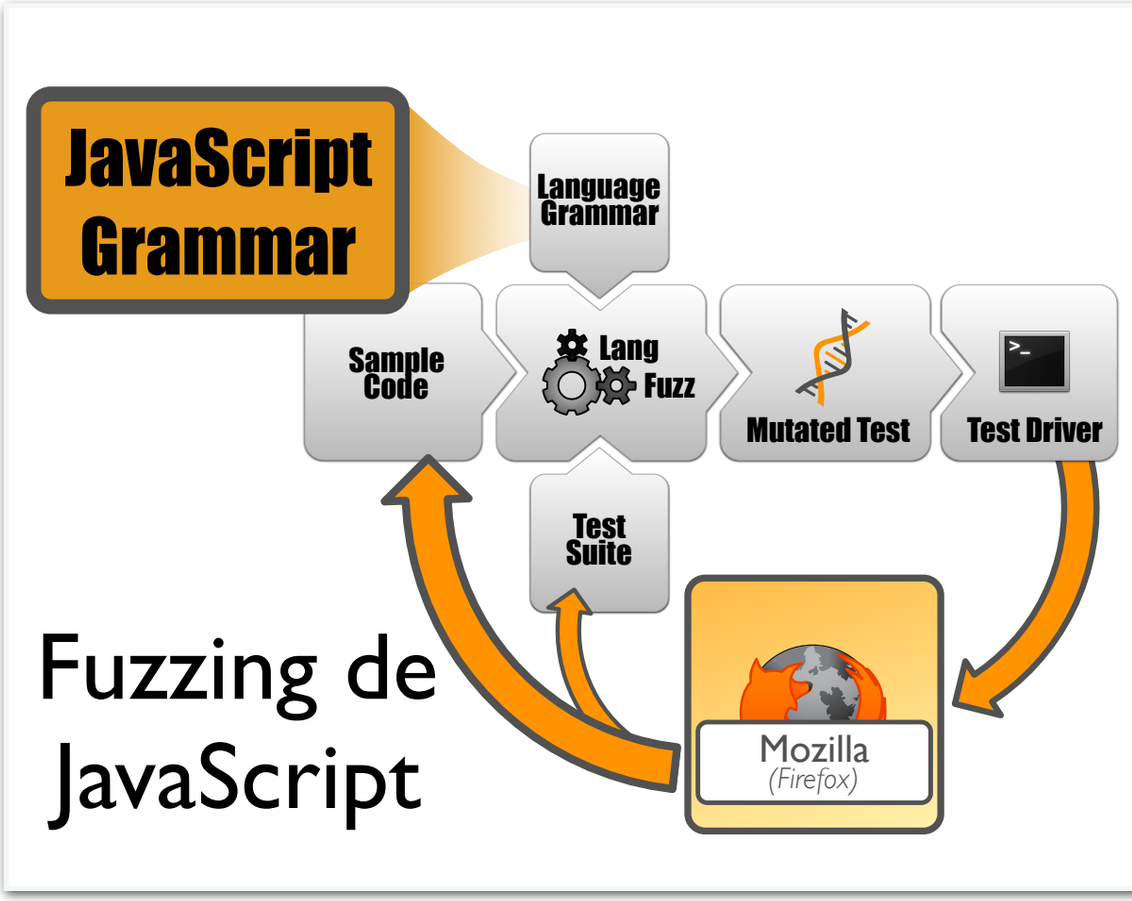
Taller #7: AFL

Generación Automática de Casos de Test - 2018

Fuzzing

“ab’d&gfdfggg” grep • sh • sed … 25%–33%

Fuzzing en sus orígenes = Random Testing a nivel de Sistema (System Level)



Fuzzing de JavaScript

DOI:10.1145/2093548.2093564

Article development led by acmqueue
queue.acm.org

SAGE has had a remarkable impact at Microsoft.

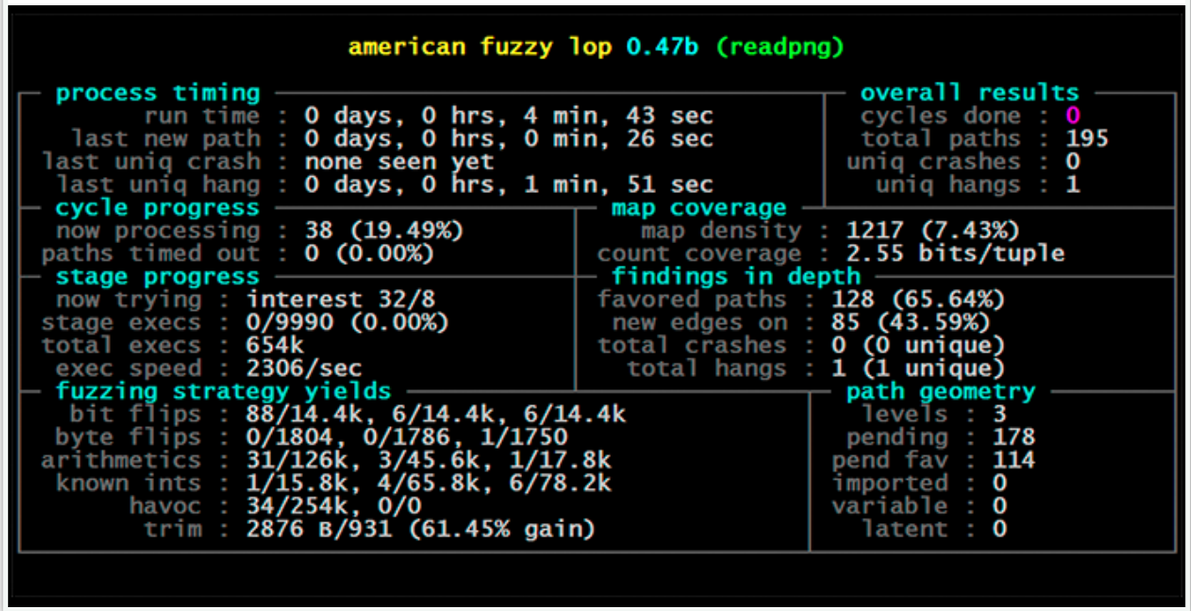
BY PATRICE GODEFROID, MICHAEL Y. LEVIN, AND DAVID MOLNAR

SAGE: Whitebox Fuzzing for Security Testing

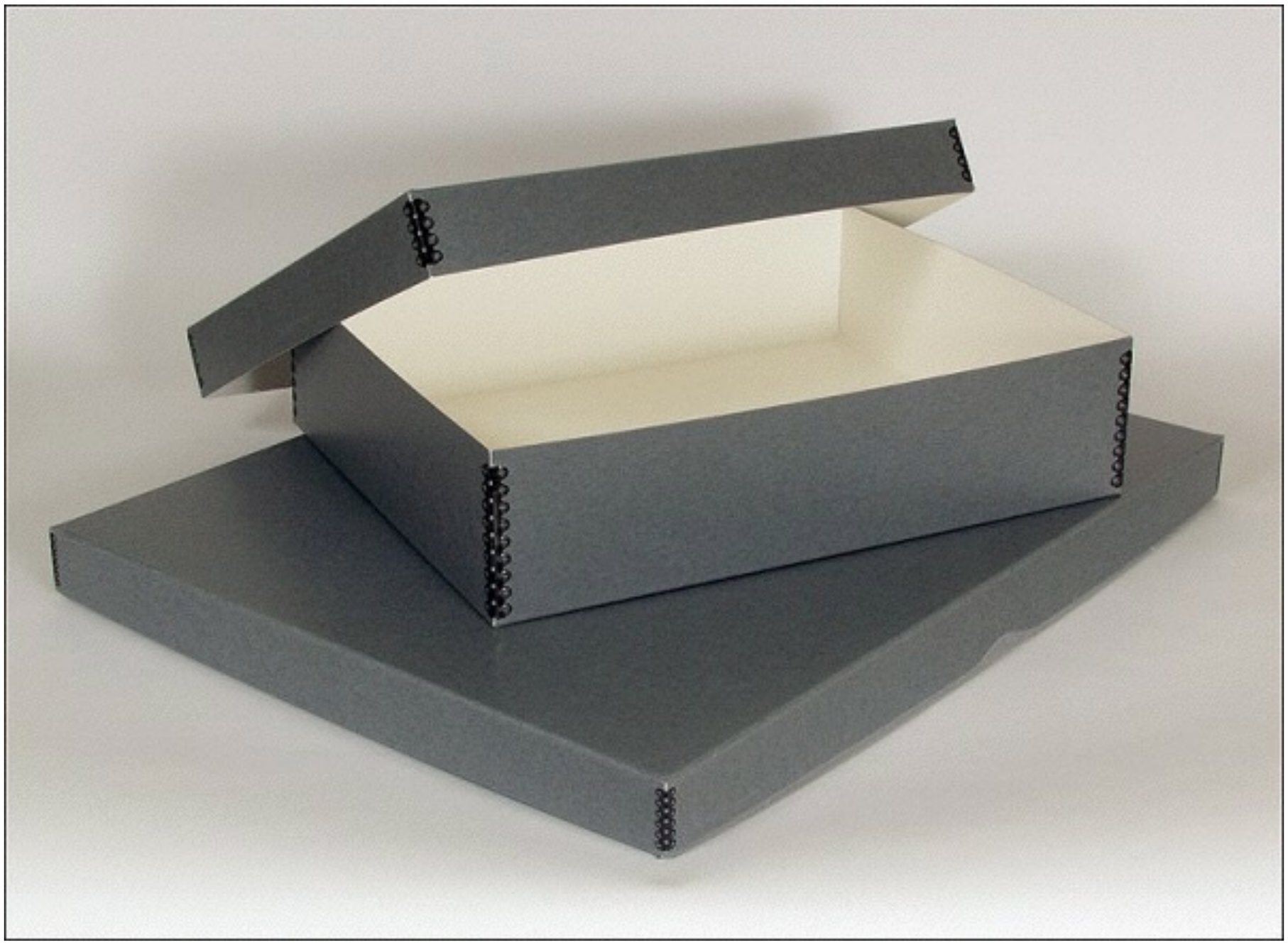
and its users millions of dollars. I monthly security update costs y $0.001 (one tenth of one cent) in j electricity or loss of productivity, th this number multiplied by one l lion people is $1 million. Of course malware were spreading on your n chine, possibly leaking some of yc private data, then that might cost

your operating system) has read t image data, decoded it, created n data structures with the decoded da and passed those to the graphics ca in your computer. If the code imp menting that jpg parser contains bug such as a buffer overflow that c be triggered by a corrupted jpg ima then the execution of this jpg par on your computer could potentially

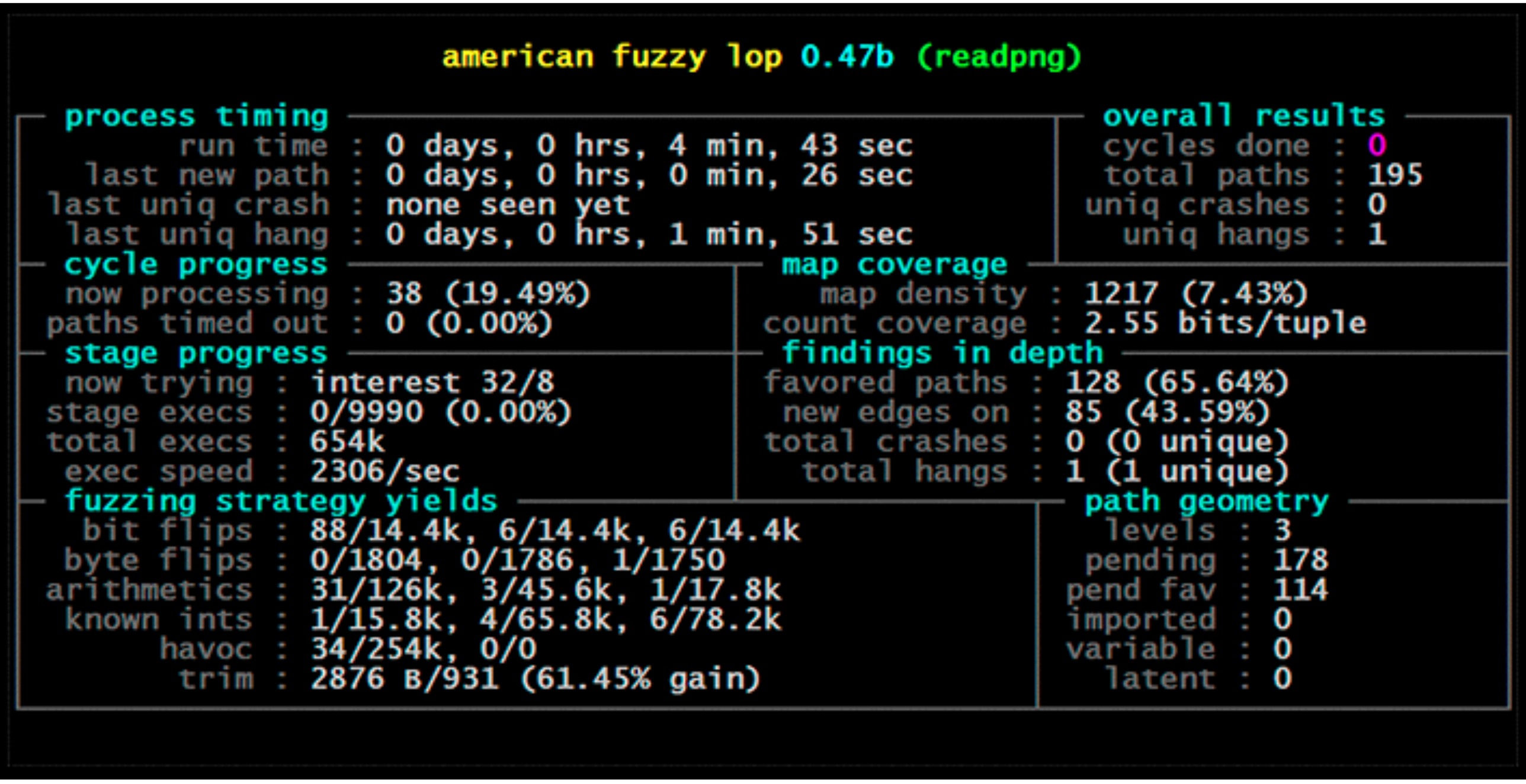
American Fuzzy Lop



Repaso: Grey-box Fuzzing



Repaso: American Fuzzy Lop



Repaso: The AFL approach

1) Encolar **inputs iniciales** provistos por el usuario a la cola Q,

2) Tomar un input de la cola Q

3) Intentar **reducir** el input a su menor tamaño si alterar su capacidad de cobertura,

4) Aplicar un conjunto de mutaciones (cambios) al input utilizando una variedad de **estrategias tradicionales de fuzzing**,

5) Si alguno de los nuevos inputs resulta en un aumento de cobertura, agregar el input a la cola Q.

6) Volver a 2).

AFL-Instalación (1)

```
$ sudo apt-get install
```

AFL-Instalación (2)

- Descargar Latest source tarball de AFL:
  - lcamtuf.coredump.cx/afl/releases/afl-latest.tgz

```
$ cd afl-2.52b

$ make

$ export AFL_PATH=/path/to/afl-2.52b

$ export PATH=$PATH:$AFL_PATH
```

Configurar Variables

$ export AFL_PATH=/path/to/afl-2.35b

$ export PATH=$PATH:$AFL_PATH

$ export AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=“”

$ export AFL_SKIP_CPUFREQ=“”

AFL-Instrumentación

```
$ afl-clang sample1.c -o sample1
```

AFL-Fuzzing

```
$ afl-fuzz -i afl_in -o afl_out ./sample1 @@
```

AFL-Minimización

```
$ afl-tmin -i INPUT -o OUTPUT ./sample1 @@
```

donde **INPUT** es el input a minimizar y **OUTPUT** es el nombre del archivo ya minimizado

Ejercicio #1

A. Instrumentar sample1.c y ejecutarlo con afl_in/input1.txt ¿Produjo alguna falla?

B. Fuzzear sample1 hasta que encuentre al menos 1 falla usando de semilla “afl_in” ¿Qué cantidad de inputs generados provocan crashes? ¿Cuántos provocan hangs? ¿Cuál es el tamaño mínimo de los inputs que provocan crashes?

C. Minimizar los inputs generados que provocan crashes únicamente. ¿Cuál es el tamaño mínimo que tienen?

D. Entregar todos los inputs generados que producen fallas (ya sean hangs o crahses) y la minimización si la hubiere.

Ejercicio #2

A. Instrumentar sample2.c y ejecutarlo con afl_in/input1.txt ¿Produjo alguna falla?

B. Fuzzear sample1 hasta que encuentre al menos 1 falla usando de semilla “afl_in” ¿Qué cantidad de inputs generados provocan crashes? ¿Cuántos provocan hangs? ¿Cuál es el tamaño mínimo de los inputs que provocan crashes?

C. Minimizar los inputs generados que provocan crashes únicamente. ¿Cuál es el tamaño mínimo que tienen?

D. Entregar todos los inputs generados que producen fallas (ya sean hangs o crahses) y la minimización si la hubiere.