Fuzz Testing Loguru with Mayhem

Vincent Titterton

Selected Repository

For this project, I've decided to fuzz test a lightweight C++ logging library, Loguru (github.com/emilk/loguru). The source code is composed of just two files (loguru.cpp and loguru.hpp). Using this library can be done with a single include:

```
#include <loguru.cpp>
```

Then, logging of a string `str` can be preformed with the following code:

```
LOG_F(INFO, foo);
```

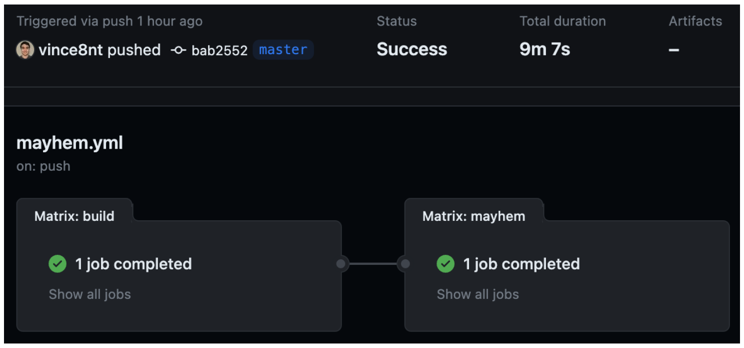
This repository uses CMake to build the logger along with some included example and test programs.

Forked Repository

Here is my forked version of Loguru: github.com/vince8nt/loguru.

This fork includes the following notable changes from its origin:

- `.github/workflows/mayhem.yml`
  Helps set up Mayhem and provides locations of the Dockerfile and Mayhemfile.
- `mayhem/Dockerfile`
  Used by Docker to build the code and dependencies in `harness`.
- `mayhem/Mayhemfile`
  Specifies the fuzzing behavior for the Mayhem run
- `pkgs/container/loguru`
  Used by github actions to automatically perform fuzz testing when pushing to the repository. List of actions: github.com/vince8nt/loguru/actions.
  The workflow is currently passing all stages.



Triggered via push 1 hour ago — vince8nt pushed bab2552 master — Status: Success — Total duration: 9m 7s — Artifacts: –

mayhem.yml
on: push

Matrix: build — 1 job completed — Show all jobs

Matrix: mayhem — 1 job completed — Show all jobs

- `harness/...`
  The current harness program to be fuzzed by Mayhem. Includes build tools.
- `old_harness/...`
  A place to store my older harness programs that are not currently being tested.

Implemented Harnessing

Since Loguru is a C++ library, it does not have a main(), and therefore cannot be directly fuzzed by Mayhem (Mayhem uses either stdin or command line arg files for fuzzing). My goal in harnessing, was to create a simple program which accepts (fuzzing) inputs from stdin, and passes these to Loguru. To do this, I looked at the provided example programs for information on how to build a program that includes Loguru. I created 3 files located in `harness`:

`CMakeLists.txt`	build instructions
`build_and_run.sh`	command line shortcut to build
`harness1.cpp`	the current harness program

Version 1 - `old_harness/harness1.cpp`

This harness program works as follows:

- Initialize Loguru logger
- Repeatedly read from stdin into a bounded buffer (64 bytes)
- Null terminate what was read
- log the null-terminated buffer (string)
- Once stdin is closed, exit.

With this harness, I was able to successfully fuzz Loguru and find 2 defects (a null pointer dereference and an improper input validation). This harness was also very efficient, resulting in 150.05 tests run per second.

Version 2 - `harness/harness1.cpp`

After some inspection of my first harness program, I noticed some issues that could be improved:

- If less than 64 characters were provided by Mayhem, nothing was passed to Loguru.
- Every string passed to Loguru was exactly 64 bits if no null characters were provided by Mayhem.
- If a null characters was provided by Mayhem, the rest of the 64 bits were not read by Loguru.

Fixing these issues, Version 2 works as follows:

- Read up to 1024 bytes from stdin into a bounded buffer.
- Initialize Loguru logger
- Loop through the buffer and log each null terminated string

This now allowed Mayhem to log many short strings (including the empty string) with a small input size to stdin. With this harness, I was able to successfully fuzz Loguru and now find 3 defects (2 null pointer dereferences and 1 improper input validation). This harness was slightly less efficient with 101.9 tests run per second (but this is still good efficiency).

```
harness1.cpp
```

```
#include <iostream>
#include "../loguru.cpp"
#include <unistd.h>
#include <fcntl.h>
#define BUFF_SIZE 1024
int main(int argc, char* argv[]) {
    char buff[BUFF_SIZE + 1];
    ssize_t bytesWritten = 0;
    ssize_t bytesRead = read(STDIN_FILENO, buff, BUFF_SIZE);
    if (bytesRead == -1)
        return 0; // exit success so not counted as defect
    buff[bytesRead] = '\0';
    loguru::init(argc, argv);
    LOG_F(INFO, "begin fuzz");
    while (1) {
        if (bytesWritten >= bytesRead)
            break;
        LOG_F(INFO, buff + bytesWritten);
        ++bytesWritten;
        while (bytesWritten < bytesRead and buff[bytesWritten] != '\0')
            ++bytesWritten;
    }
    LOG_F(INFO, "end fuzz");
}
```

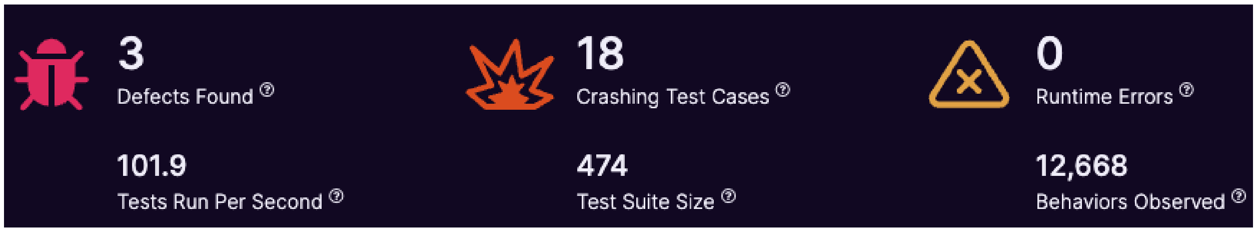
Target Eligibility and Approval

When initially forking the Loguru repository, I did not do so under mayhemheros. Thus, I am only able to submit a pull request to the Loguru repository, not the Mayhem Heroes repository. However, I did make sure that Loguru is an eligible target for a new repo submission:

- Does not already exist under github.com/mayhemheroes
- Over 100 stars (has 1.6k)
- Has a commit from the past 6 months (2 months ago)
- Not a part of OSS-Fuzz
- Nothing inappropriate
- Over 100 tests run per second
- Defects found

This means that if I were to redo the initial fork (and add my changes), I could likely get a pull request to be accepted.

Results


3
Defects Found

18
Crashing Test Cases

0
Runtime Errors

101.9
Tests Run Per Second

474
Test Suite Size

12,668
Behaviors Observed

To achieve these results, I added additional testing tasks to `Mayhemfile` (exploitability_factors, regression_testing, behavior_testing, coverage_analysis). I also did trial and error modifications of `harness1.cpp`.

Mayhem is consistently able to find 3 defects in Loguru (2 null pointer dereferences and 1 improper input validation). I was able to verify that these defects are, in fact, caused by Loguru (and not my harness) through Mayhem's stack trace feature.

NULL Pointer Dereference stack trace - found by Regression Testing

```
#0 0x7ffff7ad3f2d in strfmon+0x13dd at ??:0:0
#1 0x7ffff7ad4dcd in addseverity+0x17d at ??:0:0
#2 0x7ffff7ae9bca in psiginfo+0x676a at ??:0:0
#3 0x44e9de in loguru::vtextprintf(char const*, __va_list_tag*) at
/harness/../loguru.cpp:431:7
#4 0x453572 in loguru::vlog(int, char const*, unsigned int, char const*,
__va_list_tag*) at /harness/../loguru.cpp:1522:24
#5 0x44f763 in loguru::log(int, char const*, unsigned int, char const*, ...)
at /harness/../loguru.cpp:1517:2
#6 0x4555fd in main at /harness/harness1.cpp:25:9
#7 0x7ffff7a82083 in __libc_start_main+0xf3 at ??:0:0
#8 0x44e07e in _start+0x2e at ??:0:0
```

Mayhem runs 101.9 tests per second which is considered good efficiency. Also, Mayhem's Fuzz testing is automatically performed by github actions when pushing to the repository.

Since Loguru has not previously been fuzzed (from what I could find), I consider these results to be successful.

Conclusion

My biggest challenge for this project was technical issues and troubleshooting. To solve/attempt to solve these issues, I mainly spent time reading documentation and using trial and error. I'm not very familiar with forking and pull requests, so this took a good amount of time to figure out. Unfortunately, I only realized that I did the initial fork incorrectly after fully completing the integration. I also struggled to use Docker, CMake, and Mayhem (however, I eventually resolved all of these issues).

If I had more time/could redo this project, I would make sure to attend the Mayhem office hours. This would help streamline troubleshooting, and ensure that I'm working in the right direction. With this help, I would have done the initial fork correctly, and likely been able to have ForAllSecure approve and merge a pull request.

Another possible improvement to this project would be to create a harness that does multi-threaded testing (to check for race conditions). `loguru_example/main.cpp` is a multi-threaded example program that was provided in Loguru. Using this as a reference, it would be rather straightforward to create a harness that uses Loguru asynchronously from multiple threads. This would allow Mayhem's fuzz testing to check for race conditions (something that is missing in my current implementation).

Since this project already meets all the requirements for Mayhem Heros, I plan to contact ForAllSecure's support to help me resolve the forking/pull request issues I encountered. This will hopefully allow me to have a pull request approved without much additional work.