Tutorial: DeepState: Bringing Vulnerability Detection Tools into the Development Cycle

Peter Goodman
Trail of Bits, Inc.
peter@trailofbits.com

Gustavo Grieco
Trail of Bits, Inc.
gustavo.grieco@trailofbits.com

Alex Groce
School of Informatics, Computing & Cyber Systems
Northern Arizona University
alex.groce@nau.edu

***Abstract*—Traditionally, methods such as binary analysis, symbolic execution, and fuzzing have been used in a context that is strongly geared towards discovering existing vulnerabilities, rather than use in the development cycle to prevent vulnerabilities from arising. Unit testing, in contrast, is firmly in place as part of the development cycle, but is usually very limited in its ability to explore "deep" paths in a system, or expose completely un-anticipated aspects of system behavior. Incorporating the tools used for vulnerability discovery into the development cycle requires large expansion in the expertise that developers must possess, and significant changes in their practices. DeepState is an open-source tool that provides a Google Test-like API to give C and C++ developers push-button access to symbolic execution engines, such as Manticore and angr, and fuzzers, such as Dr. Fuzz. Rather than learning multiple complex tools, developers can learn one (familiar) interface for defining a test harness, and can use tools built to find security vulnerabilities to automatically generate more powerful unit tests for software, in an approach that merges traditional unit testing, security analysis methods, and property-based testing. This tutorial will show how to use DeepState in development, including to produce complex library and API tests, and how to take advantage of both the ability to easily apply multiple security-oriented back-ends for test generation during development and the novel strategies for improving back-end performance provided by DeepState.**

I. Topic

This tutorial is intended to allow attendees familiar with the idea of unit testing and the basic concepts of fuzzing and symbolic execution/binary analysis to use the DeepState [4] tool. As discussed in the abstract, most fuzzing and symbolic execution tools are primarily aimed at vulnerability detection; many are also quite difficult to use properly, especially binary analysis tools. DeepState, available at https://github.com/trailofbits/deepstate, provides the test generation power of such tools, but allows it to be accessed via the familiar approach of writing a (parameterized [7] and generalized) unit test. DeepState allows developers a smooth transition from the unit testing they are already familiar with and practice to producing more aggressive, security-analysis-quality tests during development. It also makes it convenient to produce effective tests of libraries and APIs, and allows easy expression of complex correctness or security properties that are hard to incorporate into more traditional fuzzing or binary analysis. In sum, DeepState allows a nice mix of unit testing, property-based testing [3], and vulnerability analysis. The aptness of this approach for SecDev should be extremely clear.

II. Tutorial Format

The tutorial will start with a short, mostly theoretical, section introducing fundamental concepts of binary analysis. In particular, we will outline the advantages and challenges of symbolic execution and fuzzing for program testing, and (briefly) demonstrate the challenges of using popular binary analysis tools, and the need for a completely separate workflow to apply fuzzing and symbolic execution. During this phase we will also introduce Manticore [5], angr [6], and Dr. Fuzz [2], the back-ends for the rest of the tutorial.

The remainder of the tutorial will be heavily interactive, in that we expect audience members to bring along laptops, and follow along with the presentation, including offering suggestions as to directions to explore. The audience will be guided through installation of the tool at the beginning of the tutorial, before a brief overview of the purpose and architecture of DeepState. After this, the remainder of the 180 minute tutorial will be devoted to learning-through-doing, by working through complete examples using DeepState.

The first example will be a relatively simple testing problem of generating integer inputs satisfying a mathematical constraint, in order to familiarize the audience with the basic syntax and workflow of DeepState (and to trouble-shoot their installations), and how to use different back-ends, including switching from symbolic execution to fuzzing. Audience members familiar with GoogleTest [1] will also see how DeepState provides an extension of that testing API.

The second example will be a problem of producing the inputs to a function with an actual security vulnerability, taking string and integer inputs. Small modifications to this problem will make clear the value of being able to express properties more complex than a simple crash easily, and to switch back-ends when a problem is easy for fuzzing to solve, but difficult for symbolic execution, or vice-versa.

The final, and most extensive, example will be using DeepState to write a simple test harness to find faults in a file system implementation. This will incorporate many of DeepState's more advanced features, and demonstrate how to control the generation process, improve the efficiency of test generation when necessary via DeepState's provided strategies, troubleshoot back-end performance issues, and use the sophisticated logging provided by DeepState. Finally, we will discuss integrating DeepState usage into existing code-

bases, and building on existing unit tests, using non-DeepState unit tests for the file system.

During all three explorations, we will encourage audience members to suggest their own properties to check, and ask questions about different ways to express tests and properties. There will also be periods of the talk where we encourage the audience to all modify their test harnesses in slightly different ways, and report back how this changes the results of testing using different back-ends or settings.

After attending the tutorial, audience members who have brought along a laptop will have (1) a working installation of DeepState and (2) modified versions of the tutorial example projects, including test output directories. In order to handle platform and installation issues we will make a VM available for use during the tutorial.

While the exact materials that will be used in the tutorial are not as yet available, browsing the DeepState GitHub repository (https://github.com/trailofbits/deepstate), reading the DeepState paper presented at the NDSS 2018 Binary Analysis Workshop (https://www.cefns.nau.edu/~adg326/bar18.pdf) and examining the in-progress file system test harness (https://github.com/pgoodman/testfs) will provide a good overview of the kind of content to be expected, in more detail.

III. Expected Audience and Learning Outcomes

A. Audience

Audience members should be basically familiar with C and C++ programming, but heavy C++-specific expertise is not required. Similarly, a basic understanding of, and at least slight experience with, the idea of unit testing is expected. Audience members should at least be aware of the concepts of fuzzing and binary/symbolic analysis for security vulnerability detection, but do not need to be current users of such tools (in fact, this is part of the purpose of DeepState: not to require deep expertise to use these tools). Familiarity with GoogleTest [1] specifically, or deeper knowledge of fuzzing/symbolic execution may increase the final expertise the audience achieves with DeepState, but is not required. We expect, in practice, a mix of security-conscious developers interested in using DeepState during normal development to improve their testing and reduce vulnerabilities, industrial security experts interested in the convenience and expressive power of DeepState for more traditional vulnerability-analysis tasks, and security researchers interested in DeepState for benchmarking security analysis back-ends or as an implementation platform for novel test-generation strategies.

B. Learning Outcomes

At the end of the tutorial, audience members should know:

1) the basic concepts of symbolic execution and fuzzing
2) the basic concepts of unit test frameworks
3) the purpose and basic idea of binary analysis frameworks such as Manticore and angr
4) the purpose and basic idea of a fuzzer such as Dr. Fuzz
5) how to install the DeepState tool
6) the primary purposes, basic design, and overall architecture of DeepState
7) how to convert a traditional fixed-input unit test into a parameterized unit test with more generalized assertions
8) how to combine vulnerability (crash/memory access violation) detection with property-driven functional testing
9) how to express various input generation tasks and properties in the DeepState API
10) how to generate concrete tests using multiple back-ends, including symbolic execution tools and fuzzers
11) how to play back DeepState-generated tests and use them in regression testing
12) how to express complex library-call sequence tests succinctly and efficiently using DeepState's special constructs for non-deterministic choice of values or code
13) how to use "strategies" to increase the efficiency and/or power of test generation back-ends
14) how to integrate DeepState use into existing code-bases
15) the value of combining traditional unit testing, property-driven testing, and vulnerability analysis.

IV. Prior Tutorials and Talks

Peter Goodman has given talks on cybersecurity at COUNTERMEASURE in 2016 and 2018 and presented a poster on sampling for data races at the LLVM developers' meeting in 2015. Gustavo Grieco has presented talks on his research at multiple academic conferences and prepared workshops on vulnerability discovery for well-known practically-oriented security conferences such as Hack In the Box, DefCamp, and Ekoparty. He is also interested in functional programming and has given talks at conferences such as C○mp○se. Alex Groce has given dozens of talks on software verification, testing, and analysis, at ICSE, ASE, FSE, ISSTA, and other top software engineering and testing conferences. He has organized multiple workshops on dynamic and static analysis (WODA, SPIN, and CREST) and the 2015 ISSTA Doctoral Symposium, and taught large university lecture classes on using such tools, in addition to tutorials for NASA/JPL engineers. Audiences for the proposers' talks have varied from small workshop settings of 10-20 participants to main sessions at large conferences with hundreds of attendees.

References

[1] "Google Test," https://github.com/google/googletest, 2008.

[2] "Dr. Fuzz: Dynamic fuzz testing extension," http://drmemory.org/docs/page_drfuzz.html, 2015.

[3] K. Claessen and J. Hughes, "QuickCheck: a lightweight tool for random testing of Haskell programs," in *International Conference on Functional Programming*, 2000, pp. 268–279.

[4] P. Goodman and A. Groce, "DeepState: Symbolic unit testing for C and C++," in *NDSS Workshop on Binary Analysis Research*, 2018.

[5] M. Mossberg, https://blog.trailofbits.com/2017/04/27/manticore-symbolic-execution-for-humans/, April 2017.

[6] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," in *Network and Distributed System Security Symposium*, 2016.

[7] N. Tillmann and W. Schulte, "Parameterized unit tests," in *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2005, pp. 253–262.