AFL++ Fuzzing Modes Evaluation Report

Introduction

This report provides an evaluation of fuzzing the **Fuzzgoat** application with **AFL++** in various modes: LLVM instrumentation, GCC instrumentation, and QEMU mode (binary-only fuzzing). The purpose is to compare the effectiveness of these modes in terms of execution speed, code coverage, and crash discovery.

Methodology

I conducted fuzzing sessions using AFL++ on the Fuzzgoat application, utilizing three different approaches:

Target Application: Fuzzgoat fuzzing benchmark.

Fuzzing with AFL++ using the following configurations:

I chose to employ both source code and binary instrumentation methods for fuzzing in order to assess the capabilities of AFL++ in these areas.

- ❖ LLVM Mode: Instrumentation done on **FuzzGoat** source code via **afl-clang-fast** for LLVM-based coverage.
- ❖ GCC Mode: Instrumentation done on **FuzzGoat** source code via **afl-gcc** for GCC-based coverage.
- ❖ QEMU Mode: Binary-only fuzzing on a GCC-compiled **FuzzGoat** binary.

The fuzzing campaign was run for 1 hour and 15 minutes in each mode.

Evaluation Metrics:

- ❖ **Execution Speed:** Measured in executions per second, indicating the efficiency of the fuzzing process.
- ❖ **Code Coverage:** Evaluated using map density and count coverage metrics to determine how thoroughly the code was tested.
- ❖ **Crash Discovery:** The number of unique crashes discovered signifies the effectiveness in uncovering potential vulnerabilities.

Rationale for Exploration:

- ❖ How different instrumentation techniques affect the fuzzing process and its outcomes.
- ❖ The trade-offs between execution speed and the thoroughness of code coverage.
- ❖ Each mode's capability to discover crashes could lead to identifying potential vulnerabilities.

The aim was to determine how these modes affect the fuzzing process and to explore potential synergies between them.

Results & Analysis

The AFL++ fuzzing modes were assessed based on their impact on speed, coverage, and crash discovery. While the following graphs illustrate the differences in execution speed, saved crashes, and count coverage across the three modes, the complete result is provided in Appendix A.

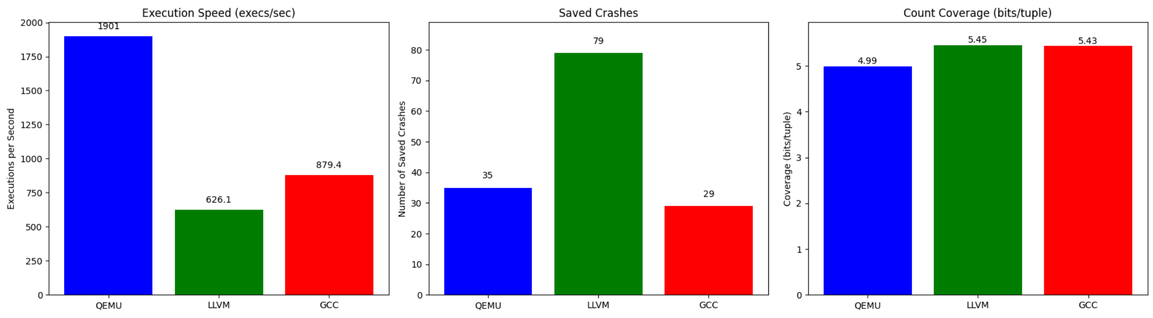


Fig 1: Comparative Analysis of three features

Comparative Analysis:

- ❖ QEMU Mode had the highest execution speed but lower code coverage.
- ❖ LLVM Mode discovered the most crashes with the highest code coverage but at a slower execution speed.
- ❖ GCC Mode offered a balance between the two in terms of speed and crash discovery.

Synergy Between Features

The investigation into whether certain features work better in tandem or individually revealed that:

❖ Binary-only fuzzing (QEMU mode) is beneficial when source code is unavailable, yet it may lack the depth of coverage that source code instrumentation can provide.
❖ LLVM instrumentation potentially offers more granular control and deeper code introspection, which might be more suitable for complex targets.
❖ While less sophisticated than LLVM mode, GCC mode still provides a balance between speed and crash discovery, making it a viable option when ease of setup is a priority.

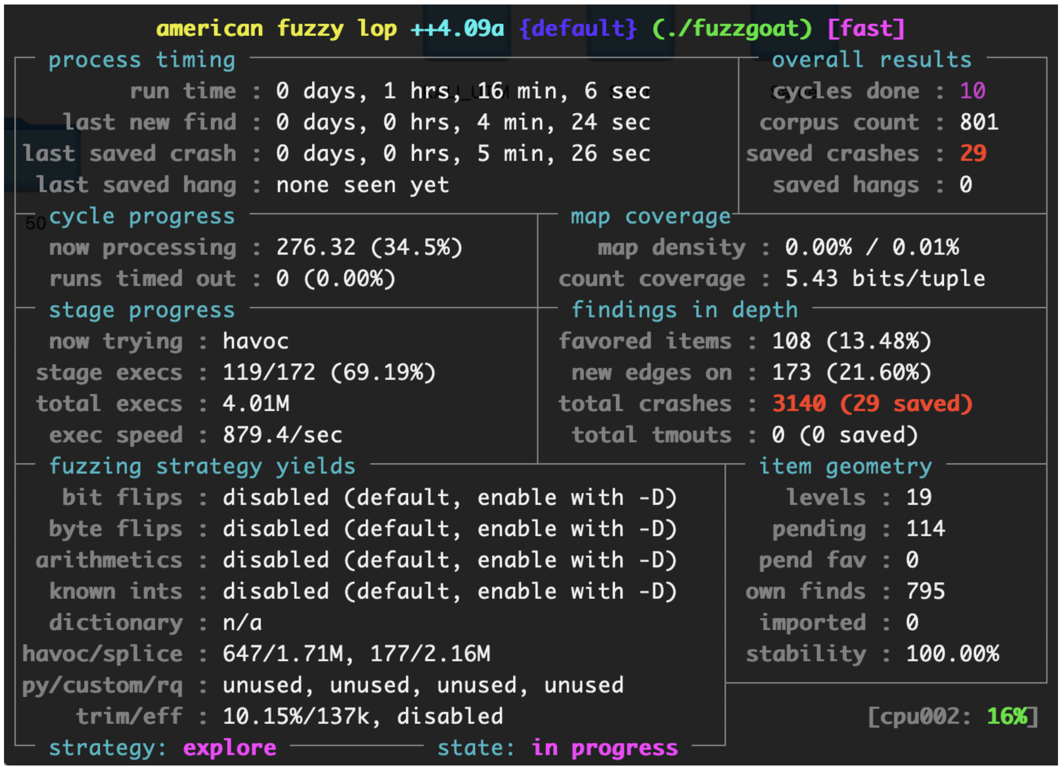
Conclusion

Each fuzzing mode has its advantages depending on the goals of the fuzzing session. LLVM mode is suited for in-depth analysis, QEMU mode is for quick scans when source code is unavailable, and GCC mode is for a balance of speed and thoroughness. These findings can inform the choice of fuzzing strategy for future security testing endeavors.

Appendix A: Detailed Output

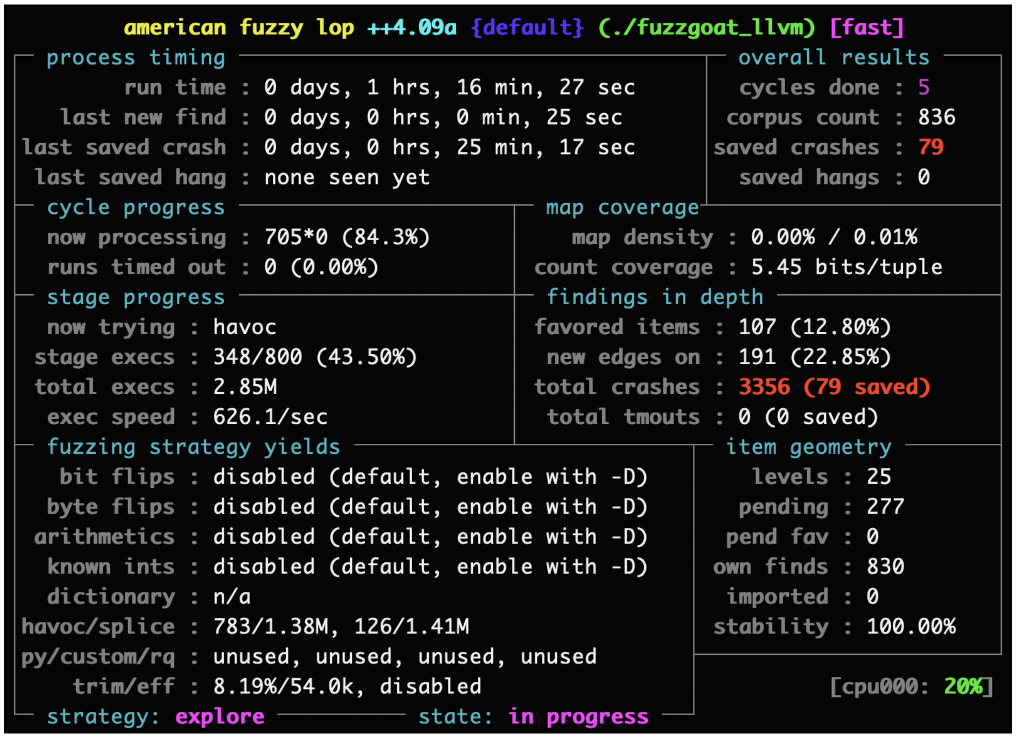
This appendix includes the complete output logs for reference.

1. **<u>The AFL++ output metrics in GCC mode:</u>**



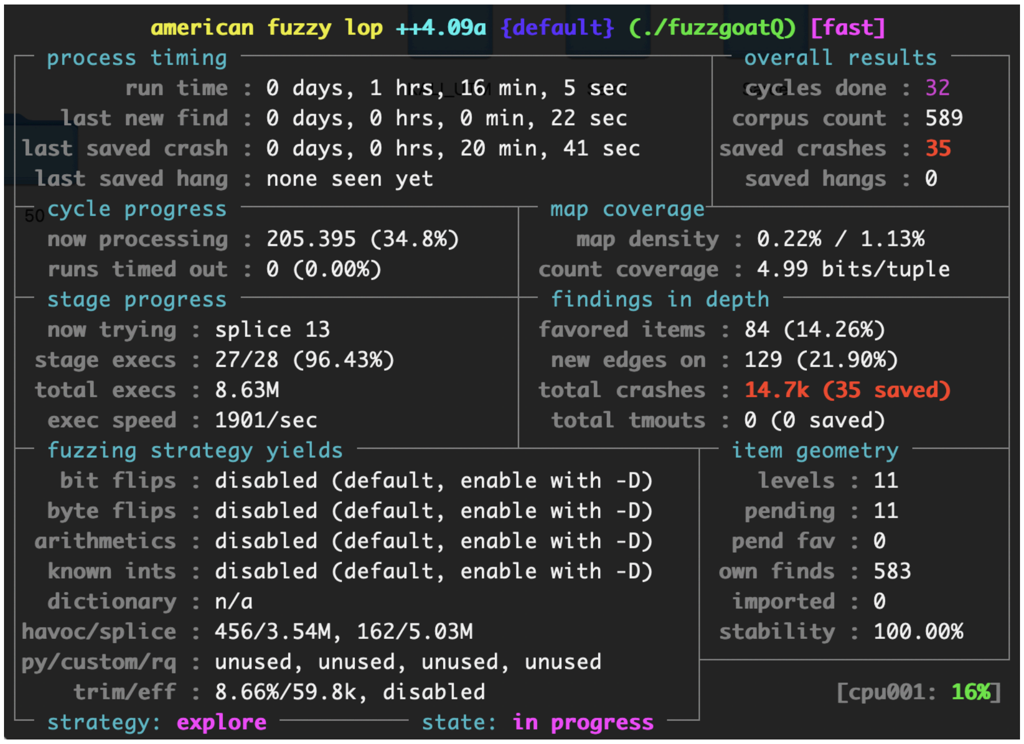
```
             american fuzzy lop ++4.09a {default} (./fuzzgoat) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 1 hrs, 16 min, 6 sec      │  cycles done : 10    │
│   last new find : 0 days, 0 hrs, 4 min, 24 sec      │ corpus count : 801   │
│last saved crash : 0 days, 0 hrs, 5 min, 26 sec      │saved crashes : 29    │
│ last saved hang : none seen yet                     │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 276.32 (34.5%)     │    map density : 0.00% / 0.01%      │
│  runs timed out : 0 (0.00%)          │ count coverage : 5.43 bits/tuple    │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                  │ favored items : 108 (13.48%)        │
│ stage execs : 119/172 (69.19%)       │  new edges on : 173 (21.60%)        │
│ total execs : 4.01M                  │ total crashes : 3140 (29 saved)     │
│  exec speed : 879.4/sec              │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ────────────┴───────────┬─ item geometry ────────┤
│   bit flips : disabled (default, enable with -D) │    levels : 19         │
│  byte flips : disabled (default, enable with -D) │   pending : 114        │
│ arithmetics : disabled (default, enable with -D) │  pend fav : 0          │
│  known ints : disabled (default, enable with -D) │ own finds : 795        │
│  dictionary : n/a                                │  imported : 0          │
│havoc/splice : 647/1.71M, 177/2.16M               │ stability : 100.00%    │
│py/custom/rq : unused, unused, unused, unused     ├────────────────────────┘
│    trim/eff : 10.15%/137k, disabled              │          [cpu002: 16%]
└─ strategy: explore ────────── state: in progress ┘
```

2. **The AFL++ output metrics in LLVM mode:**



```
          american fuzzy lop ++4.09a {default} (./fuzzgoat_llvm) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 1 hrs, 16 min, 27 sec      │  cycles done : 5     │
│   last new find : 0 days, 0 hrs, 0 min, 25 sec       │ corpus count : 836   │
│last saved crash : 0 days, 0 hrs, 25 min, 17 sec      │saved crashes : 79    │
│ last saved hang : none seen yet                      │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage─┴──────────────────────┤
│  now processing : 705*0 (84.3%)      │    map density : 0.00% / 0.01%      │
│  runs timed out : 0 (0.00%)          │ count coverage : 5.45 bits/tuple    │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                  │ favored items : 107 (12.80%)        │
│ stage execs : 348/800 (43.50%)       │  new edges on : 191 (22.85%)        │
│ total execs : 2.85M                  │ total crashes : 3356 (79 saved)     │
│  exec speed : 626.1/sec              │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ────────────┴───────────┬─ item geometry ────────┤
│   bit flips : disabled (default, enable with -D)  │    levels : 25         │
│  byte flips : disabled (default, enable with -D)  │   pending : 277        │
│ arithmetics : disabled (default, enable with -D)  │  pend fav : 0          │
│  known ints : disabled (default, enable with -D)  │ own finds : 830        │
│  dictionary : n/a                                 │  imported : 0          │
│havoc/splice : 783/1.38M, 126/1.41M                │ stability : 100.00%    │
│py/custom/rq : unused, unused, unused, unused      ├────────────────────────┘
│    trim/eff : 8.19%/54.0k, disabled               │          [cpu000: 20%]
└─ strategy: explore ────────── state: in progress ─┘
```

3. **The AFL++ output metrics in QEMU mode:**



```
          american fuzzy lop ++4.09a {default} (./fuzzgoatQ) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 1 hrs, 16 min, 5 sec       │  cycles done : 32    │
│   last new find : 0 days, 0 hrs, 0 min, 22 sec       │ corpus count : 589   │
│last saved crash : 0 days, 0 hrs, 20 min, 41 sec      │saved crashes : 35    │
│ last saved hang : none seen yet                      │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage─┴──────────────────────┤
│  now processing : 205.395 (34.8%)    │    map density : 0.22% / 1.13%      │
│  runs timed out : 0 (0.00%)          │ count coverage : 4.99 bits/tuple    │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : splice 13              │ favored items : 84 (14.26%)         │
│ stage execs : 27/28 (96.43%)         │  new edges on : 129 (21.90%)        │
│ total execs : 8.63M                  │ total crashes : 14.7k (35 saved)    │
│  exec speed : 1901/sec               │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ────────────┴───────────┬─ item geometry ────────┤
│   bit flips : disabled (default, enable with -D)  │    levels : 11         │
│  byte flips : disabled (default, enable with -D)  │   pending : 11         │
│ arithmetics : disabled (default, enable with -D)  │  pend fav : 0          │
│  known ints : disabled (default, enable with -D)  │ own finds : 583        │
│  dictionary : n/a                                 │  imported : 0          │
│havoc/splice : 456/3.54M, 162/5.03M                │ stability : 100.00%    │
│py/custom/rq : unused, unused, unused, unused      ├────────────────────────┘
│    trim/eff : 8.66%/59.8k, disabled               │          [cpu001: 16%]
└─ strategy: explore ────────── state: in progress ─┘
```