yFuzz for Effective Communication Protocol Fuzzing

Kailash Gogineni, Yongsheng Mei, Tian Lan, and Guru Venkataramani

June 17, 2021

1 Overview

1.1 Background and motivation

Network protocols are often bloated by redundant program features/functionalities due to changing user requirements and various environments. More code brings more vulnerabilities, which makes network protocols one of the top victims of cyber attacks. While coverage-based greybox fuzzing has gained great success in the field of vulnerability detection due to its simplicity and efficiency, it could become less powerful when applied directly to protocol fuzzing due to the unique challenges of protocol fuzzing. In particular, 1. The implementation of protocols usually involves multiple program binaries, i.e., multiple fuzzing entries; 2. The communication among multiple ends contains more than one packet, which are not necessarily dependent upon each other, i.e., fuzzing single (usually the first) packet can only achieve extremely limited code coverage.

To this end, we propose yFuzz, a stateful, yield-drive protocol fuzzer, to address the above-mentioned limitations through effective protocol fuzzing and to enhance the security of network protocols. yFuzz is able to explore the code related to different protocol states. It incorporates a stateful fuzzer (which contains a state switching engine) together with a multi-state forkserver (which enables multi-state program forking) to consistently and flexibly fuzz different states of an compiler-instrumented protocol program. Our experimental results on OpenSSL show that yFuzz achieves an improvement of 1.73X code coverage and 2X unique crashes when comparing against fuzzing the first packet during a protocol handshake.

1.2 Features and Capabilities

We further design a state-aware fuzzing tool, yFuzz, which intelligently fuzzes different packets according to different protocol states to improve the code coverage and find the vulnerabilities reside deep in the protocol. In particular, yFuzz create a test program involving both server and client, interface the test program with yFuzz to read/write packets through file I/O, instruments the testing program during compilation to insert fuzzing points according to protocol states as desired, and execute the program for stateful fuzzing. During runtime, yFuzz can checkpoint and restore various fuzzing states and perform state transition with different strategies.

1.3 Usage Scenarios

yFuzz can be used by network administrators to analyze program features with the goal of creating individualized communication protocol implementations that achieves improved diversity and security. Our program fuzzing tool can also be used to perform program testing before the program is released.

1.4 Dependencies and limitations

The fuzzing module implementation in yFuzz is based on modules developed for AFL (American Fuzzy Lop).

2 Description of Demos

The demo shows the process of setting up yFuzz for a test program and performing stateful fuzzing to improve the code coverage of protocol fuzzing. This demo will

- Begin with a test program involving both server and client of a target protocol.
- Interface the test program with yFuzz to read/write packets through file I/O, allowing yFuzz to mutate the packets according to the desired fuzzing strategy and configurations.
- Show the instrumentation to the target program, so that multiple fuzzing points can be inserted and the resulting code coverage monitored/collected.
- Then we will show how to use state forking and roll-back mechanism in yFuzz to create different fuzzing strategies, e.g., greedy, drill-through, and onion-peeling. We will demonstrate how to set the values of parameters that are critical for the state transitions of fuzzing.
- We will also demonstrate how to interpret and analyze the runtime statistics of stateful fuzzing.

3 Description of Hands-on exercise

The hands-on exercise will provide OpenSSL and possibly another communication protocol as an example, to use yFuzz to explore different protocol states, fuzz the test program, and improve the code coverage of protocol fuzzing using different fuzzing strategies.

In the program fuzzing exercise, the attendees have the following tasks: I. prepare a test program involving both server and client of the target protocol. II. configure the test program with yFuzz to read/write packets through file I/O. III. setting up the testing program for fuzzing, which includes identifying protocol states, inserting fuzzing points and initializing shared memory; IV. setting up the parameters for fuzzing, which includes timeout value, memory limit and conditions for fuzzing state transitions; V. observing the fuzzing process and analyzing the runtime statistics. VI. analyzing the results of fuzzing, which includes the code coverage, statistics of input mutations and fuzzing state transitions.

4 Learning-module development plan

- The demo and exercises will be developed by the end of September.
- The demo will be packaged into VMware virtual machines.
- The source code of our project is available here: https://github.com/yuroc0598/ProtocolCustomization