Project Milestone II
CS639A - Fall 2020

Baldip Singh Bijlani, Yugesh Ajit Kothari

Progress Brief

The goal of this project was to identify if we can leverage Angelic verification to suppress false positive bug reports in C/C++ programs that use EmscriptenSDK APIs. As an initial part of the project, we had set up the required toolchain. After experimenting with some examples, we decided to restrict the scope of this project by concerning ourselves with memory-safety bugs only. To find a suitable set of benchmarks programs which could benefit from angelic verification, we went through the entire emscripten API reference and identified what kinds of API are suitable for such verification. Thereafter, we gathered a small subset of programs, seeded bugs in a few of them to create a suite where it would be possible to potentially leverage Angelic verification. Next, we ran corral on these programs without any stubs for emscripten library calls to observe bugs reported. We present a brief overview of the benchmarks in this report.

Tasks Completed

As part of Milestone 2, we have been able to complete the following tasks :

1. Prepare benchmarks by taking some programs from the emscripten test suite and seeding a few of them with bugs. Most benchmark programs have false positives for baseline corral.
2. Run corral on benchmarks without stubs for library calls.
3. Prepare stubs of library calls for some of the benchmarks.
4. Build (setup) Angelic Verifier as a corral AddOn.

Tasks in Progress[1]

1. Preparing remaining stubs for library calls.
2. Run AV on the benchmarks.

Challenges

1. Restricted nature of benchmarks:

- Since emscripten is still relatively new (and as a consequence, new for us too), most of the benchmarks we have are either from or closely related to programs from Emscipten's test bench (which has close to 1400 programs). Going through

[1] Part of Adjusted Milestone2, yet to be completed.

these benchmarks and figuring out what programs would fit in the purview of Angelic Verification was extremely time intensive.

2. Including Callbacks:
- Callbacks in the web are typically related to event triggers like mouse click, key press etc. These usually have a calling context associated with them. In the case of WebAssembly, while these callbacks can be implemented in C, their context is usually in the wrapper JavaScript code. It took us a while to realize and try that we could include such programs by changing the entry-point for the verifier appropriately.

Benchmark Overview

Table shows a list of unique APIs in emscripten present in benchmark programs. This list is not exhaustive, only indicative of the features we explore for AV.

File	API Family	Nature of bug (alarm)
sdl2_swsurface.c	SDL	False invalid dereference
test_em_js.cpp	EM_JS(for writing javascript code in CPP files)	Missed memory leak
html5_webgl.c	html5_webgl	False invaild free
test_html5_mouse.c	html5	False invalid deref in callback function
idb_delete.cpp	fetch	False invalid dereference due to lack of API model
websocket.c	websocket	False invalid deref
test_gamepad.c	html5	False invalid deref in callback function
custom_em_js_test.c	EM_JS	False invalid deref

Work Remaining

Put together, this is the remaining work for completion of the project :

- Change AngelicVerifier to accept smack generated boogie.
- Run verification
    - Corral with stubs for library calls (for remaining programs)
    - Using AngelicVerifier
- Complete documentation for running benchmarks.