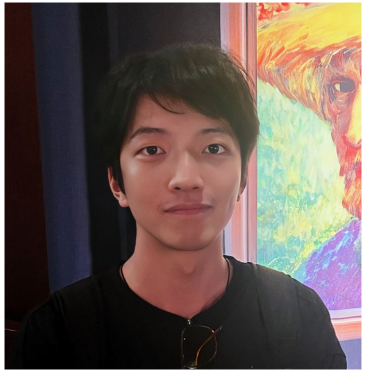
Securing the Source: Integrate Fuzzing into Open Source Software

Speaker: Jiongchi Yu

Contributors: George-Andrei Iosif, Till Kamppeter, Dongge Liu

24/08/2024

Who We Are



Jiongchi Yu

PhD Student
@ SMU

George-Andrei Iosif

Security Engineer
@ Snap Inc.

Till Kamppeter

Leader @ OpenPrinting
Fellow @ the Linux Foundation

Dongge Liu

Software Engineer
@ Google Inc.

Agenda

- OSS Security and Fuzz Testing
- OSS-Fuzz in OpenPrinting
- Advanced OSS-Fuzz Integration
- Fuzzing with Large Language Models
- Discussion and Future Works

Open Source Software Security

96%

of the total codebases
contained open source

84%

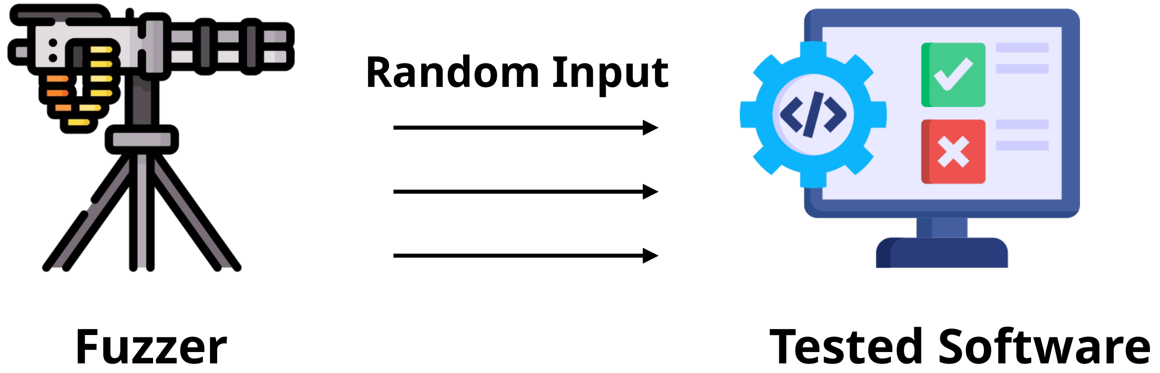
of codebases contained at least
one open source vulnerability

Open Source Software Testing

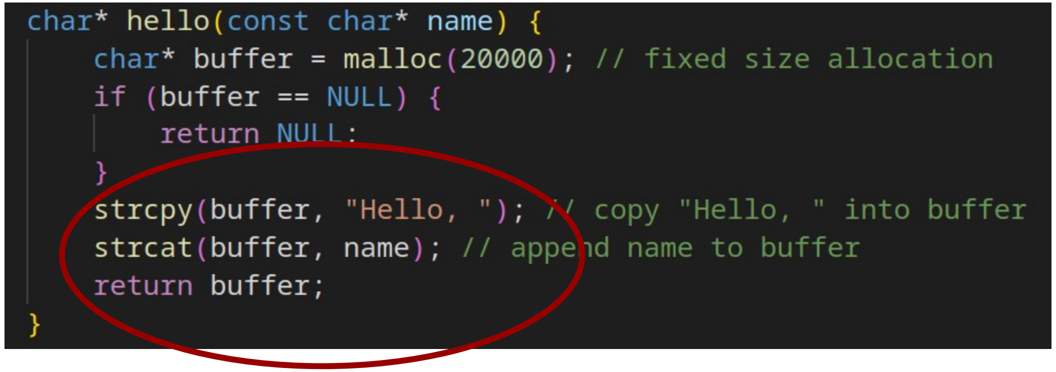
- **Continuous Testing Tools**
  - GitHub Actions, Jenkins, Travis CI ...
- **Common OSS Testing Methods**
  - Static Code Analysis
  - Unit Testing
  - Integration Testing
  - Symbolic Execution
  - **Fuzz Testing**

Fuzz Testing

- Aged but effective method: dated back to 1980s
- Blackbox fuzzing, Whitebox fuzzing, Greybox fuzzing
- Generation based fuzzing: BooFuzz , Peach Fuzzer
- **Mutation based fuzzing: AFL, LibFuzzer, Honggfuzz ...**



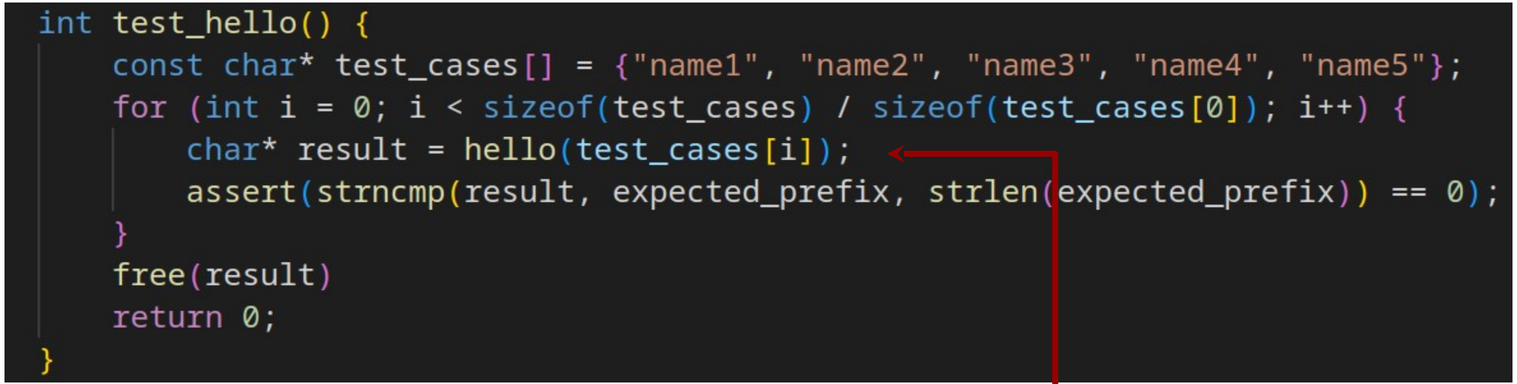
Bug Example



```c
char* hello(const char* name) {
    char* buffer = malloc(20000); // fixed size allocation
    if (buffer == NULL) {
        return NULL;
    }
    strcpy(buffer, "Hello, "); // copy "Hello, " into buffer
    strcat(buffer, name); // append name to buffer
    return buffer;
}
```

Function with

Buffer Overflow Bug



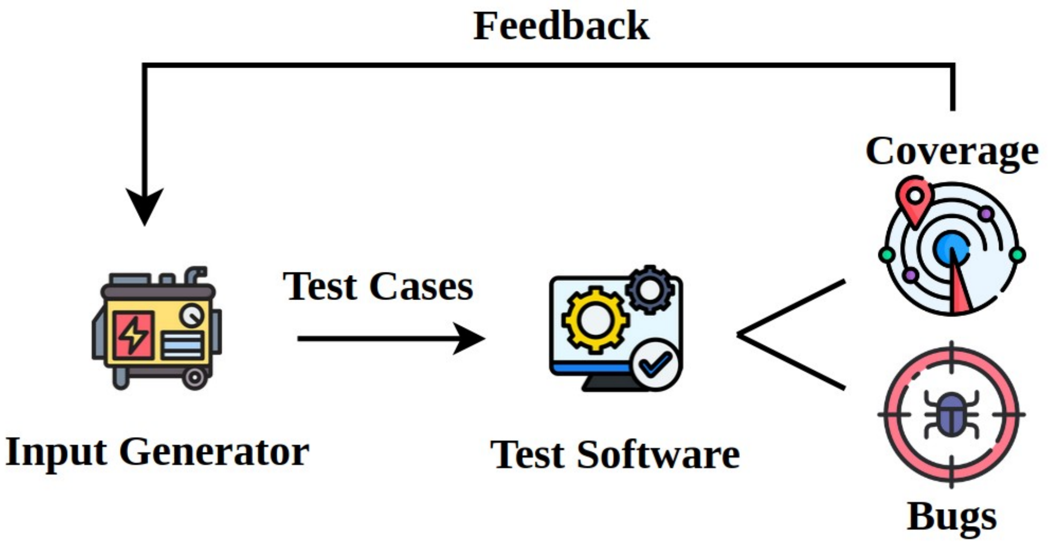
```c
int test_hello() {
    const char* test_cases[] = {"name1", "name2", "name3", "name4", "name5"};
    for (int i = 0; i < sizeof(test_cases) / sizeof(test_cases[0]); i++) {
        char* result = hello(test_cases[i]);
        assert(strncmp(result, expected_prefix, strlen(expected_prefix)) == 0);
    }
    free(result)
    return 0;
}
```

Unit Testing

Fuzzing: Infinite Input

Fuzzing Workflow

1. Write Fuzz Harness
2. Instrumentation
3. Seed Selection
4. Input Mutation
5. Execution
6. Coverage Collection
7. Bug Detection and Post Analysis



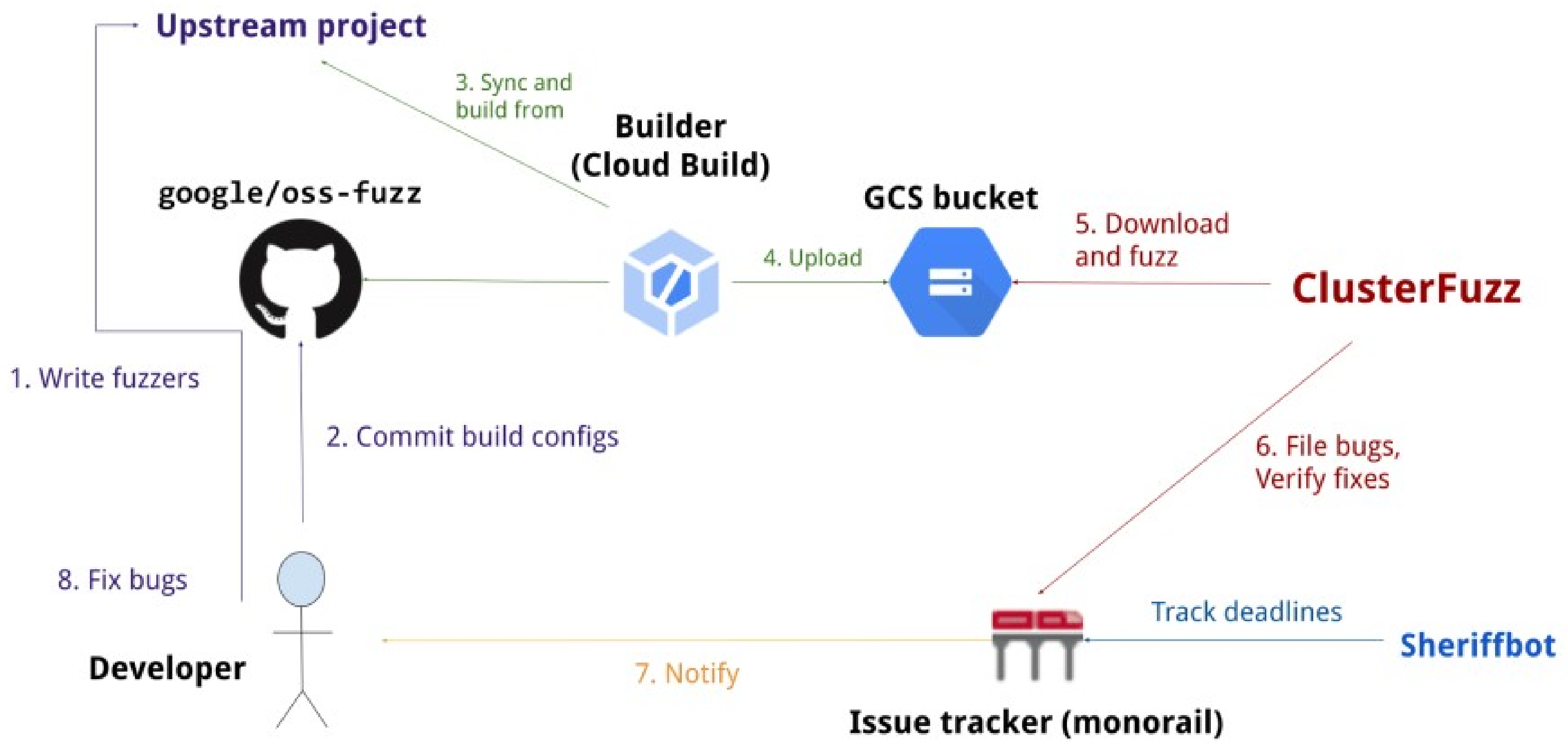
What Are We Fuzzing About?

- **Fuzzer Evaluation Metrics**
  - Code/Path/Block Coverage
  - Unique Bug Count
  - Fuzz Efficiency (TP/FP Ratio)
  - …
- **Developing/Research Directions in Fuzzing**
  - **Overcome Coverage Plateau** (Magic Number, Complex Logic …)
  - **Improve Fuzzing Extensibility** (Specific Target, Fuzz Automation)
  - **Increase Fuzzing Efficiency** (Distributed Fuzzing, Test Case Selection)

Continuous Fuzzing: OSS-Fuzz

- **Simplify the Fuzzing Workflow**
  - Build with preset fuzzing engines
- **Free Service for Developers**
  - Run fuzzers at scale
- **Automated Post-Analysis Process**
  - Fuzz result reporting (crashes, timeouts, etc.)
  - Coverage visualization

Continuous Fuzzing: OSS-Fuzz



Continuous Fuzzing: OSS-Fuzz

- Since 2016, OSS-Fuzz had found **12000+** vulnerabilities and **36000+** bugs across **1000+** projects
- Improve with **Fuzz Introspector** and **OSS-Fuzz-Gen**

In response to CVE-2014-0160

OSS-Fuzz in OpenPrinting

- Nearly **2/3 of the all** 37 OpenPrinting projects are mainly implemented in **C/C++**
- OpenPrinting projects involves **multiple test input types**
  - C structures, Image, PDF, Argv, etc.
- OpenPrinting has integrates **three main projects** into OSS-Fuzz workflow
  - cups
  - libcups
  - cups-filters

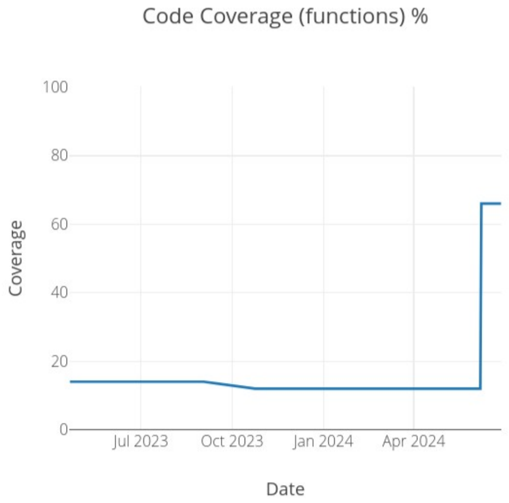
Previous OpenPrinting Testing

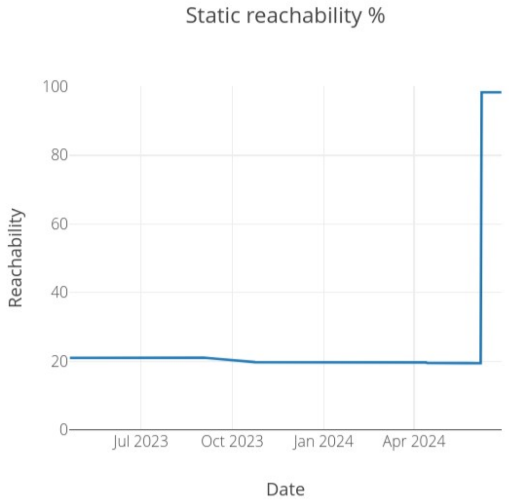
- Mainly **unit tests** written by developers
- **One** manual written fuzz driver in cups by for testing libipp
- AFL build scripts for libcups

NOT ENOUGH FUZZING!

Our Progress

- **Integrate three projects into OSS-Fuzz**
  - **35** issues reported, **22** resolved
  - **65%** average coverage, **nearly 100%** static reachability
- **Enable Fuzz Introspector**
- **Adopt OSS-Fuzz-Gen**



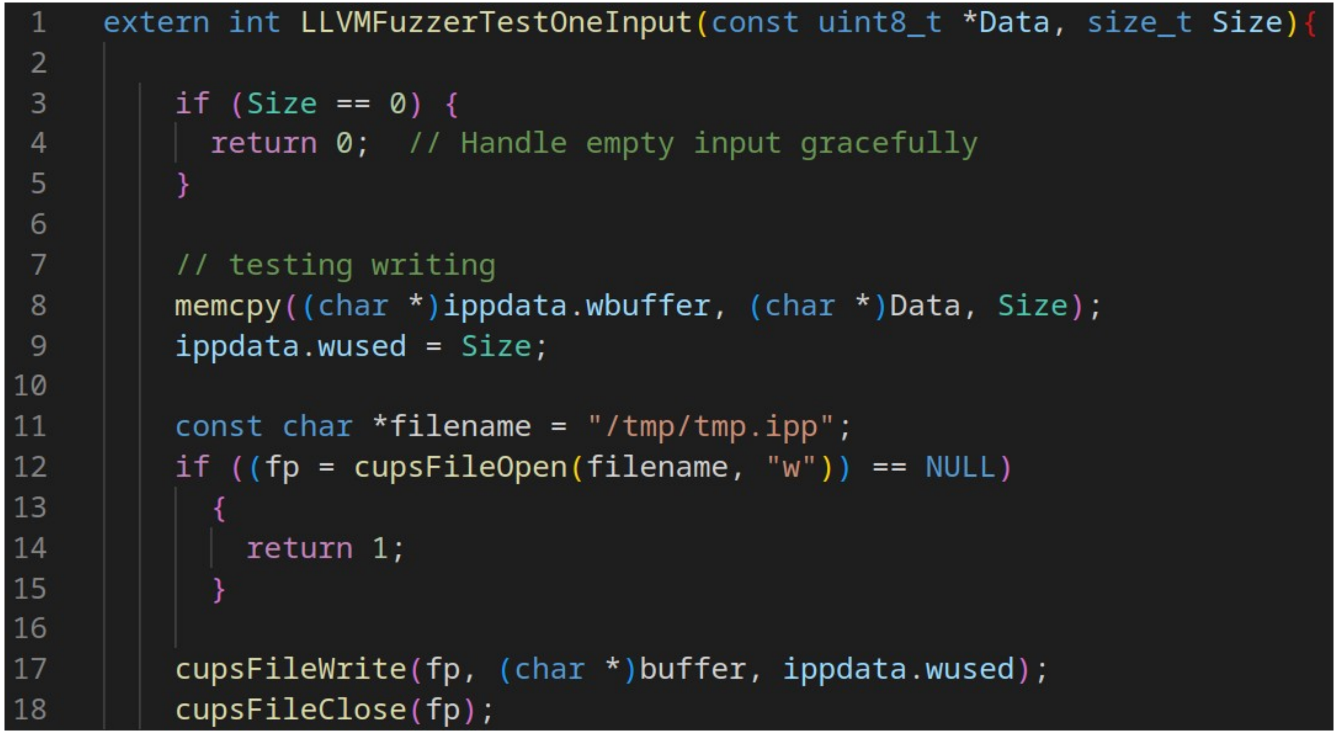


How To Integrate

1. Prepare fuzz harnesses and build locally
2. Create project configuration and merge into OSS-Fuzz
3. **Leave everything else to OSS-Fuzz…**

Prepare Fuzz Harnesses

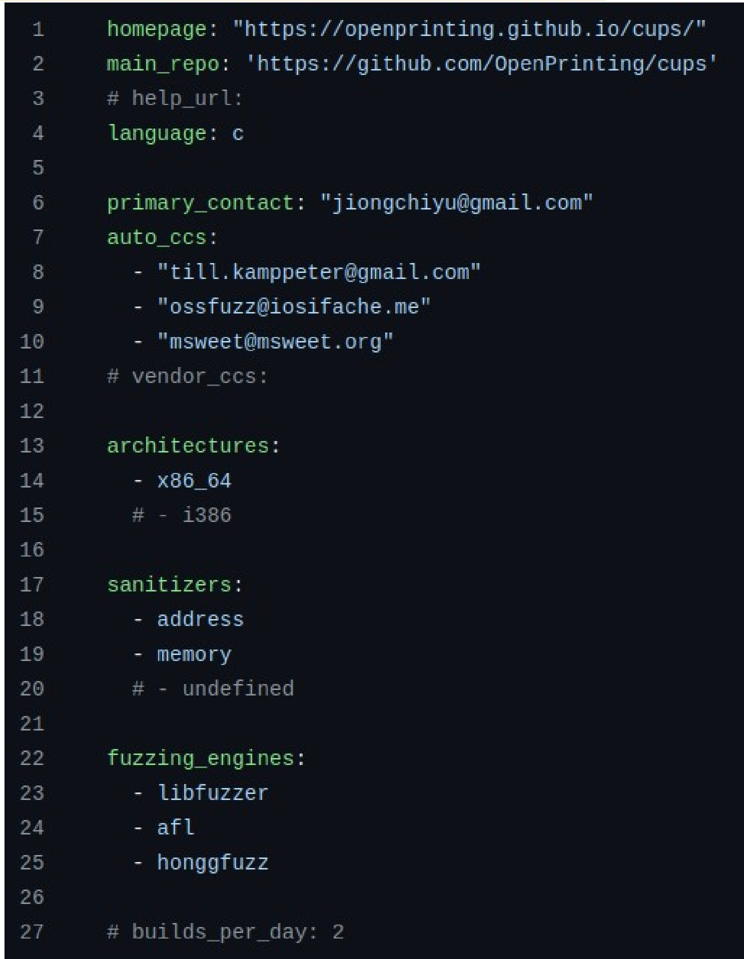
Replace main function with LLVMFuzzerTestOneInput and build harness locally



```
extern int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size){

    if (Size == 0) {
      return 0;  // Handle empty input gracefully
    }

    // testing writing
    memcpy((char *)ippdata.wbuffer, (char *)Data, Size);
    ippdata.wused = Size;

    const char *filename = "/tmp/tmp.ipp";
    if ((fp = cupsFileOpen(filename, "w")) == NULL)
      {
        return 1;
      }

    cupsFileWrite(fp, (char *)buffer, ippdata.wused);
    cupsFileClose(fp);
```

Create OSS-Fuzz Project Configs

- **Dockerfile**
    - Dependencies for building the harnesses
- **project.yaml**
- **build.sh**
    - Shell script to build harnesses
    - Can replace with customized path in Dockerfile



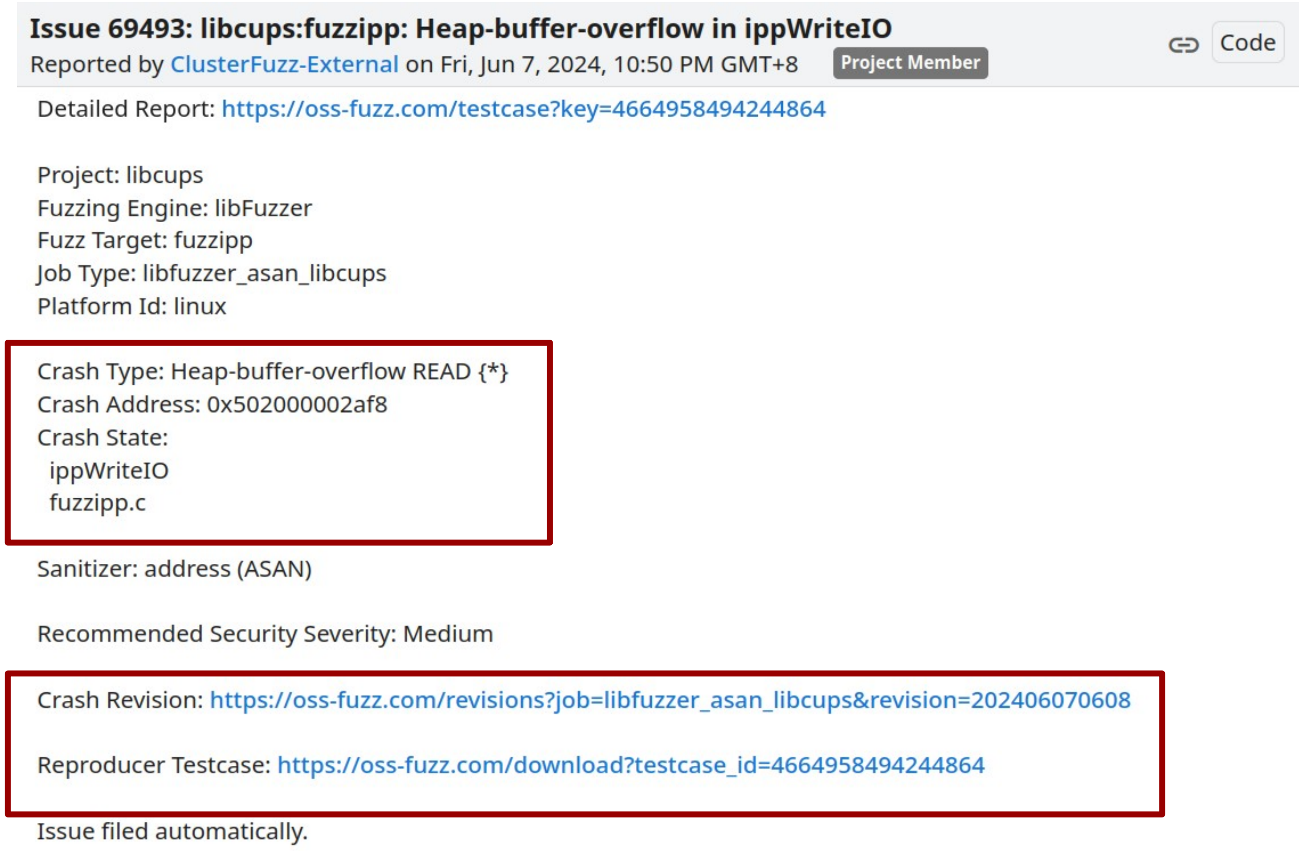
```yaml
homepage: "https://openprinting.github.io/cups/"
main_repo: 'https://github.com/OpenPrinting/cups'
# help_url:
language: c

primary_contact: "jiongchiyu@gmail.com"
auto_ccs:
  - "till.kamppeter@gmail.com"
  - "ossfuzz@iosifache.me"
  - "msweet@msweet.org"
# vendor_ccs:

architectures:
  - x86_64
  # - i386

sanitizers:
  - address
  - memory
  # - undefined

fuzzing_engines:
  - libfuzzer
  - afl
  - honggfuzz

# builds_per_day: 2
```

Our Suggested Strategies

- Internal discussion for **prioritized projects**
- Start from existing **unit and integration** tests
- Identify **important or complex** functions with domain knowledge
- Refer from coverage information for **less tested** functions

Fuzzing Statistics

- **OSS-Fuzz Dashboard**
  - Build logs
  - Potential bug information
- **Fuzz Introspector Visualization**



Issue 69493: libcups:fuzzipp: Heap-buffer-overflow in ippWriteIO
Reported by ClusterFuzz-External on Fri, Jun 7, 2024, 10:50 PM GMT+8 Project Member

Detailed Report: https://oss-fuzz.com/testcase?key=4664958494244864

Project: libcups
Fuzzing Engine: libFuzzer
Fuzz Target: fuzzipp
Job Type: libfuzzer_asan_libcups
Platform Id: linux

Crash Type: Heap-buffer-overflow READ {*}
Crash Address: 0x502000002af8
Crash State:
 ippWriteIO
 fuzzipp.c

Sanitizer: address (ASAN)

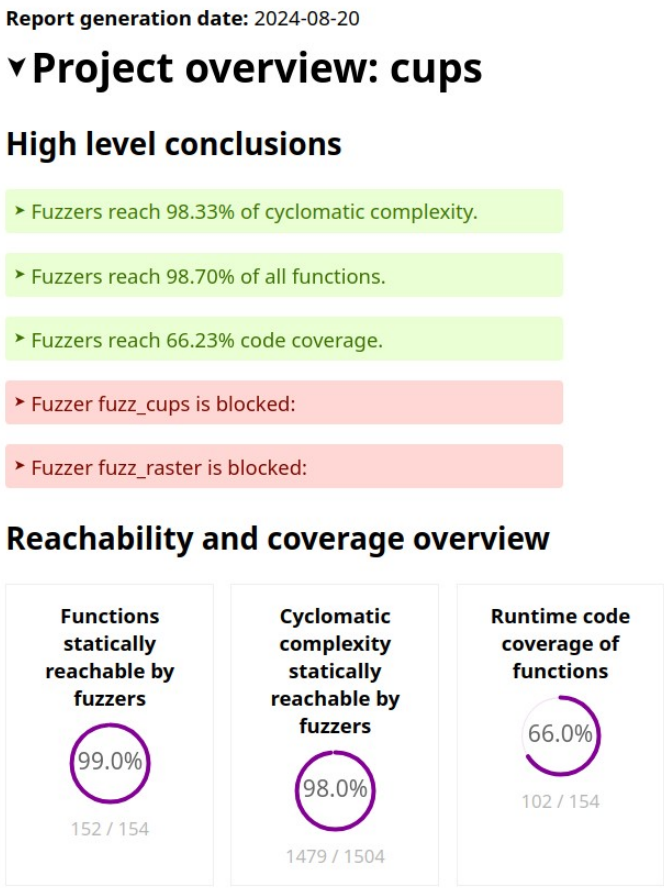
Recommended Security Severity: Medium

Crash Revision: https://oss-fuzz.com/revisions?job=libfuzzer_asan_libcups&revision=202406070608

Reproducer Testcase: https://oss-fuzz.com/download?testcase_id=4664958494244864

Issue filed automatically.

Fuzz Introspector Visualization



Report generation date: 2024-08-20

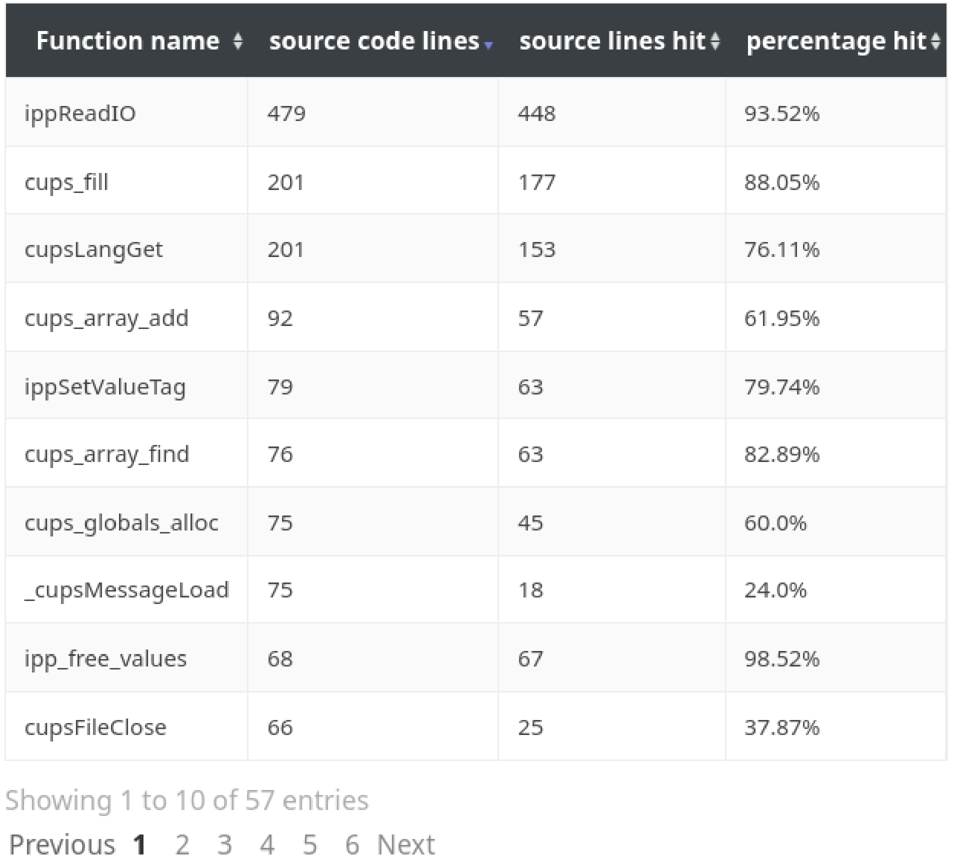
Project overview: cups

High level conclusions

- Fuzzers reach 98.33% of cyclomatic complexity.
- Fuzzers reach 98.70% of all functions.
- Fuzzers reach 66.23% code coverage.
- Fuzzer fuzz_cups is blocked:
- Fuzzer fuzz_raster is blocked:

Reachability and coverage overview

Functions statically reachable by fuzzers
99.0%
152 / 154
Cyclomatic complexity statically reachable by fuzzers
98.0%
1479 / 1504
Runtime code coverage of functions
66.0%
102 / 154

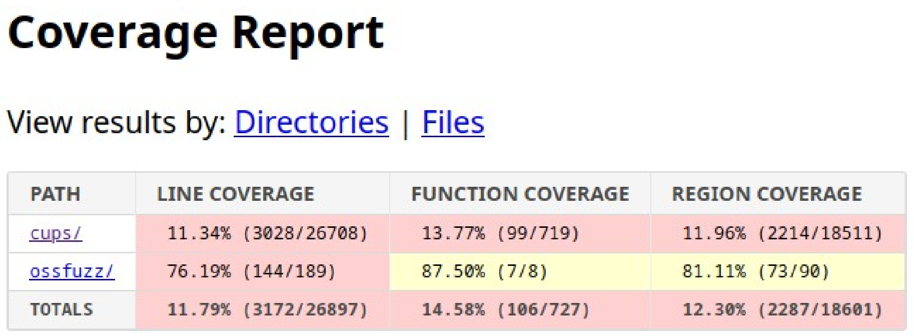


Function name	source code lines	source lines hit	percentage hit
ippReadIO	479	448	93.52%
cups_fill	201	177	88.05%
cupsLangGet	201	153	76.11%
cups_array_add	92	57	61.95%
ippSetValueTag	79	63	79.74%
cups_array_find	76	63	82.89%
cups_globals_alloc	75	45	60.0%
_cupsMessageLoad	75	18	24.0%
ipp_free_values	68	67	98.52%
cupsFileClose	66	25	37.87%

Showing 1 to 10 of 57 entries

Previous **1** 2 3 4 5 6 Next

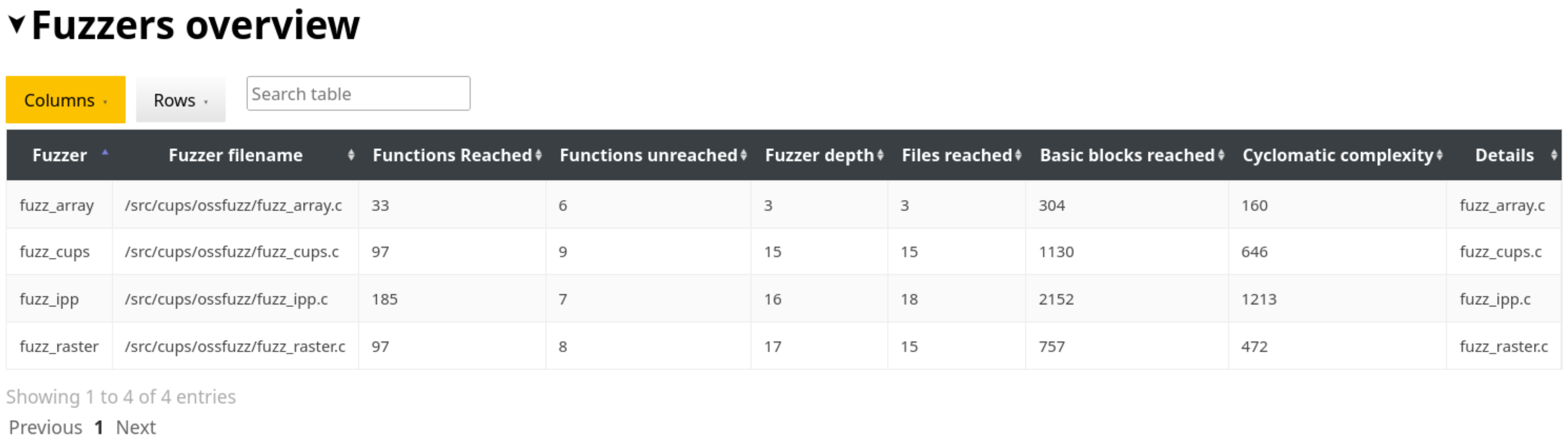
Fuzz Introspector Visualization



Coverage Report

View results by: Directories | Files

PATH	LINE COVERAGE	FUNCTION COVERAGE	REGION COVERAGE
cups/	11.34% (3028/26708)	13.77% (99/719)	11.96% (2214/18511)
ossfuzz/	76.19% (144/189)	87.50% (7/8)	81.11% (73/90)
TOTALS	11.79% (3172/26897)	14.58% (106/727)	12.30% (2287/18601)



Fuzzers overview

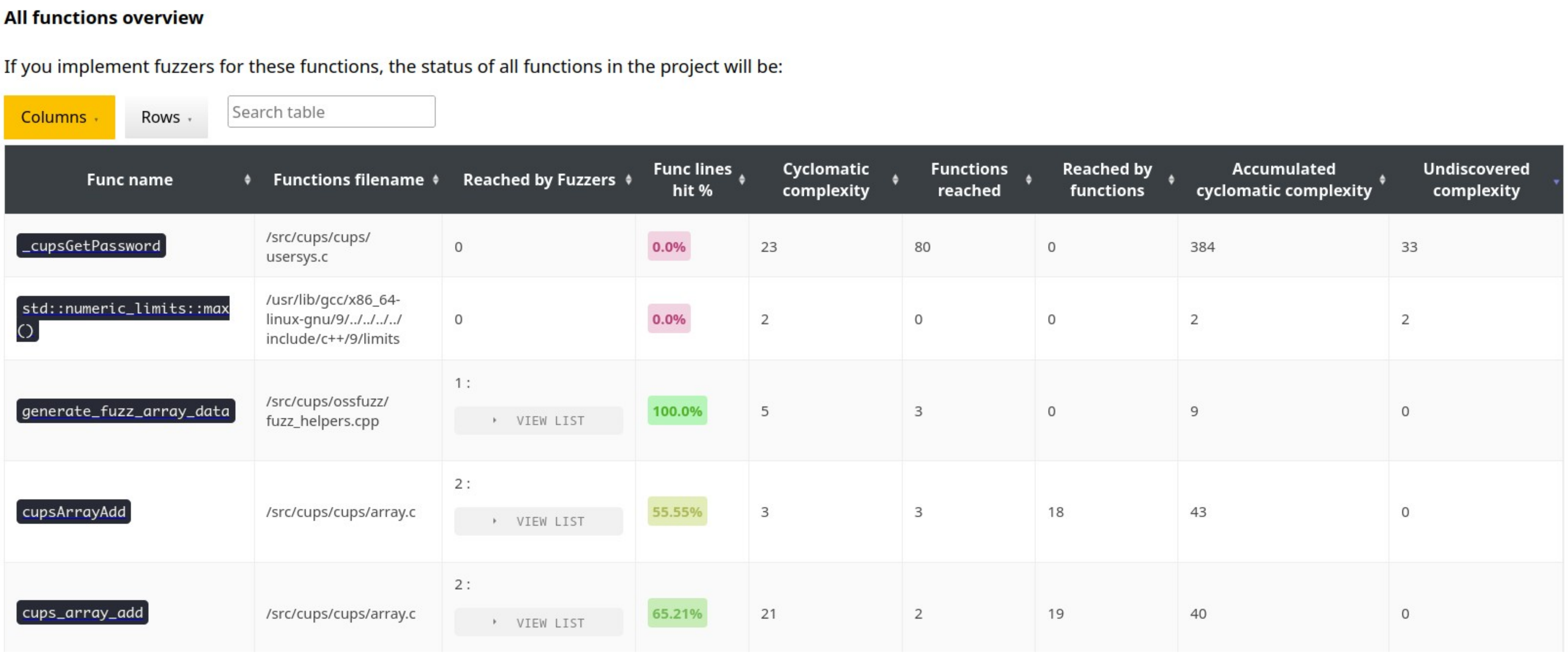
Columns Rows Search table

Fuzzer	Fuzzer filename	Functions Reached	Functions unreached	Fuzzer depth	Files reached	Basic blocks reached	Cyclomatic complexity	Details
fuzz_array	/src/cups/ossfuzz/fuzz_array.c	33	6	3	3	304	160	fuzz_array.c
fuzz_cups	/src/cups/ossfuzz/fuzz_cups.c	97	9	15	15	1130	646	fuzz_cups.c
fuzz_ipp	/src/cups/ossfuzz/fuzz_ipp.c	185	7	16	18	2152	1213	fuzz_ipp.c
fuzz_raster	/src/cups/ossfuzz/fuzz_raster.c	97	8	17	15	757	472	fuzz_raster.c

Showing 1 to 4 of 4 entries

Previous 1 Next

Fuzz Introspector Visualization



All functions overview

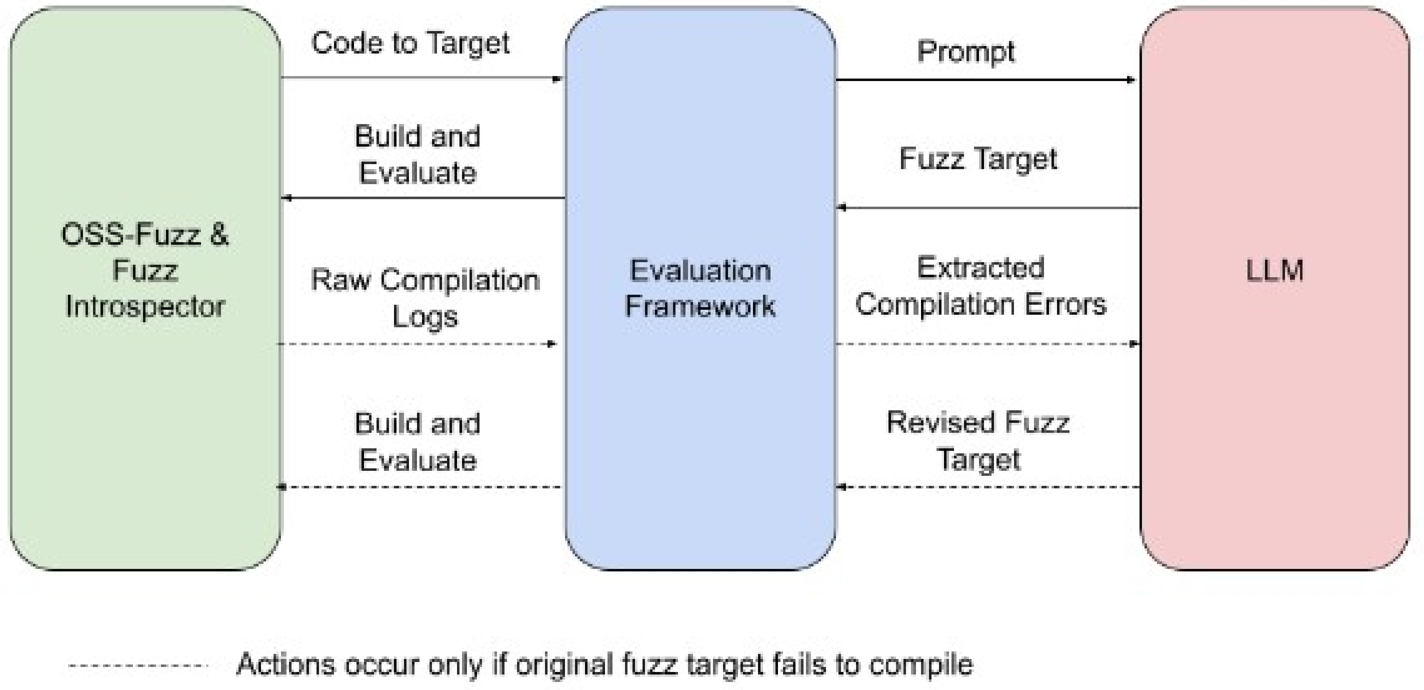
If you implement fuzzers for these functions, the status of all functions in the project will be:

Func name	Functions filename	Reached by Fuzzers	Func lines hit %	Cyclomatic complexity	Functions reached	Reached by functions	Accumulated cyclomatic complexity	Undiscovered complexity
_cupsGetPassword	/src/cups/cups/usersys.c	0	0.0%	23	80	0	384	33
std::numeric_limits::max()	/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/limits	0	0.0%	2	0	0	2	2
generate_fuzz_array_data	/src/cups/ossfuzz/fuzz_helpers.cpp	1 : VIEW LIST	100.0%	5	3	0	9	0
cupsArrayAdd	/src/cups/cups/array.c	2 : VIEW LIST	55.55%	3	3	18	43	0
cups_array_add	/src/cups/cups/array.c	2 : VIEW LIST	65.21%	21	2	19	40	0

Advanced OSS-Fuzz Integration

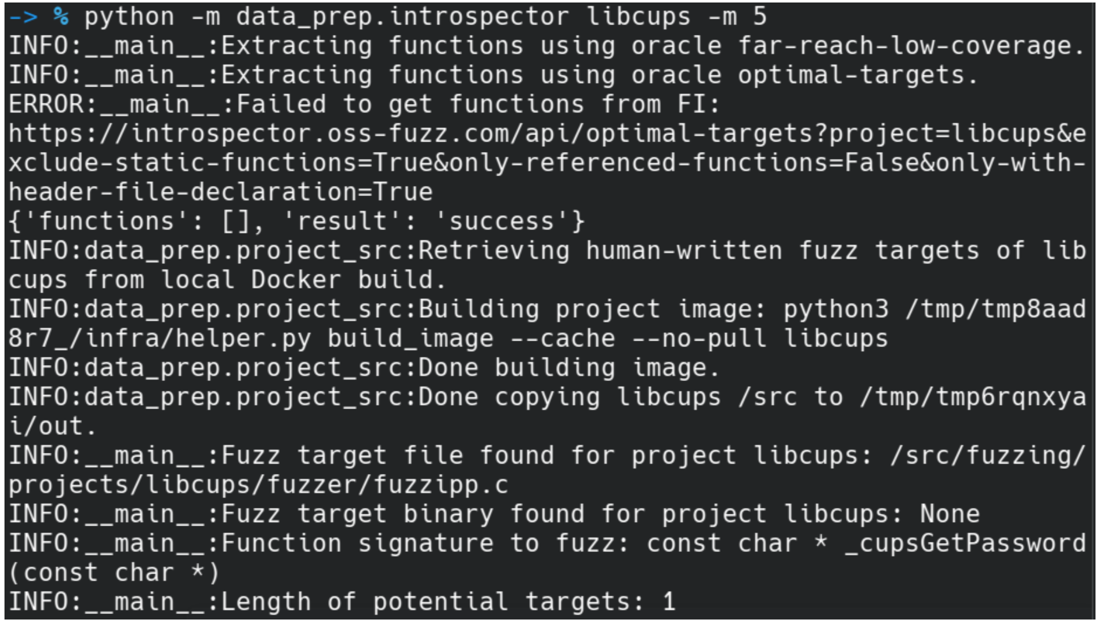
- Structured Input
- Dynamic Linking
- High Quality Seed
- Mutate Dictionary
- Customized Fuzz Oracle

OSS-Fuzz-Gen


OSS-Fuzz & Fuzz Introspector
Code to Target
Build and Evaluate
Raw Compilation Logs
Build and Evaluate
Evaluation Framework
Prompt
Fuzz Target
Extracted Compilation Errors
Revised Fuzz Target
LLM
Actions occur only if original fuzz target fails to compile

OSS-Fuzz-Gen

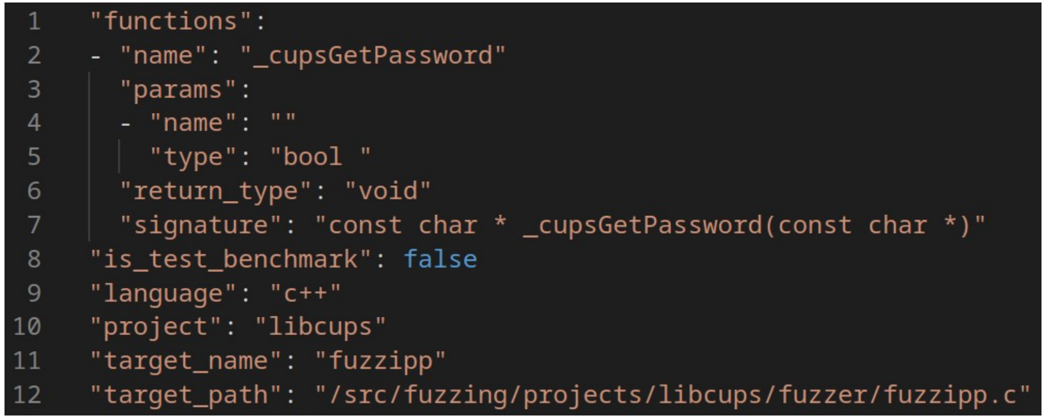
- **Few-shot** prompt with **Gemini, GPTs**, etc.
- Generate config with assistance of **Fuzz-Introspector**
- Has included **1300+** benchmarks from **297+** OSS projects



```
-> % python -m data_prep.introspector libcups -m 5
INFO:__main__:Extracting functions using oracle far-reach-low-coverage.
INFO:__main__:Extracting functions using oracle optimal-targets.
ERROR:__main__:Failed to get functions from FI:
https://introspector.oss-fuzz.com/api/optimal-targets?project=libcups&e
xclude-static-functions=True&only-referenced-functions=False&only-with-
header-file-declaration=True
{'functions': [], 'result': 'success'}
INFO:data_prep.project_src:Retrieving human-written fuzz targets of lib
cups from local Docker build.
INFO:data_prep.project_src:Building project image: python3 /tmp/tmp8aad
8r7_/infra/helper.py build_image --cache --no-pull libcups
INFO:data_prep.project_src:Done building image.
INFO:data_prep.project_src:Done copying libcups /src to /tmp/tmp6rqnxya
i/out.
INFO:__main__:Fuzz target file found for project libcups: /src/fuzzing/
projects/libcups/fuzzer/fuzzipp.c
INFO:__main__:Fuzz target binary found for project libcups: None
INFO:__main__:Function signature to fuzz: const char * _cupsGetPassword
(const char *)
INFO:__main__:Length of potential targets: 1
```

OSS-Fuzz-Gen: Example in OpenPrinting

- **Function**
  - Function Name
  - Parameter Type
  - Return Type
  - Signature
- **Project Language**
- **Fuzz Target Name**
- **Fuzzer Source Code Path**



```
"functions":
- "name": "_cupsGetPassword"
  "params":
  - "name": ""
    "type": "bool "
  "return_type": "void"
  "signature": "const char * _cupsGetPassword(const char *)"
"is_test_benchmark": false
"language": "c++"
"project": "libcups"
"target_name": "fuzzipp"
"target_path": "/src/fuzzing/projects/libcups/fuzzer/fuzzipp.c"
```

Use LLM for Fuzz Driver Generation

- **Prompt Design**
  - Task Description, Library Information, Example Code, API Specifications, Error Building Information, Errored Code
- **Promising Temperature**
  - Around 0.5
- **Backbone LLM**

GPT - 4 Gemini Claude deepseek coder

How Effective Are They? Exploring Large Language Model Based Fuzz Driver Generation

OSS-Fuzz Integration Bounty

● **OSS-Fuzz Initial Integration**	Up to $5,000
● **Ideal Fuzzing Integration**	Up to $15,000
○ CI Fuzz	
○ Coverage > 50%	
○ > 2 fixed bugs	
● **Line Coverage Improvements**	Up to $1,000 per 10% increase
● **Fuzz Introspector**	Up to $5,000 per project
● **New Sanitizer, Impactful Vulnerability Detected, FuzzBench Integration Reward**	Up to $11,337

Discussion

Towards Intelligent OpenPrinting Security Analysis

- Threat Model, Characterized Attack Surfaces
- Upstream/Downstream Vulnerability Tracing
- Automated Exploit Generation and Evolution
- Fuzzer Prioritization and False Positive Result Elimination

Contribute to OpenPrinting Security

- Do not hesitate to contact developers to join testing development
- Highly recommend to join the OSS-Fuzz and OSS-Fuzz-Gen integration
- Security auditing / research for OpenPrinting projects

Thanks You !

Project Link: https://github.com/OpenPrinting/fuzzing

Feel free to ask questions or contact me after the conference :)

Contacts: Jiongchi Yu (ttfish@ttfish.cc)