Our goal is to discover functions with the same behavior despite differences in their interfaces. To this end, we present a technique we call adaptor synthesis that determines whether the behavior of one function can be made to match the behavior of another function by appropriately modifying its arguments. Consider functions f1 and f2 below:

int f1(int x, unsigned y) {
    if (x < 1) y += (y % 2);
    return x + d = (a & b);
}

For any integer x and unsigned integer y, f2(y, 1, x) will return the same value as f1(x,y). We call the function that maps (x,y) to (y,1,x) an adaptor. With this particular adaptor, we can consider f2 to be semantically equivalent to f1. We write this relationship as f1 \rightleftharpoons f2.

Applications

Security

If we allow for semantic equivalence between functions that have different error behaviors, then we can use adaptor synthesis to find different versions of a function with and without certain bugs. Adaptor synthesis can also be used to find different versions of a function with other desirable properties, such as efficiency or clarity.

Example: Adaptor synthesis can find that a call to OpenSSL’s BN_hex2bn function, which has a null dereference/heap corruption bug (CVE-2016-0797), can be replaced with a call to mbedtls’s mbedtls_mpi_read_string.

The CEGIS search is restricted to a finite family of adaptors. One family of adaptors we support allows for an argument of f2 to be replaced by (1) an argument of f1, (2) a constant value, or (3) a type conversion applied to an argument of f1. We have also experimented with adaptors that can replace arguments with the string length of a pointer argument or a bounded depth arithmetical expression and adaptors that can convert between different struct arguments. We also support simple adaptations of return values.

Algorithm

We use counterexample guided inductive synthesis (CEGIS) to search for an adaptor that maps the arguments of f1 to the arguments of f2 (and the return value of f2 to the return value of f1) in such a way that the behavior of the two functions match. Our specification for synthesis is the behavior of f1 and we define counterexamples to be inputs on which the behavior of f1 and f2 differ with a given adaptor.

We implement counterexample guided inductive synthesis (CEGIS) to search for an adaptor that maps the arguments of f1 to the arguments of f2 (and the return value of f2 to the return value of f1) in such a way that the behavior of the two functions match. Our specification for synthesis is the behavior of f1 and we define counterexamples to be inputs on which the behavior of f1 and f2 differ with a given adaptor.

Evaluation

As a large-scale evaluation, we ran our adaptor synthesis tool on 13,130 function pairs from the system C library (eglibc 2.19). Using a family of adaptors allowing argument substitution and type conversion, we found 8909 pairs to be inequivalent and 383 pairs to be equivalent. We also had 2989 timeouts and 649 crashes.

Conclusions and Future Work

Our results confirm that several instances of adaptably equivalent binary functions exist in real-world code, and suggest that these functions can be used to construct cleaner, less buggy, more efficient programs. Some ideas for future work include:

• automatically generate binary code for adaptor functions
• experiment with other symbolic representations of adaptors
• add support for additional adaptor families (e.g. floating point values)
• infer preconditions in order to find adaptors that make functions equivalent provided that their preconditions are satisfied

Acknowledgements

This research was completed, in part, with support from the Defense Advanced Research Projects Agency (DARPA) under contract FA8750-15-C-0110 and, in part, with support from the National Science Foundation under grant 1563930. We acknowledge the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing computing resources for large scale evaluation experiments.