


CS486C – Senior Capstone Design in Computer Science

Project Description

Project Title: Automatic Standalone Unit Test Generation for DeepState	
Sponsor Information:  School of Informatics, Computing, and Cyber Systems	Dr. Alex Groce , Associate Professor, SICCS alex.groce@nau.edu Peter Goodman, Gustavo Greico Trail of Bits project staff School of Informatics, Computing, and Cyber Systems Northern Arizona University

Project Overview:

Software testing has never been more relevant than now; software is the life-blood of modern life, driving the thousands of processors in everything from refrigerators to cars to mobile phones. Since the first programmer wrote the first program, the fundamental problem of coding has been with us: the entities commonly known as “bugs.” Bugs can range in behavior from crashing a system completely to producing subtly wrong (and perhaps dangerous) results. Because humans are bad at guessing where subtle bugs lie in software, and malicious actors, including those employed by nation states, are very good at finding subtle bugs, tools that automatically generate high-quality *tests* to find even the most obscure bugs are increasingly important. Unfortunately, most such tools are not designed for use by the average developer, who is only used to writing manual unit tests.

In our lab, in cooperation with the security company Trail of Bits, we have developed a software testing framework called DeepState (<https://github.com/trailofbits/deepstate>) that is a C and C++ unit testing framework modeled on Google's GoogleTest (<https://github.com/google/googletest>). Where most unit test frameworks focus on allowing users to execute completely manually written tests, where all values are well-defined (e.g., "foo(10); bar(5); assert (baz() == 0);"), DeepState is focused on allowing the definition of unit tests where some values are chosen by an automated test generation tool. A DeepState test is of the more general form "X = Choose_Int(); Y = Choose_Int(); foo(X); bar(Y); assert(baz() == correctValue(X, Y));". X and Y can be chosen by a "back-end" tool, possibly a fuzzer (generator of random tests) or a symbolic execution tool. DeepState supports saving, shrinking, and replaying tests generated by these tools.

The Problem

While DeepState is very useful for *producing* tests, there is some complexity and overhead involved, so many projects will prefer not to depend on such a complex system for their core unit testing. Instead, one could envision careful configuration of DeepState to generate an optimal testing regime for a particular class of software... and then exporting those generated

test suites so that they could be run as “standalone” unit testing regimes as many times as desired in future. Therefore, a long term goal of DeepState is to support "exporting" individual tests from DeepState's native format (a binary buffer containing the actual bits of X, Y, etc.) to standalone unit tests that use a more traditional unit testing framework such as GoogleTest. That is, developers would like to take a test of the form "X = Choose_Int(); Y = Choose_Int(); foo(X); bar(Y); assert(baz() == correctValue(X, Y));" plus concrete values for X and Y, and produce a test of the form: "foo(X); bar(Y); assert (baz() == 0);". The generated test should not require linking to or including any DeepState code.

Solution Details:

We envision a **command line tool** that, given as inputs 1) a DeepState harness file (C++ code, perhaps written in a certain style and with significant restrictions on the way code is constructed) foo.cpp and 2) a test (a binary file containing input bytes) foo.test, will produce 3) a C++ file, foo_standalone.cpp, similar to the original DeepState harness, except that rather than calling DeepState to produce values for use in the test, it hard-codes the values from the file foo.test as the test values. The file foo_standalone.cpp will not #include <deepstate.hpp>.

An early sketch of a possible solution is present in DeepState PR #209 (<https://github.com/trailofbits/deepstate/pull/209>) which resolves DeepState issue #208 (<https://github.com/trailofbits/deepstate/issues/208>).

- Implement a usable, effective way for developers to get standalone C++ code that allows using tests generated by DeepState without invoking DeepState itself.
- This task is accomplished by processing the test harness code and the test case (and running the test case using DeepState) in order to produce a new unit test file.
- An initial solution should finish the approach taken in PR#209, and extend it to more types of input values, and make the command line tool more robust. This approach takes a harness, produces a version that *prints a subset of its own code, populating the code with values from a test*, then executes the new version of the code. Currently, most of the work is left to the user, and only a very small set of types (basically, ints) is supported.
- A *minimal* working solution will support at least creation of simple non-recursive, possibly nested structs and code for all low-level types DeepState can generate, including strings, as well as capturing decisions of which function to call or selection from a finite set of values.
- A *good* solution will handle creation of pointer structures based on choices made by a DeepState harness.
- Support for C code, without requiring any C++ features, is a valuable extension that would see wide industry use.
- Provide documentation for users of how the approach works, and a working example of integrating it into a real-world complex system's testing.

Impacts:

DeepState was made public in early 2018, and now has over 400 GitHub stars. It is a focus of ongoing research & development at Trail of Bits (a leading cybersecurity company), and users

at Google, Facebook, Bloomberg and other high-profile companies have expressed interest in using DeepState. Google has considered eventually rolling DeepState into Google Test, one of the most widely used C/C++ unit testing frameworks. Making it possible to integrate DeepState into existing testing, especially Continuous Integration (CI) more easily would be a major advance in the usefulness of DeepState. This functionality request originated on the Empire Hacking Slack channel for DeepState, from a real developer using DeepState.

Knowledge, skills, and expertise required for this project:

- C and C++ coding skills: DeepState is to a large extent low-level systems software, and the tests to be generated will be C/C++ code
- Python coding skills: DeepState's tooling (including the prototype of a small subset of the desired functionality implemented in PR 209) is written in Python
- Knowledge of unit testing frameworks and complex build systems is a major plus; development of non web/UI-focused code in a Unix environment
- Compiler/code generation knowledge may be useful, but can be picked up during the project

Equipment Requirements:

- There are no unusual equipment requirements. DeepState provides a Docker solution, so almost any reasonable desktop/laptop machines will work, using any standard operating system (Linux, macOS, or Windows). For purposes of this project, DeepState can also work natively in macOS or Linux.

Software and Other Deliverables:

- Complete implementation of functionality described above, as either a continuation of DeepState PR #209, or another PR or series of PRs to the DeepState repository
- Documentation of the approach, pitfalls, and usage
- Standalone GitHub repository showing the approach in use, with documentation so developers can examine the workflow of moving from DeepState generation to standalone tests that can be added to CI and existing-framework unit tests