kAFL : Hardware-Assisted Feedback Fuzzing for OS Kernels

This paper is included in the Proceedings of the 26th USENIX Security Symposium

What is fuzzing?

1. insert an input file into a program
2. mutate the input (randomly?)
3. let the program open the input file
4. execute to see where is crashed

What is Feedback Fuzzing?

uses mechanisms to learn which inputs are interesting and which are not

The difficulty of fuzzing kernel

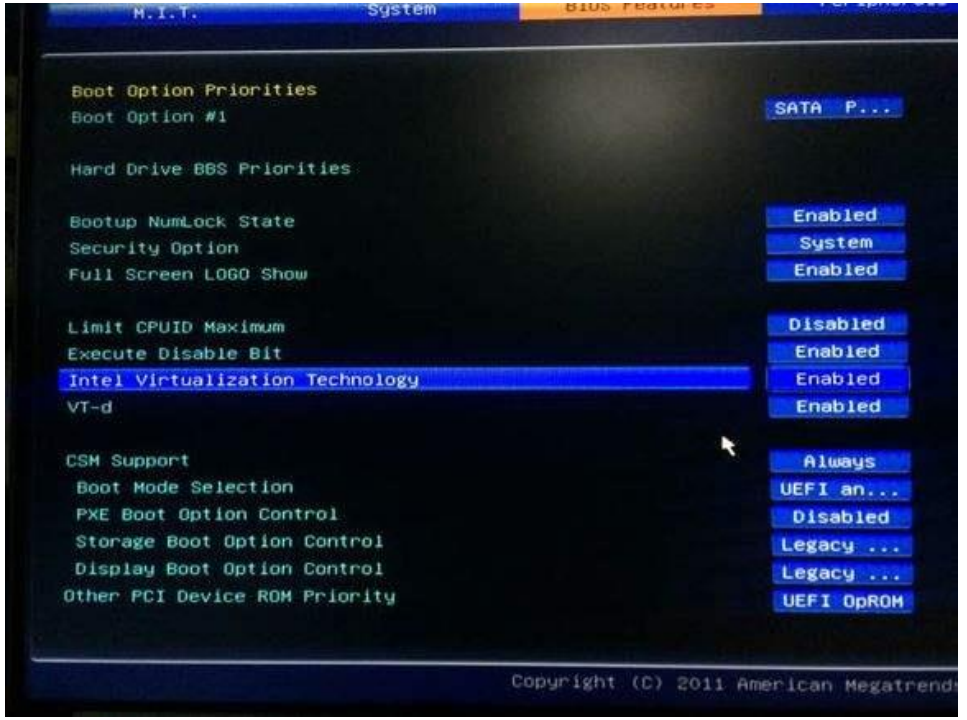
1. kernel-level code has significantly more non-determinism than the average ring 3 program—mostly due to interrupts, kernel threads, statefulness, and similar mechanisms.

2. crashes and timeouts mandate the use of virtualization to be able to catch faults and continue gracefull

3. close source

Basic Knowledge

- Intel CPU Privilege: ring 0, ring 1, ring 2, ring3. (ring 0 and ring 3 in windows). ring 0 -> kernel, ring 3 -> user space
- Intel VT-x
  - physical CPU
  - logical CPU
  - vCPU
    - VM
    - VMM



Basic Knowledge

- Intel PT

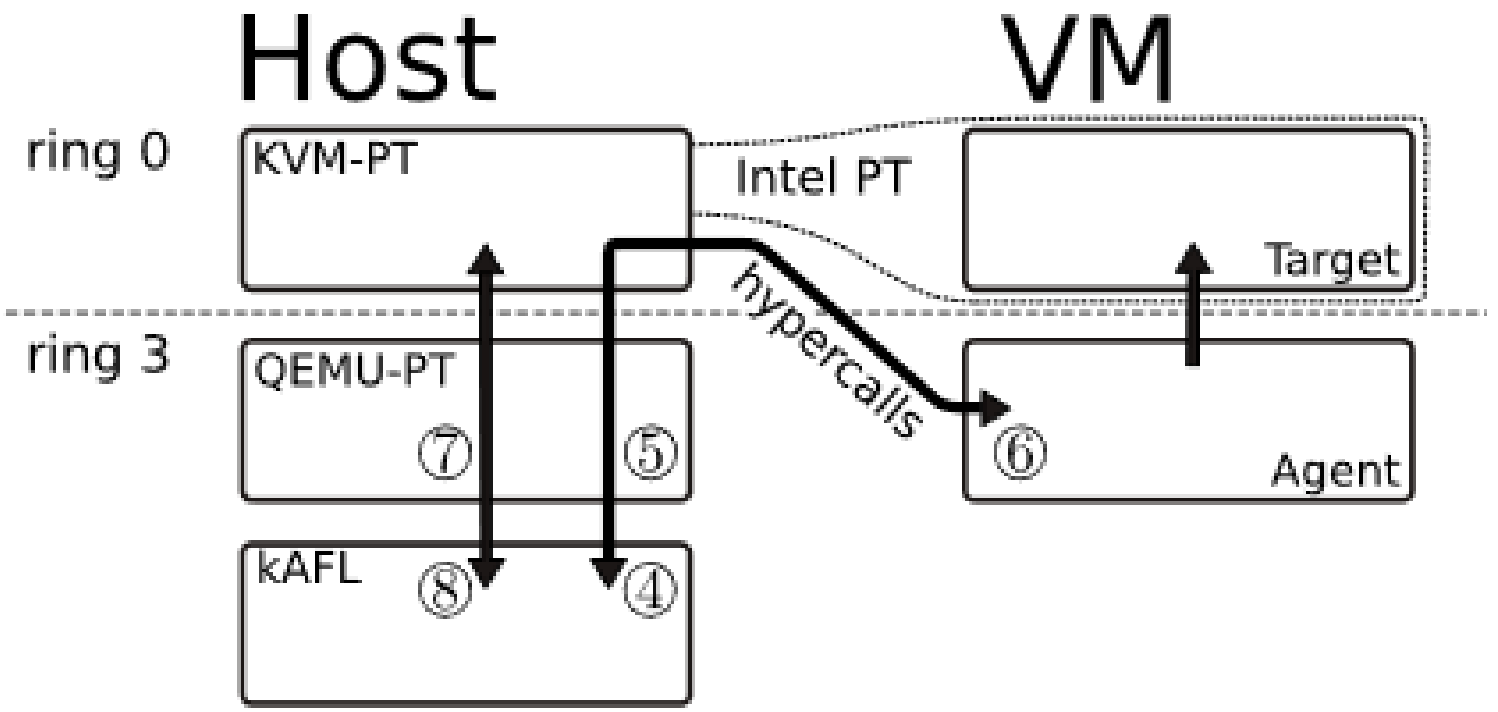
  With the fifth generation of Intel Coreprocessors, Intel PT provides execution and branch tracing information.

  general execution information and control flow information packets.

  - Taken-Not-Taken(TNT): e.g. jmp
  - Target IP (TIP): e.g. ret
  - Flow Update Packets(FUP): interrupts and traps
- CR3 filter

  distinguish kernel mode or user mode

Overview of KAFL



1. fuzzing logic
2. VM infrastructure
3. user mode agent (loader and agent)

Procedure

1. loader uses HC_SUBMIT_PANIC function to transfer panic handler address to QEMU-PT, in order to get information before crashing (modify the panic handler).
2. loader uses HC_GET_PROGRAM to request actual user agent, in order to initialize the fuzzing.
3. user agent uses HC_SUBMIT_CR3 function to KVM-PT to let VM give current CR3 to QEMU-PT；Finally trigger HC_SUBMIT_BUFFER to tell host the input location， then finish the initialization.
4. during the main loop, user agent uses HC_GET_INPUT function to get inputs.
5. fuzzer generates inputs to QEMU-PT， then QEMU-PT puts the inputs to the specific address，then trigger VM-ENTRY VM's execution；
6. VM-ENTRY also triggers PT tracer.
7. during fuzzing, QEMU-PT decodes tracing data, once get back to the user space, VM send signal HC_FINISHED, then trigger VM-EXIT to exit the VM.
8. The result is passed to the logic for further processing

KVM-PT

Why KVM-PT?

monitor vCPU instead of logical CPU

How to avoid unwanted trace data?

MSR autoload capabilities of Intel VT-x

How to save tracing?

ToPA

How to solve ToPA overflowing?

add a small buffer of variable length

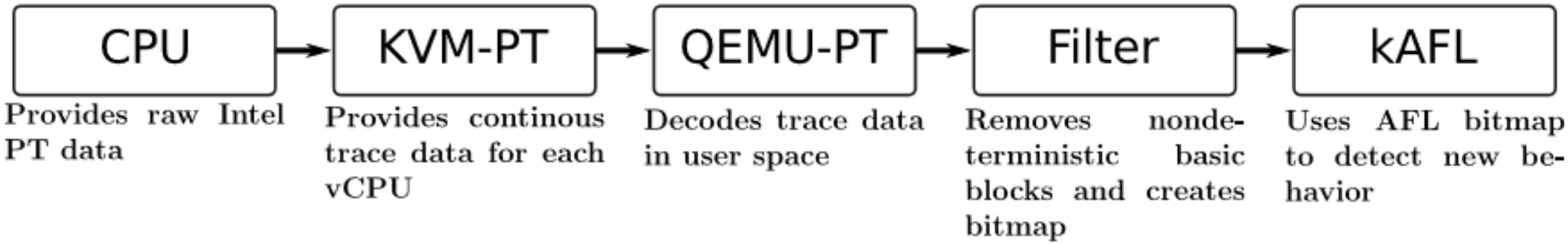
QEMU-PT

Why QEMU-PT?

a user space counterpart of KVM-PT, enable, disable and configure Intel PT at run time

decode the tracing data, and convert it to a bitmap

Procedure


CPU
KVM-PT
QEMU-PT
Filter
kAFL
Provides raw Intel PT data
Provides continous trace data for each vCPU
Decodes trace data in user space
Removes nonde-terministic basic blocks and creates bitmap
Uses AFL bitmap to detect new be-havior

Evaluation

Windows: NTFS Div-by-Zero

macOS: HFS Div-by-Zero

HFS Assertion Fail

HFS Use-After-Free

APFS Memory Corruption

Linux: keyctl Null Pointer Dereference

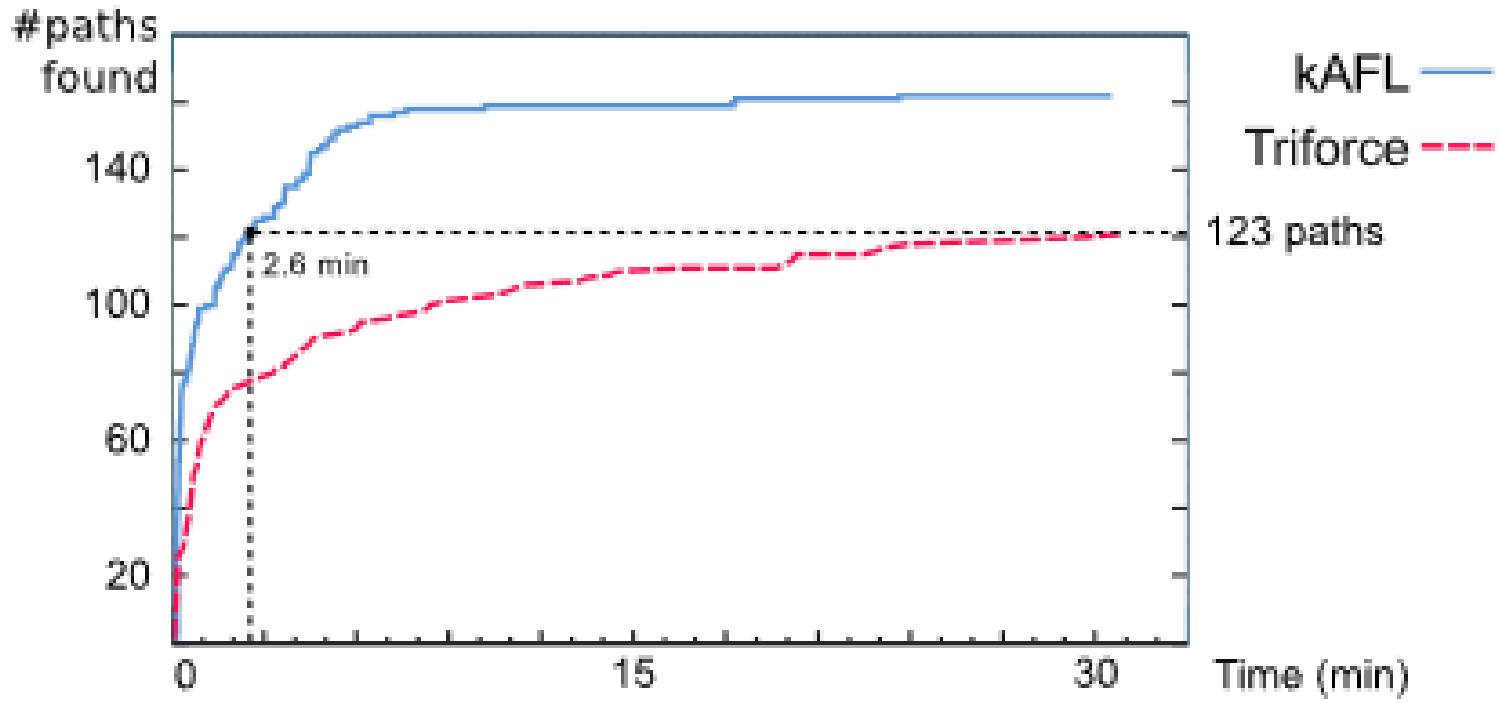
ext4 Memory Corruption

ext4 Error Handling

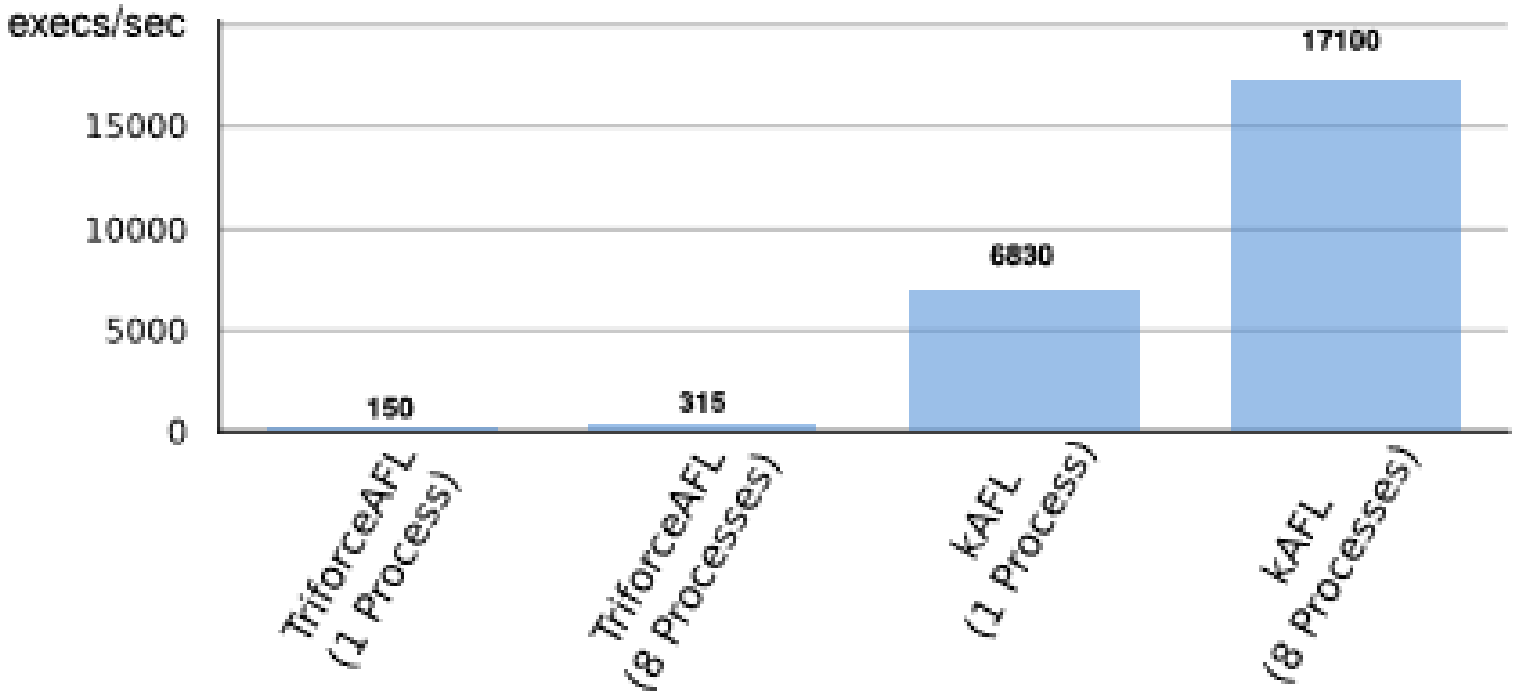
Evaluation

```
1  jsmn_parser parser;
2  jsmntok_t tokens[5];
3  jsmn_init(&parser);
4
5  int res=jsmn_parse(&parser, input, size, tokens, 5);
6  if(res >= 2){
7      if(tokens[0].type == JSMN_STRING){
8          int json_len = tokens[0].end - tokens[0].
               start;
9          int s = tokens[0].start;
10         if(json_len > 0 && input[s+0] == 'K'){
11         if(json_len > 1 && input[s+1] == 'A'){
12         if(json_len > 2 && input[s+2] == 'F'){
13         if(json_len > 3 && input[s+3] == 'L'){
14             panic(KERN_INFO "KAFL...\n");
15     }}}}}
16 }
```

Evaluation


#paths found
140
100
60
20
0
15
30
Time (min)
2.6 min
123 paths
kAFL
Triforce

Evaluation


execs/sec
15000
10000
5000
0
150
315
6830
17100
TriforceAFL (1 Process)
TriforceAFL (8 Processes)
kAFL (1 Process)
kAFL (8 Processes)

Limitation

- technique limitation
- decoding just-in-time code

APPENDDIX

The Introduction of AFL

Overview

AFL(American Fuzzing Loop) is an open-source fuzzing tool.

- It can fuzz software in user space
- It can fuzz open-source software or closed-source software

The use of AFL

open-sourced software:

gcc -----> afl-gcc

g++-----> afl-g++

clang -----> afl-clang

e.g CC=/path/to/afl/afl-gcc ./configure

The use of AFL

start fuzzing:

stdin:

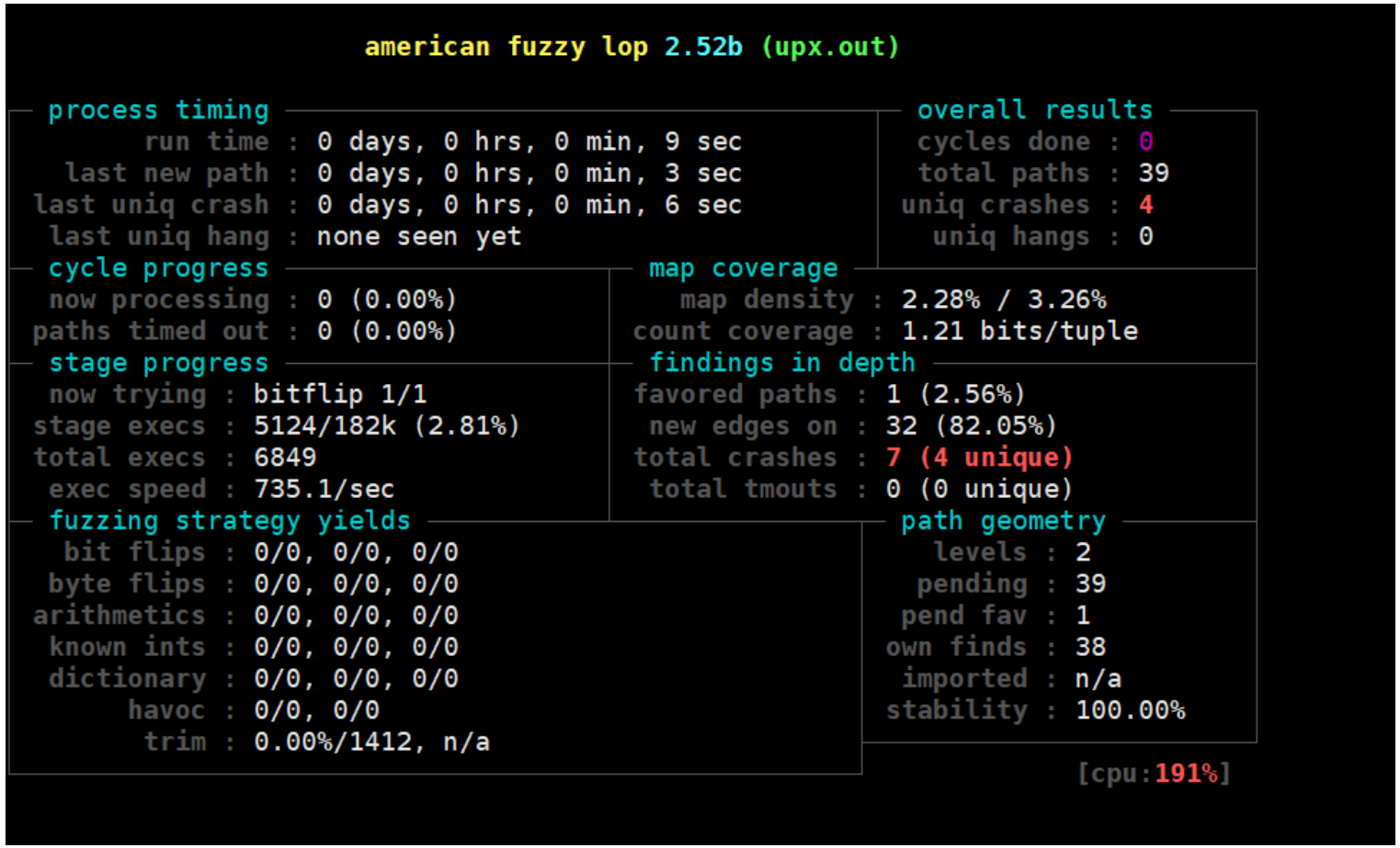
```
./afl-fuzz -i testcase_dir -o findings_dir /path/to/program […params…]
```

file:

```
./afl-fuzz -i testcase_dir -o findings_dir /path/to/program @@
```

Example

```
root@hwsrv-521890:~/upx/src# afl-fuzz -i ../../afl_in -o ../../afl_out ./upx.out @@
```



close-sourced: (poor efficiency)

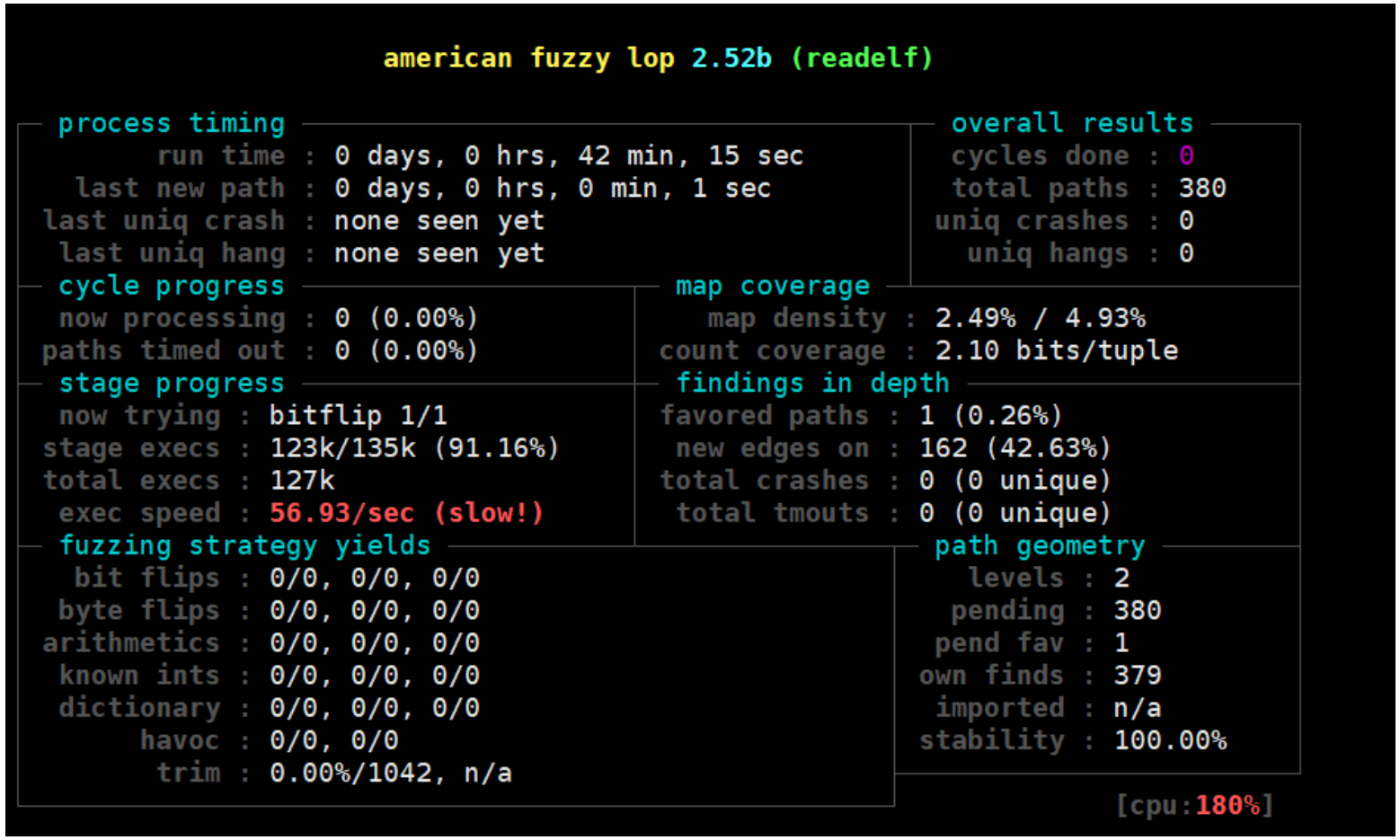
compile quem_mode

start fuzzing:

afl_fuzz -i afl_in -o afl_out **-Q** program -a @@

Example

```
root@hwsrv-521890:~/afl-master# ./afl-fuzz -i ../afl_in_2 -o ../afl_out_2 -Q readelf -a @@
```



Algorithm

1. Load user-supplied initial test cases into the queue,
2. Take next input file from the queue,
3. Attempt to trim the test case to the smallest size that doesn't alter the measured behavior of the program,
4. Repeatedly mutate the file using a balanced and well-researched variety of traditional fuzzing strategies,
5. If any of the generated mutations resulted in a new state transition recorded by the instrumentation, add mutated output as a new entry in the queue.
6. Go to 2.

Details

Why using afl-gcc/g++ instead of gcc/g++?

It will hook the branch instruction.

(after compiling the source code to assembly language, using afl-as)

Why hooking the branch instruction?

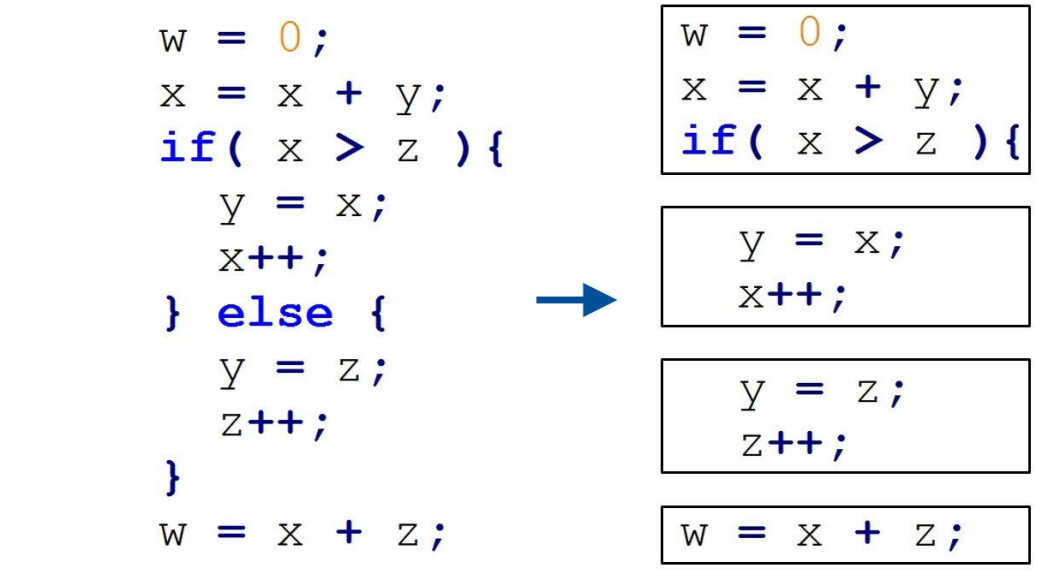
improving the coverage

calculating the count of hitting the branch

About Coverage

- Function
- Basic Block
- Edge

```
w = 0;
x = x + y;
if( x > z ){
   y = x;
   x++;
} else {
   y = z;
   z++;
}
w = x + z;
```



About Coverage

- tuple (last block + current block)

  e.g. A -> B -> C -> D -> E (tuples: AB, BC, CD, DE)

- how to save and find a new transition?

  count of branch execution ---> 8 bits

  using global map to save

- what is interesing?

  a new one

  the one from one bucket to another bucket

```
static const u8
count_class_lookup8[256] = {
  [0]          = 0,
  [1]          = 1,
  [2]          = 2,
  [3]          = 4,
  [4 ... 7]    = 8,
  [8 ... 15]   = 16,
  [16 ... 31]  = 32,
  [32 ... 127]  = 64,
  [128 ... 255] = 128
};
```

Mutation

- bitflip
- arithmetic
- interest
- dictionary
- havoc
- splice

Fork server

- fuzzer ----------> fork server

- copy on write