yFuzz Initial Evaluation
PJR Corp
Evaluator Name

22 December 2020

This brief is Controlled Unclassified Information (CUI) with Distribution Statement D. Distribution authorized to Department of Defense (DoD) and U.S. DoD contractors only.

Overview

yFuzz is a framework that provides stateful protocol fuzzing

- Expands on existing "American Fuzzy Lop" (AFL) project by adding ability to introduce fuzzed input at multiple stages of processing the protocol under test
- Leverages AFL's yield-driven results to focus on input that generates more interesting results
- Requires instrumentation of the program/protocol under test
- Research from George Washington University
  - Professors/PI –Tian Lan, Guru Venkataramani
  - Students – Kailash Gogineni, Yongsheng Mei, Yurong Chen

Initial Evaluation

Impressions

- The increase in protocol testing capability to AFL is extensive
- Adding instrumentation to a test program requires some knowledge fuzzing processes and yFuzz (learning curve)
- Supporting a test program that exercises both client/server portions of protocol can be non-trivial

Initial Evaluation

Environment Setup:

- **Time Spent:** 1/2 day
- Reproducing test environment from provided VM was straightforward
- OS used: Ubuntu 18.04.4
- yFuzz uses similar build instructions as AFL
    - yFuzz is based on AFL release v2.52b (Nov. 2017), latest AFL version on github is v2.57b (July 2020) <Researcher confirmation needed>

Initial Evaluation (Cont.)

Using yFuzz:

- Test program details
  - Test program code must exercise both client/server portions of the protocol under test
  - yFuzz specific Instrumentation must be added to test program
    - User must find and decide where in the test program source code to add macro "__AFL_INIT()"
      - Macro creates a forking point where yFuzz's mutated input is fed into protocol
    - User must add logic to the test program to interface with yFuzz
      - Synchronize state information with yFuzz via specific shared memory location
      - Read in mutated packet and feed to protocol at relevant step
  - Any relevant libraries must be statically linked
  - Test program and relevant libraries must be compiled with yFuzz’s (ALF) compiler (e.g., afl-clang-fast, afl-clang-fast++)
    - Address sanitizer feature must be enabled which can significantly increase memory requirements (600-800MB for 32-bit, up to 20Tb for 64-bit)

Initial Evaluation (Cont.)

Using yFuzz:

- **Time Spent: 3 days**
- Once test program is compiled, yFuzz takes the program as a parameter
- yFuzz will run continuously mutating/fuzzing packets at instrumented points searching for crashes and hangs
  - Recommended running time: hours to weeks (typically ~24h)
  - Longer runtime has diminishing returns
  - Coverage metric provided
- A Packet that causes a crash or hang is stored in a directory named according to location within the protocol
- After a crash/hang is identified, the user takes the packet and feeds it back into the test program to debug the issue

Estimation of Work

- Bring/Keep up to date with AFL versions – **1-3 Months**
  - yFuzz is currently not too behind latest AFL release
  - Involves ongoing effort to keep up to date
- Bug fixes and stability updates – **3-6 Months**
  - AFL Text UI bugs
    - UI corruption over time
    - "Number of queued packets to test" UI text field alignment issue
    - Potential display issue for protocols with high number of states
  - [Address various Issues/Todos in README file]
- Create and/or update documentation – **0.5 – 1 Month**
  - Documentation should help users create a test program, use yFuzz, and interpret results
  - Provide clear example test program (openssl/selftest)

Estimation of Work

- Improve yFuzz packet logistics – **3-6 Months**
  - Improve integration of AFL's single "input" packet fuzzing and yFuzz's multi-packet tracking
  - Streamline handling of test program input (fuzzed packets)
- Instrumentation API – **1-3 Months**
  - Create API to minimize need for user to be familiar with yFuzz internal logic.
  - Abstract away internal concepts such as fork-state, protocol state, and which packet to be read/writen
- Compatibility updates to related tools – **3-6 Months**
  - Updating related AFL tools to be compatible with new yFuzz functionality
    - afl-plot (visualization tool), afl-showmap (single run basic block reporting tool), etc...