fuzzing 툴 분석

1. Melkor fuzzer

> https://github.com/IOActive/Melkor_ELF_Fuzzer

Linux 시스템에서 ELF 파일에 대한 fuzzing을 위한 툴이다.

사용법

```
Usage: melkor <ELF metadata to fuzz><ELF file template>[-n num -l likelihood -q]

<ELF metadata to fuzz>:

-       -a: Autodetect (fuzz according to e_type except -H [the header])
-       -H: ELF header
-       -S: Selection Header Table
-       -P: Program Header Table
-       -D: Dynamic section
-       -s: Symbols Table(s)
-       -R: Relocations Table(s)
-       -N: Notes section
-       -Z: String Tables
-       -A: All of the above(except -a[Autodetect])
-       -B: All of the above(except -a[Autodetected] and -H[ELF Header])
-       -q: Quiet mode(doesn't print to STDOUT every executed fuzzing rule)
```

섹션 헤더

- SHT_NULL

  이 값은 섹션 헤더를 비활성으로 표시합니다. 관련 섹션이 없습니다. 섹션 헤더의 다른 멤버에는 정의되지 않은 값이 있습니다.

- SHT_PROGBITS

  섹션은 프로그램에 의해 정의 된 정보를 보유하며, 프로그램의 형식과 의미는 프로그램에 의해서만 결정됩니다.

- SHT_SYMTAB 과 SHT_DYNSYM

  <="" a="">이 섹션에는 기호 표가 있습니다. 현재 오브젝트 파일에는 각 유형마다 섹션이 하나만있을 수 있지만 이 제한 사항은 향후 완화 될 수 있습니다. 일반적으로 SHT_SYMTAB 링크 편집을위한 기호를 제공하지만 동적 링크에도 사용할 수 있습니다. 전체 심볼 테이블로, 동적 연결에 불필요한 많은 심볼을 포함 할 수 있습니다. 결과적으로, 오브젝트 파일은 SHT_DYNSYM 공간을 절약하기 위해 최소한의 동적 링킹 심볼 세트를 보유하는 섹션을 포함 할 수도 있습니다 . 오.

- SHT_STRTAB

  섹션에는 문자열 테이블이 있습니다. 객체 파일에는 여러 개의 문자열 테이블 섹션이있을 수 있습니다.

- SHT_RELA

섹션 Elf32_Rela은 오브젝트 파일의 32 비트 클래스에 Elf64_Rela대한 유형 또는 오브젝트 파일 의 64 비트 클래스에 대한 유형과 같은 명시적인 가수가있는 재배치 항목을 보유 합니다. 오브젝트 파일에는 여러 재배치 섹션이있을 수 있습니다.

- SHT_HASH

  섹션은 심볼 해시 테이블을 포함합니다. 현재 개체 파일에는 해시 테이블이 하나만있을 수 있지만이 제한은 나중에 완화 될 수 있습니다.

- SHT_DYNAMIC

  이 섹션에는 동적 연결에 대한 정보가 있습니다. 현재, 오브젝트 파일은 동적 섹션을 하나만 가질 수 있지만,이 제한은 장래에 완화 될 수 있습니다.

- SHT_NOTE

  이 섹션에는 파일을 어떤 식 으로든 표시하는 정보가 들어 있습니다.

- SHT_NOBITS

  이 유형의 섹션은 파일에서 공백을 차지하지 않지만 다른 경우에는 유사 SHT_PROGBITS합니다. 이 섹션에는 바이트가 없지만 sh_offset 멤버에는 개념적 파일 오프셋이 포함됩니다.

- SHT_REL

  이 섹션에는 Elf32_Rel32 비트 클래스의 오브젝트 파일에 Elf64_Rel대한 유형 또는 64 비트 클래스의 오브젝트 파일에 대한 유형과 같이 명시적인 가중치가없는 재배치 항목이 있습니다. 오브젝트 파일에는 여러 재배치 섹션이있을 수 있습니다.

- SHT_SHLIB

  이 섹션 유형은 예약되었지만 지정되지 않은 의미를가집니다.

- SHT_LOOS ...을 통하여 SHT_HIOS

  이 포함 범위의 값은 운영 체제 관련 의미론을 위해 예약되어 있습니다.

- SHT_LOPROC ...을 통하여 SHT_HIPROC

  이 포함 범위의 값은 프로세서 관련 의미론을 위해 예약되어 있습니다.

- SHT_LOUSER

  이 값은 응용 프로그램 용으로 예약 된 색인 범위의 하한을 지정합니다.

- SHT_HIUSER

  이 값은 응용 프로그램 용으로 예약 된 색인 범위의 상한을 지정합니다. 현재 또는 미래의 시스템 정의 섹션 유형과 충돌하지 않고 응용 프로그램이 사용하는 섹션 유형 SHT_LOUSER및 SHT_HIUSER사용할 수 있습니다.

실습

- 타깃 프로그램

  해당 프로그램은 오버플로우에 취약한 단순한 프로그램이다.

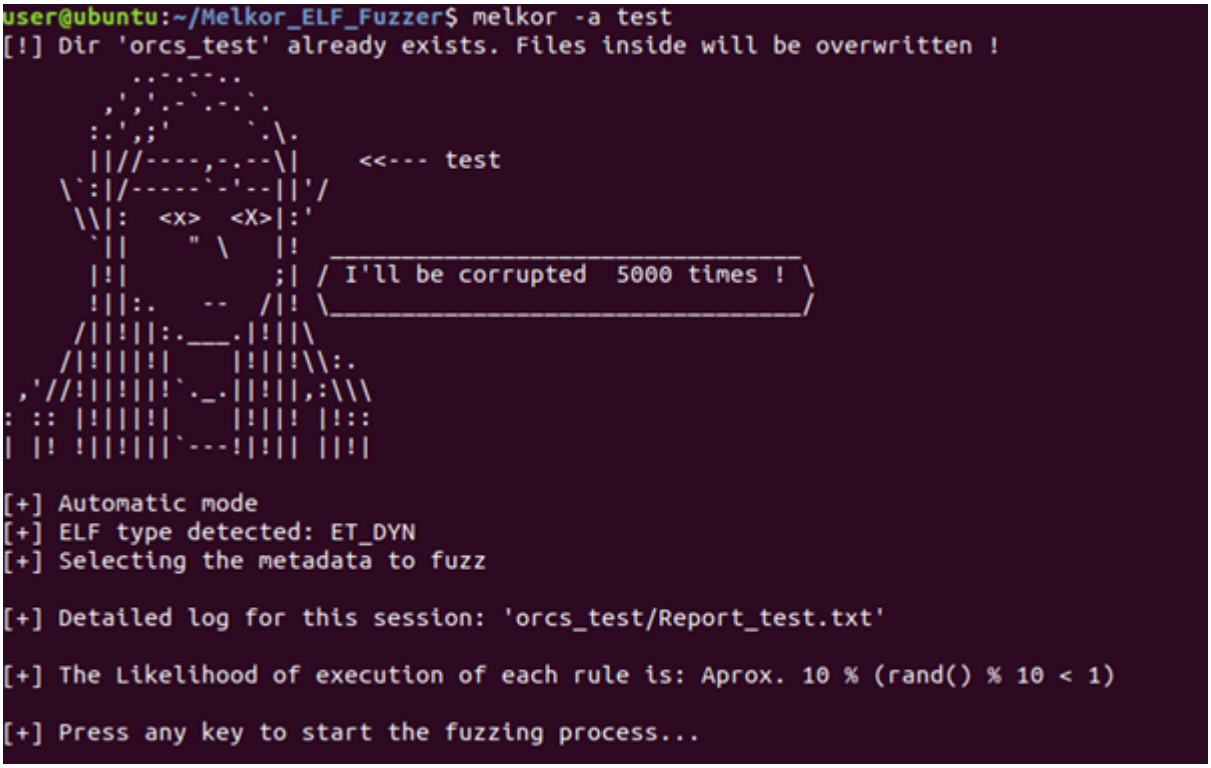
  - 소스코드

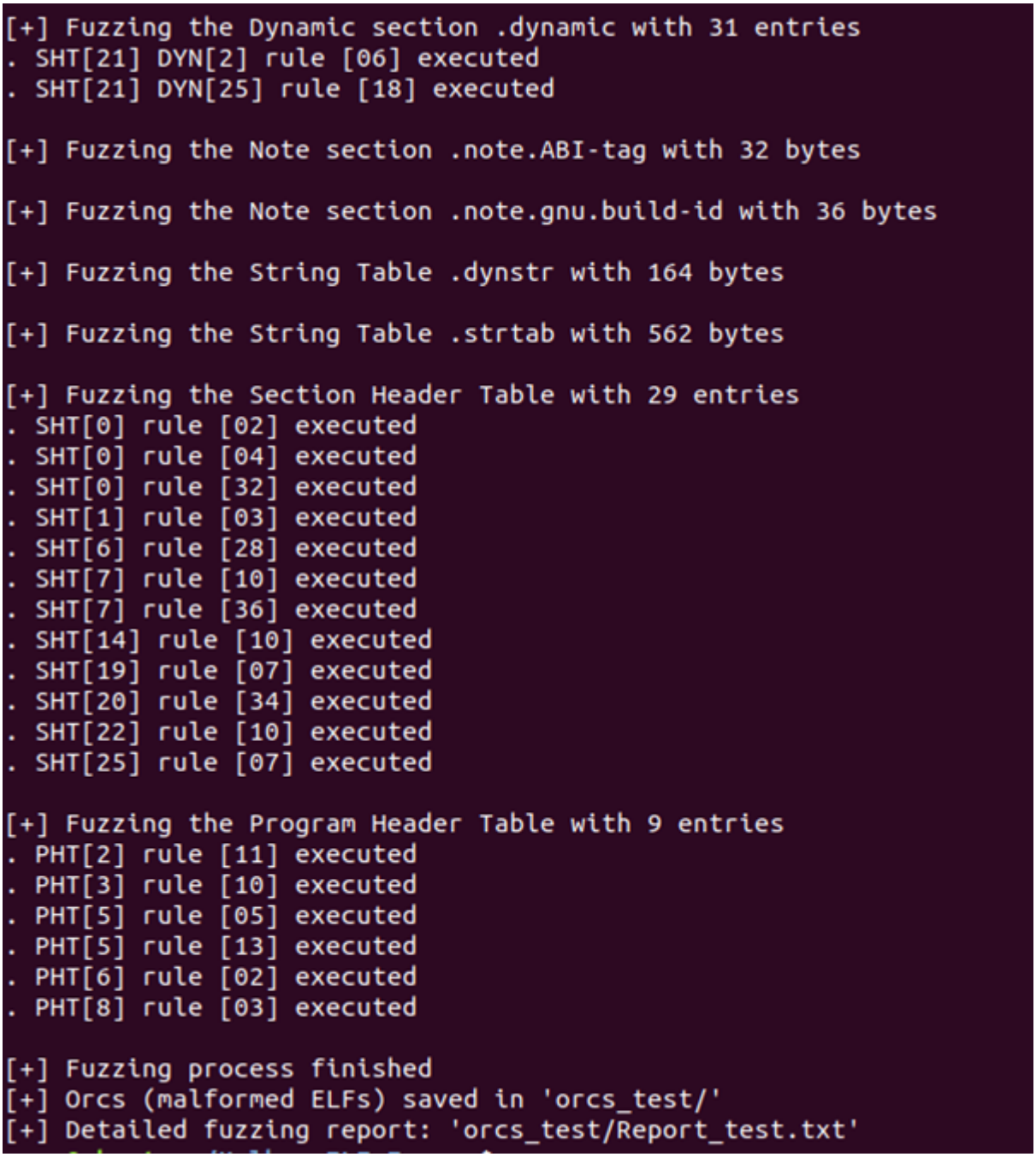
```c
#include <stdio.h>
#include <string.h>



void main(int argc, char *argv[])
{
        char buffer[256];
        strcpy(buffer, argv[1]);
        printf("%s\n", buffer);
}
```

- 실행 화면





실행 후 실행 결과인 report가 '/orcs_test/에 저장된 것을 확인할 수 있다.

하이라이트 쳐 둔 부분을 열어보면 아래와 같이 fuzzing한 결과를 알 수 있다.

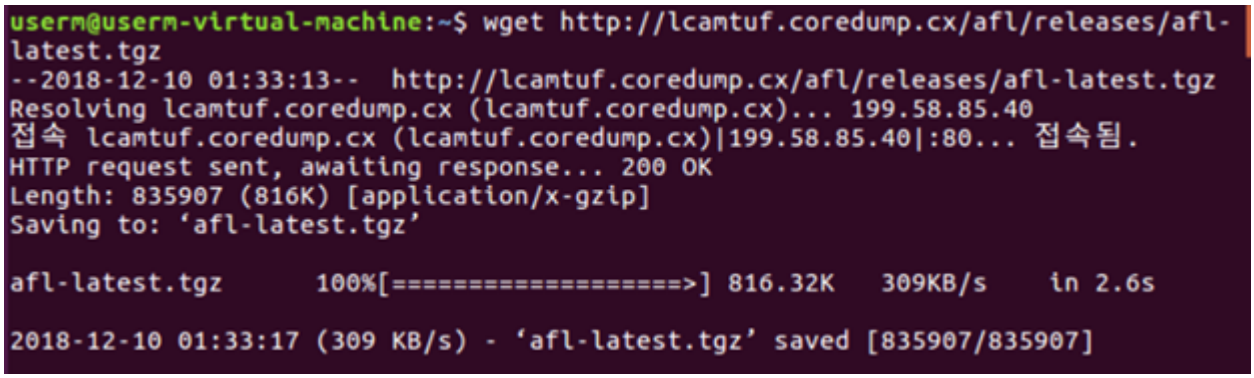
2. AFL(American Fuzzy Lop)

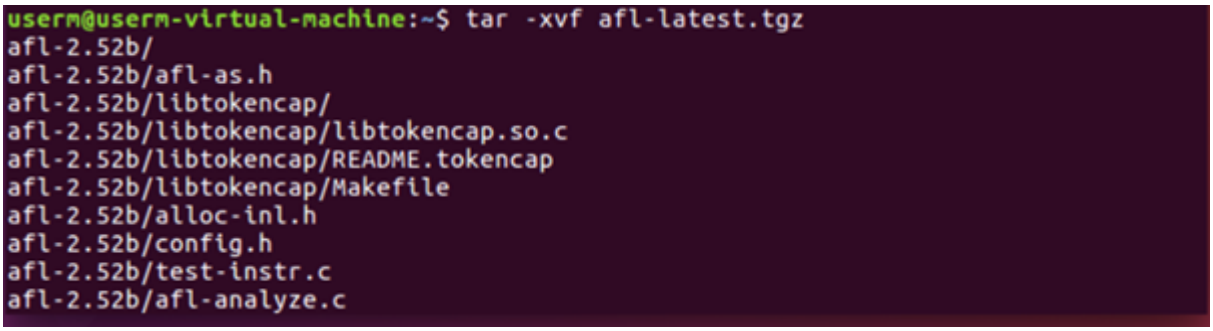
타킷 파일을 컴파일을 통해 퍼징을 하여 분석하는 프로그램, 코드를 필요로 함

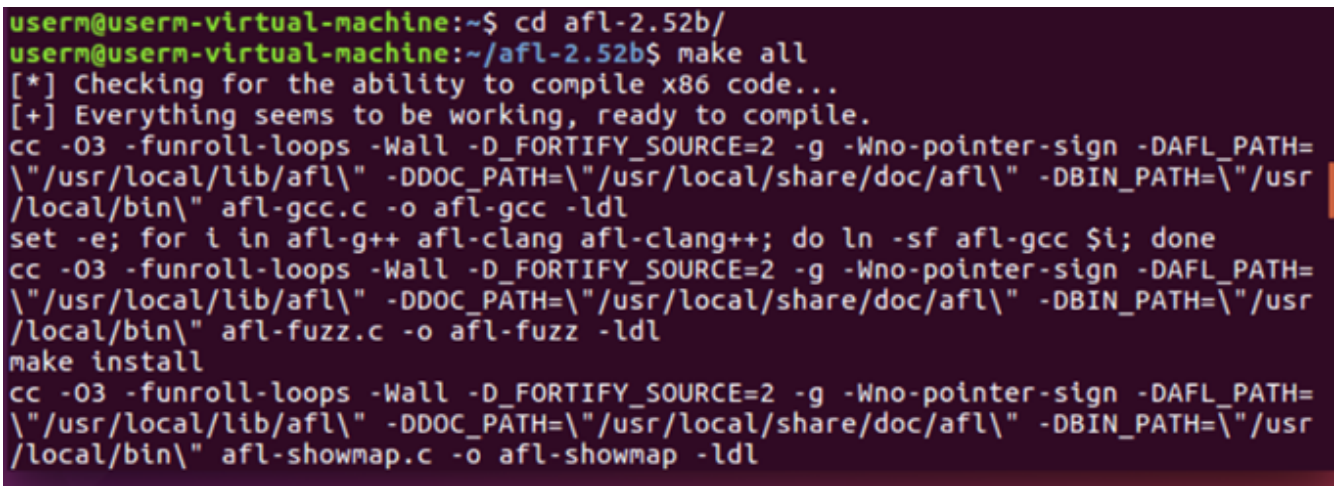
실습

1. 내려받기 & 설치

```
wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
tar -xvf afl-latest.tgz
cd afl-2.52b/
make all
sudo make install
```







```
userm@userm-virtual-machine:~/afl-2.52b$ sudo make install
[sudo] password for userm:
[*] Checking for the ability to compile x86 code...
[+] Everything seems to be working, ready to compile.
[*] Testing the CC wrapper and instrumentation output...
unset AFL_USE_ASAN AFL_USE_MSAN; AFL_QUIET=1 AFL_INST_RATIO=100 AFL_PATH=. ./afl
-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PAT
H=\"/usr/local/lib/afl\" -DDOC_PATH=\"/usr/local/share/doc/afl\" -DBIN_PATH=\"/u
sr/local/bin\" test-instr.c -o test-instr -ldl
echo 0 | ./afl-showmap -m none -q -o .test-instr0 ./test-instr
echo 1 | ./afl-showmap -m none -q -o .test-instr1 ./test-instr
[+] All right, the instrumentation seems to be working!
[+] All done! Be sure to review README - it's pretty short and useful.
```

2. AFL을 이용하여 타깃 프로그램 컴파일

```
./afl-gcc -o test-instr test-instr.c
```

```
userm@userm-virtual-machine:~/afl-2.52b$ ./afl-gcc -o test-instr test-instr.c
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 6 locations (64-bit, non-hardened mode, ratio 100%).
```

- 테스트 프로그램 소스 코드

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char** argv) {

  char buf[8];

  if (read(0, buf, 8) < 1) {
    printf("Hum?\n");
    exit(1);
  }

  if (buf[0] == '0')
    printf("Looks like a zero to me!\n");
  else
    printf("A non-zero value? How quaint!\n");

  exit(0);

}
```

3. 퍼징

```
sudo ./afl-fuzz -i ./testcases/others/text/ -o ./afl-out/ ./test-instr
```

```
usern@usern-virtual-machine:~/afl-2.52b$ ./afl-fuzz -i ./testcases/others/text/ -o ./afl-out/ ./test-instr
afl-fuzz 2.52b by <lcamtuf@google.com>
[+] You have 2 CPU cores and 3 runnable tasks (utilization: 150%).
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[*] Checking core_pattern...
[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning './testcases/others/text/'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Validating target binary...
[*] Attempting dry run with 'id:000000,orig:hello_world.txt'...
[*] Spinning up the fork server...
[+] All right - fork server is up.
    len = 6, map size = 4, exec speed = 2472 us
[+] All test cases processed.

[+] Here are some useful stats:

    Test case count : 1 favored, 0 variable, 1 total
       Bitmap range : 4 to 4 bits (average: 4.00 bits)
        Exec timing : 2472 to 2472 us (average: 2472 us)

[*] No -t option specified, so I'll use exec timeout of 20 ms.
[+] All set and ready to roll!
```

```
                american fuzzy lop 2.52b (test-instr)

┌─ process timing ─────────────────────┐┌─ overall results ─┐
│        run time : 0 days, 0 hrs, 0 min, 25 sec ││  cycles done : 22 │
│   last new path : 0 days, 0 hrs, 0 min, 25 sec ││  total paths : 2  │
│ last uniq crash : none seen yet      ││ uniq crashes : 0  │
│  last uniq hang : none seen yet      ││   uniq hangs : 0  │
├─ cycle progress ─────┬─ map coverage ─┴──────────────────┘
│  now processing : 1 (50.00%)  ││   map density : 0.01% / 0.01%
│ paths timed out : 0 (0.00%)   ││ count coverage : 1.00 bits/tuple
├─ stage progress ─────┼─ findings in depth ──────────────┐
│  now trying : havoc  ││ favored paths : 2 (100.00%)      │
│ stage execs : 290/384 (75.52%) ││  new edges on : 2 (100.00%)      │
│ total execs : 39.0k  ││ total crashes : 0 (0 unique)     │
│  exec speed : 1470/sec ││  total tmouts : 3 (1 unique)     │
├─ fuzzing strategy yields ────┴──────────────┐ path geometry ┐
│   bit flips : 0/56, 0/54, 0/50             ││    levels : 2 │
│  byte flips : 0/7, 0/5, 0/1                ││   pending : 0 │
│ arithmetics : 0/392, 0/21, 0/0             ││  pend fav : 0 │
│  known ints : 0/39, 0/140, 0/44            ││ own finds : 1 │
│  dictionary : 0/0, 0/0, 0/0                ││  imported : n/a │
│       havoc : 1/15.9k, 0/22.0k             ││ stability : 100.00% │
│        trim : 33.33%/1, 0.00%              ├┴──────────────┘
└────────────────────────────────────────────┘                [cpu000:128%]
```

4. 결과 확인

   지정 해줬던 afl-out 폴더에 결과가 저장된다.

```
cd afl-out
```

```
usern@usern-virtual-machine:~/afl-2.52b$ ls
Makefile            afl-as      afl-cmin    afl-gcc.c     afl-showmap    alloc-inl.h  docs          llvm_mode     types.h
QuickStartGuide.txt afl-as.c    afl-fuzz    afl-gotcpu    afl-showmap.c  as           experimental  qemu_mode
README              afl-as.h    afl-fuzz.c  afl-gotcpu.c  afl-tmin       config.h     hash.h        test-instr
afl-analyze         afl-clang   afl-g++     afl-out       afl-tmin.c     debug.h      libdislocator test-instr.c
afl-analyze.c       afl-clang++ afl-gcc     afl-plot      afl-whatsup    dictionaries libtokencap   testcases
usern@usern-virtual-machine:~/afl-2.52b$ cd afl-out
usern@usern-virtual-machine:~/afl-2.52b/afl-out$ ls
crashes  hangs  plot_data  queue
usern@usern-virtual-machine:~/afl-2.52b/afl-out$ cd crashes
usern@usern-virtual-machine:~/afl-2.52b/afl-out/crashes$ ls
usern@usern-virtual-machine:~/afl-2.52b/afl-out/crashes$ cd ..
usern@usern-virtual-machine:~/afl-2.52b/afl-out$ cd queue
usern@usern-virtual-machine:~/afl-2.52b/afl-out/queue$ ls
id:000000,orig:hello_world.txt
usern@usern-virtual-machine:~/afl-2.52b/afl-out/queue$
```

3. Radamsa

퍼징 테스팅 도구, 설치와 사용이 간단

설치

1. 내려받기 & 설치

```
git clone https://gitlab.com/akihe/radamsa
make
sudo make install # optional, you can also just grab bin/radamsa
```

2. 테스트

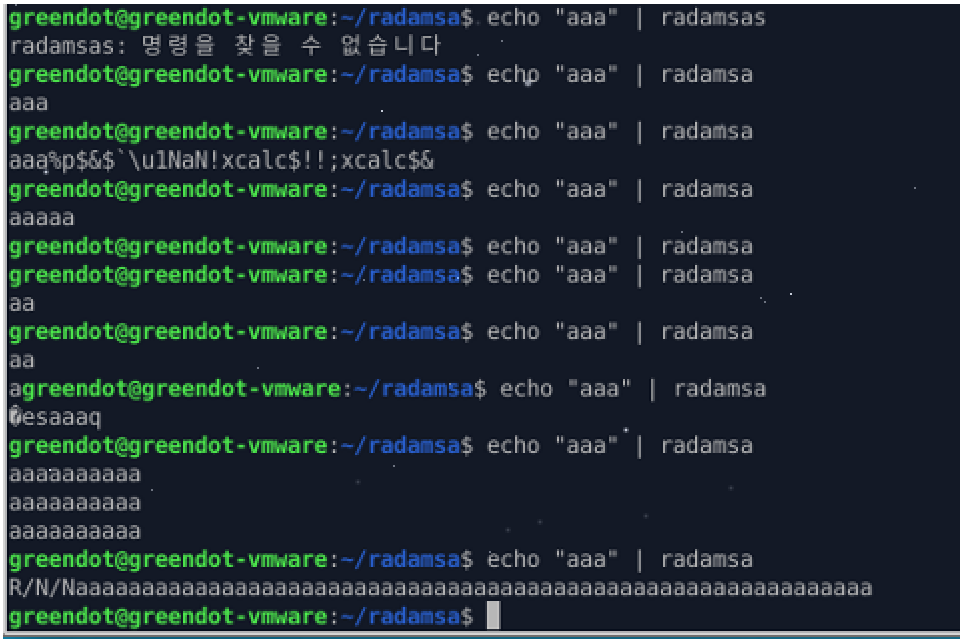
```
radamsa --help
```

```
greendot@greendot-vmware:~/radamsa$ radamsa --help
Usage: radamsa [arguments] [file ...]
  -h | --help, show this thing
  -a | --about, what is this thing?
  -V | --version, show program version
  -o | --output <arg>, output pattern, e.g. out.bin /tmp/fuzz-%n.%s, -, :80 or 127.0.0.1:
80 [-]
  -n | --count <arg>, how many outputs to generate (number or inf) [1]
  -s | --seed <arg>, random seed (number, default random)
  -m | --mutations <arg>, which mutations to use [ft=2,fo=2,fn,num=5,td,tr2,ts1,tr,ts2,ld
,lds,lr2,li,ls,lp,lr,lis,lrs,sr,sd,bd,bf,bi,br,bp,bei,bed,ber,uw,ui=2,xp=9,ab]
  -p | --patterns <arg>, which mutation patterns to use [od,nd=2,bu]
  -g | --generators <arg>, which data generators to use [random,file=1000,jump=200,stdin=
100000]
  -M | --meta <arg>, save metadata about generated files to this file
  -r | --recursive, include files in subdirectories
  -S | --seek <arg>, start from given testcase
  -d | --delay <arg>, sleep for n milliseconds between outputs
  -l | --list, list mutations, patterns and generators
  -C | --checksums <arg>, maximum number of checksums in uniqueness filter (0 disables) [
10000]
  -v | --verbose, show progress during generation
```

실습

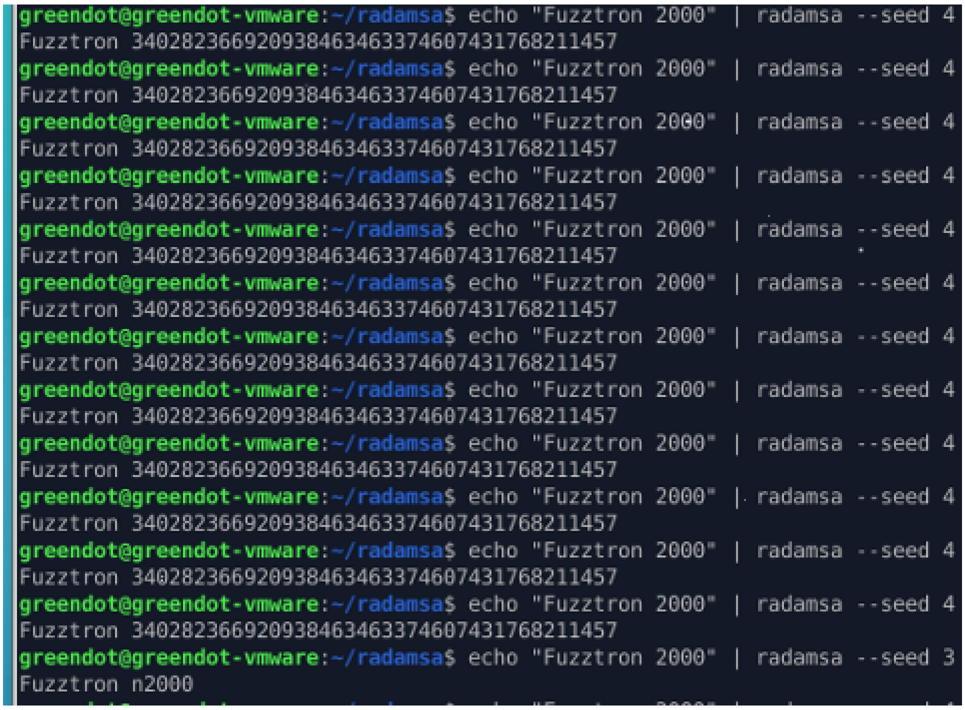
- 간단한 문자열 퍼징

```
echo "aaa" | radamsa
```




```
echo "Fuzztron 2000" | radamsa --seed 4
```

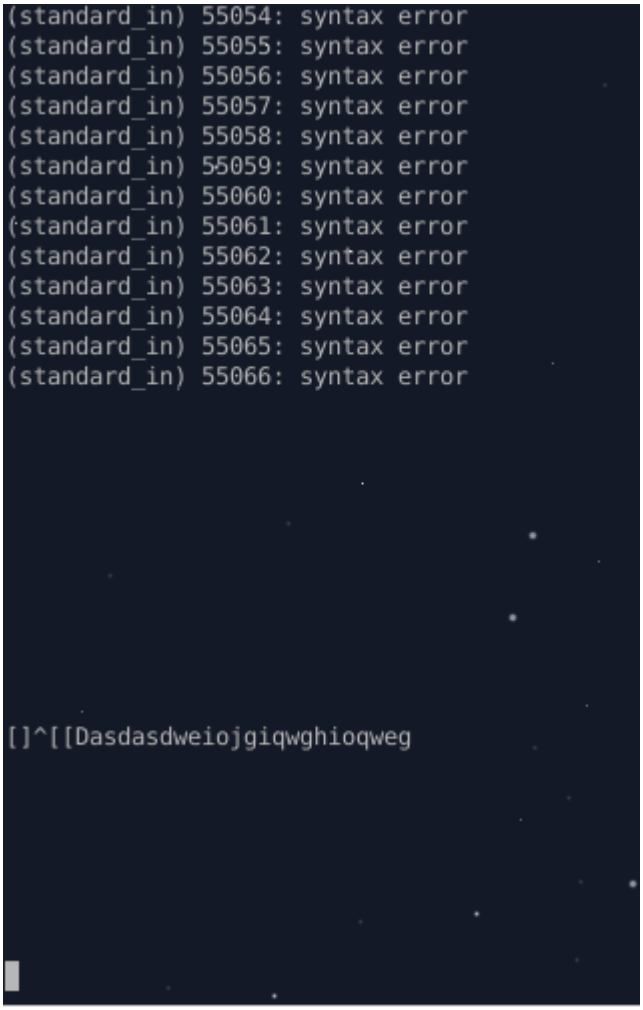
시드도 설정이 가능함



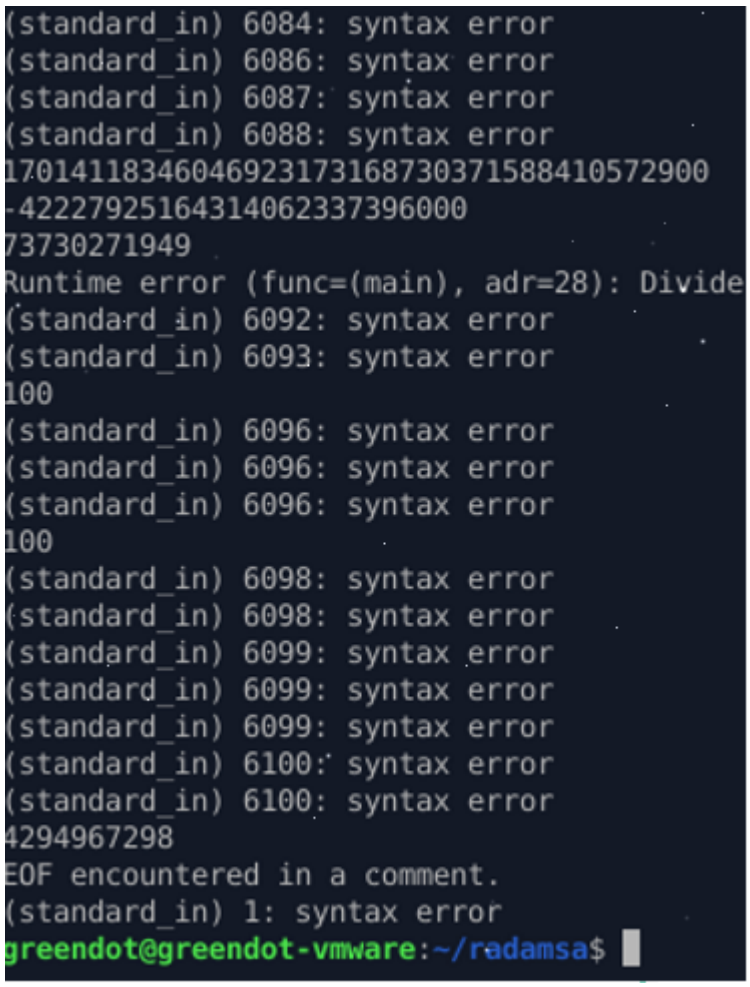
- 유닉스 유틸리티 bc(계산기)에 퍼징

  ```
  echo "100 * (1 + (2 / 3))" | radamsa -n 10000 | bc
  ```

  - 실행결과
    - 행걸린 경우


(standard_in) 55054: syntax error
(standard_in) 55055: syntax error
(standard_in) 55056: syntax error
(standard_in) 55057: syntax error
(standard_in) 55058: syntax error
(standard_in) 55059: syntax error
(standard_in) 55060: syntax error
(standard_in) 55061: syntax error
(standard_in) 55062: syntax error
(standard_in) 55063: syntax error
(standard_in) 55064: syntax error
(standard_in) 55065: syntax error
(standard_in) 55066: syntax error
[]^[[Dasdasdweiojgiqwghioqweg

- 비정상 종료


(standard_in) 6084: syntax error
(standard_in) 6086: syntax error
(standard_in) 6087: syntax error
(standard_in) 6088: syntax error
170141183460469231731687303715884105729000
-42227925164314062337396000
73730271949
Runtime error (func=(main), adr=28): Divide
(standard_in) 6092: syntax error
(standard_in) 6093: syntax error
100
(standard_in) 6096: syntax error
(standard_in) 6096: syntax error
(standard_in) 6096: syntax error
100
(standard_in) 6098: syntax error
(standard_in) 6098: syntax error
(standard_in) 6099: syntax error
(standard_in) 6099: syntax error
(standard_in) 6099: syntax error
(standard_in) 6100: syntax error
(standard_in) 6100: syntax error
4294967298
EOF encountered in a comment.
(standard_in) 1: syntax error
greendot@greendot-vmware:~/radamsa$