

# Fuzzing with AFL++

Moonzoo Kim

School of Computing

KAIST

# Introduction to AFL++

- American Fuzzy Lop plus plus (AFL++)
  - Most actively maintained greybox fuzzing tool (i.e., coverage-guided fuzzing tool)
    - A fork of Google's American Fuzzy Lop (AFL is not updated since 2017)
  - <https://github.com/AFLplusplus/AFLplusplus>
  - Supports C, C++ and Objective C

- Fuzzing with AFL++

- Instrumenting target
- Collect input testcases
- Run fuzzing
- Triage the result

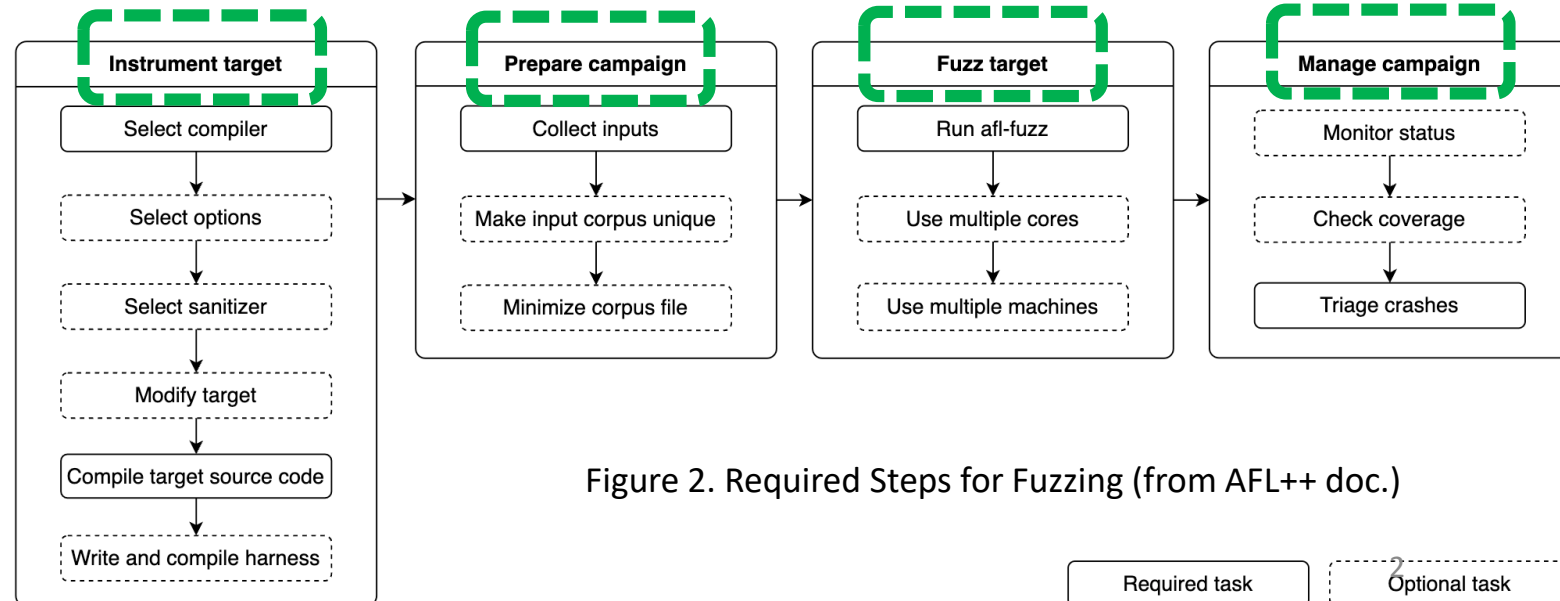



Figure 2. Required Steps for Fuzzing (from AFL++ doc.)



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  last new path : 0 days, 0 hrs, 0 min, 1 sec
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 261*1 (37.1%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 14
  stage execs : 31/32 (96.88%)
  total execs : 2.55M
  exec speed : 61.2k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  havoc/splice : 506/1.05M, 193/1.44M
  py/custom : 0/0, 0/0
  trim : 19.25%/53.2k, n/a
map coverage
  map density : 5.78% / 13.98%
  count coverage : 3.30 bits/tuple
findings in depth
  favored paths : 114 (16.22%)
  new edges on : 167 (23.76%)
  total crashes : 0 (0 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
[cpu000: 12%]
```

Figure 1. AFL++ Status Screen



# Step 1. Target Instrumentation

- AFL++ inserts probes to capture (approximate) branch coverage by obtaining branch hit-counts
  - **Tuple** := (from basic block, to basic block), equivalent to a branch.
  - Tuples in A -> B -> C -> D -> E : {(A, B), (B, C), (C, D), (D, E)}
- At each branch point, AFL++ compiler injects:
  - `shared_mem[branch_id]` – Branch hit count map
    - 64KB shared memory region, fits in L2 cache (more entries in AFL++)
    - Every byte indicates hit counts for (branch\_src, branch\_dst)
  - `branch_id` is obtained by bitwise operation on information on basic\_blocks
- (new) **Path** := It covers a new tuple, or a branch's hit count is in a new range of hit counts
  - Recall that AFL uses eight buckets [1, 2, 3, 4-7, 8-15, 16-31, 32-127, 128+].

Branch cnt	Colliding tuples	Example targets
1,000	0.75%	giflib, lzo
2,000	1.5%	zlib, tar, xz
5,000	3.5%	libpng, libwebp
10,000	7%	libxml
20,000	14%	sqlite
50,000	30%	•

Figure 3. Hit count map collision in AFL



# Step 1. Target Instrumentation

- AFL++ provides several compilers to instrument target.
  - afl-clang-lto / afl-clang-lto++ (Link Time Optimization mode, available for LLVM ver. >13)
  - afl-clang-fast / afl-clang-fast++ (LLVM mode)
  - afl-gcc-fast / afl-gcc-fast++ (GCC\_PLUGIN mode)
  - afl-gcc / afl-g++ (GCC mode)
- afl-clang-lto is fastest ([reference](#))

## Examples

- grep (Homework)  
`afl-clang-lto grep.c -o grep`
- Applying AFL++ in Makefile  
`CC=afl-clang-lto CXX=afl-clang-lto++ ./configure --disable-shared`
- Applying ALF++ in CMake  
`mkdir build; cd build; \  
cmake -DCMAKE_C_COMPILER=afl-clang-lto -DCMAKE_CXX_COMPILER=afl-clang-lto++ ..`



## Step 2. Input Preparation and Fuzzing

- AFL++ fuzzer explores input file space by mutating an input file.
  - “@@” for input file placeholder
    - If there is no “@@”, AFL++ fuzzer explores standard input.

```
afl-fuzz -i ./input_seeds -o ./out -- pdftotext @@ out.txt
```

input file directory    output directory    command line input

- In this case, `input_seeds` should contain pdf files as initial seed input files.
- Fuzzer runs `pdftotext <input_file> out.txt` with various `<input_file>`, starting with the files in `./input_seeds`.



## Step 3. Fuzz Target

- To see the full list of options for fuzzing, check `afl-fuzz --help`.
  - `-s seed` : use a fixed seed for the RNG
  - `-V seconds` : fuzz for a specified time then terminate
  - `-g minlength, -G maxlength` : set min/max length of generated fuzz input in bytes
  - `-x dict_file` : dictionary input (optional)
  - and more...
- Ctrl + c (SIGINT) for stop fuzzing
- Hint: run afl-fuzz in screen or tmux shell to prepare unexpected log-off during long fuzzing process (~24 hours)

# Step 3. Fuzz Target (env. variables)



LD\_BIND\_LAZY: do not set LD\_BIND\_NOW env var for target

ASAN\_OPTIONS: custom settings for ASAN

(must contain abort\_on\_error=1 and symbolize=0)

MSAN\_OPTIONS: custom settings for MSAN

(must contain exitcode=86 and symbolize=0)

AFL\_AUTORESUME: resume fuzzing if directory specified by -o already exists

AFL\_BENCH\_JUST\_ONCE: run the target just once

AFL\_BENCH\_UNTIL\_CRASH: exit soon when the first crashing input has been found

AFL\_CMPLOG\_ONLY\_NEW: do not run cmplog on initial testcases (good for resumes!)

AFL\_CRASH\_EXITCODE: optional child exit code to be interpreted as crash

AFL\_CUSTOM\_MUTATOR\_LIBRARY: lib with afl\_custom\_fuzz() to mutate inputs

AFL\_CUSTOM\_MUTATOR\_ONLY: avoid AFL++'s internal mutators

AFL\_CYCLE\_SCHEDULES: after completing a cycle, switch to a different -p schedule

AFL\_DEBUG: extra debugging output for Python mode trimming

AFL\_DEBUG\_CHILD: do not suppress stdout/stderr from target

AFL\_DISABLE\_TRIM: disable the trimming of test cases

AFL\_DUMB\_FORKSRV: use fork server without feedback from target

AFL\_EXIT\_WHEN\_DONE: exit when all inputs are run and no new finds are found

AFL\_EXIT\_ON\_TIME: exit when no new coverage is found within the specified time

AFL\_EXPAND\_HAVOC\_NOW: immediately enable expand havoc mode (default: after 60 minutes and a cycle without finds)

AFL\_FAST\_CAL: limit the calibration stage to three cycles for speedup

AFL\_FORCE\_UI: force showing the status screen (for virtual consoles)

AFL\_FORKSRV\_INIT\_TMOUT: time spent waiting for forkserver during startup (in ms)

AFL\_HANG\_TMOUT: override timeout value (in milliseconds)

AFL\_I\_DONT\_CARE\_ABOUT\_MISSING\_CRASHES: don't warn about core dump handlers

AFL\_IGNORE\_PROBLEMS: do not abort fuzzing if an incorrect setup is detected

AFL\_IGNORE\_TIMEOUTS: do not process or save any timeouts

AFL\_IGNORE\_UNKNOWN\_ENVS: don't warn on unknown env vars

AFL\_IMPORT\_FIRST: sync and import test cases from other fuzzer instances first

AFL\_INPUT\_LEN\_MIN/AFL\_INPUT\_LEN\_MAX: like -g/-G set min/max fuzz length produced

AFL\_PIZZA\_MODE: 1 - enforce pizza mode, 0 - disable for April 1st

AFL\_KILL\_SIGNAL: Signal ID delivered to child processes on timeout, etc.  
(default: SIGKILL)

AFL\_FORK\_SERVER\_KILL\_SIGNAL: Kill signal for the fork server on termination  
(default: SIGTERM). If unset and AFL\_KILL\_SIGNAL is set, that value will be used.

AFL\_MAP\_SIZE: the shared memory size for that target. must be >= the size the target was compiled for

AFL\_MAX\_DET\_EXTRAS: if more entries are in the dictionary list than this value then they are randomly selected instead all of them being used. Defaults to 200.

AFL\_NO\_AFFINITY: do not check for an unused cpu core to use for fuzzing

AFL\_TRY\_AFFINITY: try to bind to an unused core, but don't fail if unsuccessful

AFL\_NO\_ARITH: skip arithmetic mutations in deterministic stage

AFL\_NO\_AUTODICT: do not load an offered auto dictionary compiled into a target

AFL\_NO\_CPU\_RED: avoid red color for showing very high cpu usage

AFL\_NO\_FORKSRV: run target via execve instead of using the forkserver

AFL\_NO\_SNAPSHOT: do not use the snapshot feature (if the snapshot lkm is loaded)

AFL\_NO\_STARTUP\_CALIBRATION: no initial seed calibration, start fuzzing at once

AFL\_NO\_UI: switch status screen off

AFL\_PATH: path to AFL support binaries

AFL\_PYTHON\_MODULE: mutate and trim inputs with the specified Python module

AFL\_QUIET: suppress forkserver status messages

AFL\_PRELOAD: LD\_PRELOAD / DYLD\_INSERT\_LIBRARIES settings for target

AFL\_TARGET\_ENV: pass extra environment variables to target

AFL\_SHUFFLE\_QUEUE: reorder the input queue randomly on startup

AFL\_SKIP\_BIN\_CHECK: skip afl compatibility checks, also disables auto map size

AFL\_SKIP\_CPUFREQ: do not warn about variable cpu clocking

AFL\_STATSD: enables StatsD metrics collection

AFL\_STATSD\_HOST: change default statsd host (default 127.0.0.1)

AFL\_STATSD\_PORT: change default statsd port (default: 8125)

AFL\_STATSD\_TAGS\_FLAVOR: set statsd tags format (default: disable tags)

Supported formats are: 'dogstatsd', 'librato',  
'signalfx' and 'influxdb'

AFL\_SYNC\_TIME: sync time between fuzzing instances (in minutes)

AFL\_NO\_CRASH\_README: do not create a README in the crashes directory

AFL\_TESTCACHE\_SIZE: use a cache for testcases, improves performance (in MB)

AFL\_TMPDIR: directory to use for input file generation (ramdisk recommended)

AFL\_EARLY\_FORKSERVER: force an early forkserver in an afl-clang-fast/  
afl-clang-lto/afl-gcc-fast target

AFL\_PERSISTENT: enforce persistent mode (if \_\_AFL\_LOOP is in a shared lib

AFL\_DEFER\_FORKSRV: enforced deferred forkserver (\_\_AFL\_INIT is in a .so)

Compiled with Python 3.6.9 module support, see docs/custom\_mutator.md

Compiled without AFL\_PERSISTENT\_RECORD support.

Compiled with shmat support.

For additional help please consult /usr/local/share/doc/afl/README.md :)



# AFL++ Status Screen (1/5)

```
american fuzzy lop ++4.06a {default} (...install_afl++/bin/pdftotext) [fast]
- process timing
  run time : 0 days, 0 hrs, 1 min, 7 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 23 sec
  last saved hang : none seen yet
- cycle progress
  now processing : 897.0 (79.7%)
  runs timed out : 0 (0.00%)
- stage progress
  now trying : havoc
  stage execs : 4794/25.6k (18.73%)
  total execs : 112k
  exec speed : 1703/sec
- fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : havoc mode
  havoc/splice : 986/77.1k, 108/11.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 2.69%/11.7k, disabled
- map coverage
  map density : 5.87% / 12.10%
  count coverage : 3.74 bits/tuple
- findings in depth
  favored items : 192 (17.07%)
  new edges on : 288 (25.60%)
  total crashes : 3 (3 saved)
  total tmouts : 1 (0 saved)
- item geometry
  levels : 4
  pending : 1120
  pend fav : 190
  own finds : 1122
  imported : 0
  stability : 100.00%
[cpu000: 8%]
```

## 1) Process timing

- **run time**: running time
- **last new find**: how much time has elapsed since its most recent path finds
  - something wrong if no new path is found for several minutes after start
- **last saved crash**: how much time has elapsed since most recent crash finds.
- **last saved hang**: how much time has elapsed since most recent hang finds (default timeout: 1 sec)





# AFL++ Status Screen (2/5)

```
american fuzzy lop ++4.06a {default} (...install_afl++/bin/pdftotext) [fast]
process timing
  run time : 0 days, 0 hrs, 1 min, 7 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 23 sec
  last saved hang : none seen yet
cycle progress
  now processing : 897.0 (79.7%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 4794/25.6k (18.73%)
  total execs : 112k
  exec speed : 1703/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : havoc mode
  havoc/splice : 986/77.1k, 108/11.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 2.69%/11.7k, disabled
map coverage
  map density : 5.87% / 12.10%
  count coverage : 3.74 bits/tuple
findings in depth
  favored items : 192 (17.07%)
  new edges on : 288 (25.60%)
  total crashes : 3 (3 saved)
  total tmouts : 1 (0 saved)
item geometry
  levels : 4
  pending : 1120
  pend fav : 190
  own finds : 1122
  imported : 0
  stability : 100.00%
[cpu000: 8%]
```

## 2) Overall results

- **cycles done:** the count of an entire pass through the queue so far.
- **corpus count:** # of unique test cases discovered so far
- **saved crashes:** # of unique crashes discovered so far
- **saved hangs:** # of hangs discovered so far

Crashes and hangs are considered "unique" if the associated execution paths (represented in the branch map) involve any state transitions not seen in previously-recorded faults.



# AFL++ Status Screen (3/5)

```
american fuzzy lop ++4.06a {default} (...install_afl++/bin/pdftotext) [fast]
process timing
  run time : 0 days, 0 hrs, 1 min, 7 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 23 sec
  last saved hang : none seen yet
cycle progress
  now processing : 897.0 (79.7%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 4794/25.6k (18.73%)
  total execs : 112k
  exec speed : 1703/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : havoc mode
  havoc/splice : 986/77.1k, 108/11.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 2.69%/11.7k, disabled
overall results
  cycles done : 0
  corpus count : 1125
  saved crashes : 3
  saved hangs : 0
map coverage
  map density : 5.87% / 12.10%
  count coverage : 3.74 bits/tuple
findings in depth
  favored items : 192 (17.07%)
  new edges on : 288 (25.60%)
  total crashes : 3 (3 saved)
  total tmouts : 1 (0 saved)
item geometry
  levels : 4
  pending : 1120
  pend fav : 190
  own finds : 1122
  imported : 0
  stability : 100.00%
[cpu000: 8%]
```

## 3) Map coverage

- **map density**: branch coverage of the current input / accumulated branch coverage of the entire inputs
- **count coverage**: the variability in tuple hit counts
  - 1~8 bits/tuple



# AFL++ Status Screen (4/5)

```
american fuzzy lop ++4.06a {default} (...install_afl++/bin/pdfotext) [fast]
process timing
  run time : 0 days, 0 hrs, 1 min, 7 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 23 sec
  last saved hang : none seen yet
cycle progress
  now processing : 897.0 (79.7%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 4794/25.6k (18.73%)
  total execs : 112k
  exec speed : 1703/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : havoc mode
  havoc/splice : 986/77.1k, 108/11.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 2.69%/11.7k, disabled
overall results
  cycles done : 0
  corpus count : 1125
  saved crashes : 3
  saved hangs : 0
map coverage
  map density : 5.87% / 12.10%
  count coverage : 3.74 bits/tuple
findings in depth
  favored items : 192 (17.07%)
  new edges on : 288 (25.60%)
  total crashes : 3 (3 saved)
  total tmouts : 1 (0 saved)
item geometry
  levels : 4
  pending : 1120
  pend fav : 190
  own finds : 1122
  imported : 0
  stability : 100.00%
[cpu000: 8%]
```

## 4) Stage process

- **now trying:** current fuzzing stage [\[stages\]](#)
  - havoc = mutation with random tweaks
- **total execs:** a global exec counter
- **exec speed:** current program execution speed

## 5) Findings in depth

- **favored items:** # of favored paths (% of favored paths/corpus count)
- **new edges on:** # of test inputs that reached higher edge coverage (% of such paths/corpus count)
- **total crashes, total timeouts**



# AFL++ Status Screen (5/5)

```
american fuzzy lop ++4.06a {default} (...install_afl++/bin/pdfotext) [fast]
process timing
  run time : 0 days, 0 hrs, 1 min, 7 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 23 sec
  last saved hang : none seen yet
cycle progress
  now processing : 897.0 (79.7%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 4794/25.6k (18.73%)
  total execs : 112k
  exec speed : 1703/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : havoc mode
  havoc/splice : 986/77.1k, 108/11.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 2.69%/11.7k, disabled
overall results
  cycles done : 0
  corpus count : 1125
  saved crashes : 3
  saved hangs : 0
map coverage
  map density : 5.87% / 12.10%
  count coverage : 3.74 bits/tuple
findings in depth
  favored items : 192 (17.07%)
  new edges on : 288 (25.60%)
  total crashes : 3 (3 saved)
  total tmouts : 1 (0 saved)
item geometry
  levels : 4
  pending : 1120
  pend fav : 190
  own finds : 1122
  imported : 0
  stability : 100.00%
[cpu000: 8%]
```

## 6) Path geometry

- **levels**: reached path depth
  - level 1: **Initial test inputs** supplied by a user
  - level 2: The test inputs that can be **derived from the level 1 test inputs** through fuzzing
  - level 3: The ones derived from the level 2 inputs
- **pending**: # of inputs that have not gone through any fuzzing yet.
- **pend fav**: # of favored inputs that have not gone through any fuzzing yet.
- **own finds**: # of new paths found
- **stability**: Consistency of observed traces
  - If a program always behaves the same for the same input data, it will earn a score of 100%.



# Interpreting Output

```
afl-fuzz -i ./input_seeds -o ./out -- pdftotext @@ out.txt
```

output directory

- queue/ - Input files for every distinct execution path + all the seeds given by the user.
- crashes/ - Unique crashing test cases.
- hangs/ - Unique timeout test cases. The default time limit is 1s.
- cmdline – The command line input executed
- fuzz\_bitmap – Internal bitmap data used in AFL++
- fuzzer\_setup – The full afl-fuzz command used for fuzzing
- fuzzer\_stats – Overall fuzzing stats seen in status screen
- plot\_data – Stats for each testcase



# Interpreting Output

- The generated input files are stored in queue/, hang/ crashes/

id:004728,src:004510,time:1540784,execs:8407259,op:havoc,rep:4,+cov

testcase id

mutation source  
testcase id

accumulated time spent

nth executed.  
(similar to time spent)

mutation stage

Reached new branch

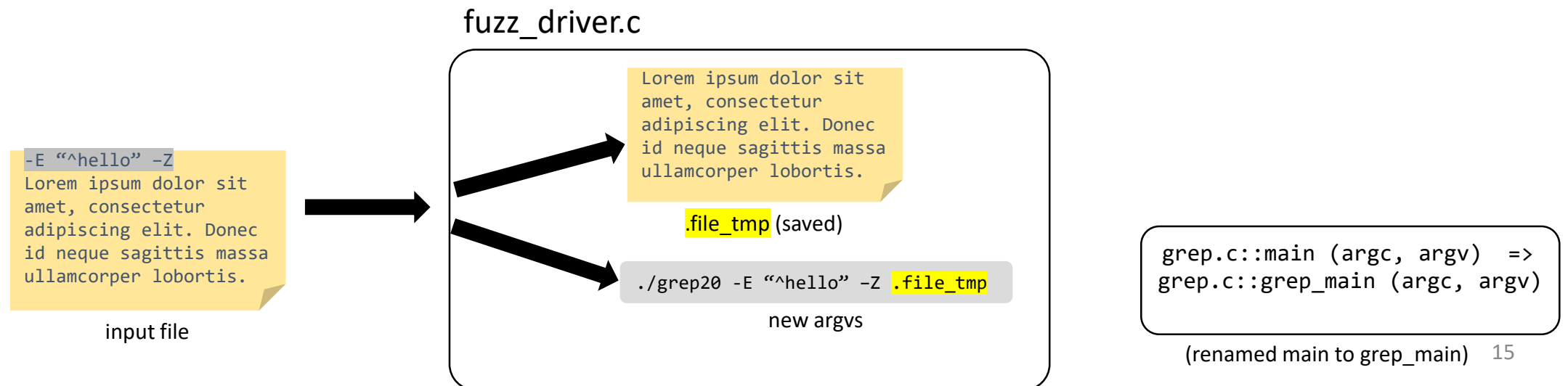
- Example. Running test case in crashes directory:

```
→ exercise01 ./pdftotext ./out/default/crashes/id:000000,sig:11,src:000000,time:20402,execs:32502,op:havoc,rep:16 out.txt
Error: PDF file is damaged - attempting to reconstruct xref table ...
Error (469): Illegal character <29> in hex string
Error (470): Illegal character <5d> in hex string
Error (471): Illegal character <2f> in hex string
Error (472): Illegal character <50> in hex string
Error (473): Illegal character <72> in hex string
Error (475): Illegal character <76> in hex string
Error (482): Illegal character '>'
Error (736): Dictionary key must be a name object
Error (738): Dictionary key must be a name object
Error (744): Dictionary key must be a name object
Error: Unterminated string
Error: End of file inside array
Error: End of file inside dictionary
Error (736): Dictionary key must be a name object
Error (738): Dictionary key must be a name object
Error (744): Dictionary key must be a name object
[1] 1875293 segmentation fault ./pdftotext out.txt
```

# Necessity of a Customized Fuzzing Driver



- By default, AFL++ does **not** mutate command line arguments
  - ex. AFL++ does not generate diverse executions on grep such as “grep **apple** abc.txt” or “grep **orange** abc.txt”
- You need to build a fuzzing driver to combine inputs of various types (e.g., command line arguments, network packets, GUI events, etc.) to an input file
  - ex. for grep, you should write a fuzzing driver to utilize a single input file that has both command-line arguments and the input file
- Fuzzing driver is widely used in various domain like library fuzzing.







# Target Instrumentation

```
prev = None
while(...) {
    (current, prev logic)
    prev = current
}
[ 1, 2, 3, 4, 5, 6 ]
```



```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]
prev_location = cur_location >> 1;
```

- Captures (approximate) branch coverage along with branch hit-counts.
  - **Tuple** := (from basic block, to basic block), equivalent to a branch.
  - Tuples in  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$  :  $\{(A, B), (B, C), (C, D), (D, E)\}$
- At each branch point, AFL++ compiler injects:
  - `shared_mem[branch_id]` – Branch hit count map
    - 64KB shared memory region, fits in L2 cache (more entries in AFL++)
    - Every byte indicates hit counts for  $(branch\_src, branch\_dst)$
  - (Assumption) `cur_location ^ prev_location` is unique for each tuple.
    - However, this assumption does not hold in general. (Figure 3)
    - Therefore, **branch coverage in AFL++  $\neq$  Actual branch coverage using gcov**
  - **Why `prev_location` is shifted by 1?**
    - Preserve directionability of tuples ( without this,  $(A, B)$  is indistinguishable from  $(B, A)$  )
    - Collision avoidance for tight loops ( without this,  $(A, A)$  is indistinguishable from  $(B, B)$  )
- (new) **Path** := It covers a new tuple, or a branch's hit count is in a new range of hit counts
  - Recall that AFL uses eight buckets [1, 2, 3, 4-7, 8-15, 16-31, 32-127, 128+].

```
cur_location = random();
shared_mem[cur_location ^ (prev_location >> 1)];
```

Branch cnt	Colliding tuples	Example targets
1,000	0.75%	giflib, lzo
2,000	1.5%	zlib, tar, xz
5,000	3.5%	libpng, libwebp
10,000	7%	libxml
20,000	14%	sqlite
50,000	30%	•

Figure 3. Hit count map collision in AFL