AFL Installation

American Fuzzy Lop (AFL).

Brief Introduction

AFL is developed by Michal Zalewski (lcamtuf@google.com). It is a brute-force fuzzer coupled with an exceedingly simple but rock-solid instrumentation-guided genetic algorithm. It uses a modified form of edge coverage to effortlessly pick up subtle, local-scale changes to program control flow.

1. Install AFL by entering the following commands in the terminal:

```
sudo apt update
sudo apt install afl
```

Alternatively, to build (experimental) QEMU mode:

1.1. Download latest version:

```
wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
```

1.2. To build:

```
tar xfz afl-latest.tgz && cd afl-*
sudo apt install bison
sudo apt install libtool-bin libglib2.0-dev automake
cd qemu_mode && ./build_qemu_support.sh
cd ..
```

1.3. To install

```
sudo make install
```

Update PATH, and set AFL_PATH; e.g.:

```
sudo gedit  /.bashrc
export AFL_PATH= /bin/afl
export PATH=$PATH:$AFL_PATH
```

2. Core files

Sending crash notifications to external utilities creates a significant delay between the actual crash and having this information relayed to AFL, which may misinterpret crashes as hangs.
Core dumps for Ubuntu 16.x and later versions, follow the steps below:

```
sudo systemctl stop apport.service
ulimit -c unlimited
```

Check the contents of `/proc/sys/kernel/core_pattern`; it should be `core`

3. CPU frequency scaling

Apparently, the scaling algorithm in the kernel is imperfect and can miss the short-lived processes spawned

by afl-fuzz.

3.1. To keep things moving, run these commands as root:
```
cd /sys/devices/system/cpu
echo performance | sudo tee cpu*/cpufreq/scaling_governor
```

3.2. Go back to the original state by replacing performance with ondemand or powersave; see:
```
/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_governors
```

4. Environment variables setting
After installing AFL but before you can use it effectively, you must set the following environment variables:
```
export AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1
export AFL_SKIP_CPUFREQ=1
```

5. Compile, Seeding and Run

5.1. To compile a C program:
```
afl-gcc -fno-stack-protector -z execstack test1.c -o test1
```

5.2. Seeding
Get a small but valid input file (create 'seed.txt' inside "seeds" folder and enter some meaningful values) that makes sense to the program. For example, if a C program has one e user defined integer variable to select test input, so insert one meaningful integer value in seed.txt file as seed.

5.3. Running AFL
```
afl-fuzz -i ./seeds/ -o ./results/ ./test1
```

Important options are as follows:

1. -i dir –input directory with test cases
2. -o dir – output directory for fuzzer findings
3. -t msec – timeout for each run (auto-scaled, 50-1000 ms)
4. -m megs – memory limit for child process (50 MB)
5. -M / -S id – distributed mode (see parallel_fuzzing.txt)
6. -C – crash exploration mode
7. timeout 600 – Prefix this option to command line for killing the process

For more detail please run *afl-fuzz –help*.

6. Outputs

6.1. The running of AFL shows in UI monitor.
6.2. Press "ctl + c" to terminate the process, otherwise AFL will run longer. The stopping criteria depends on user.

6.3. The newly discovered paths (or mutated valid test cases) are saved in folder “results/queue”. The hang test cases are saved in “results/hangs”. Also, all unique crash test cases are saved inside “results/crashed”.

For more information, please check the AFL Website.