following techniques, tools, and libraries helpful in your analysis:

- Dataflow analysis:
  - IFDS/IDE framework: https://dl.acm.org/citation.cfm?id=199462
  - PhASAR library for LLVM:
    * github: https://github.com/secure-software-engineering/phasar

- Heap modelling:
  http://doi.acm.org/10.1145/2931098

- Datalog-based points-to analysis:
  - Soufflé Datalog engine: https://souffle-lang.github.io/docs/datalog/
  - DOOP (for Java):
    * https://plast-lab.github.io/doop-pldi15-tutorial/
    * http://smaragd.org/points-to-tutorial15.pdf
  - cclyzer (WIP DOOP for C++): https://github.com/plast-lab/cclyzer

Keywords

- Static program analysis
- C++
- Memory Safety
- Axis Communications AB
- Audio/Video stream processing
- WARA-SW