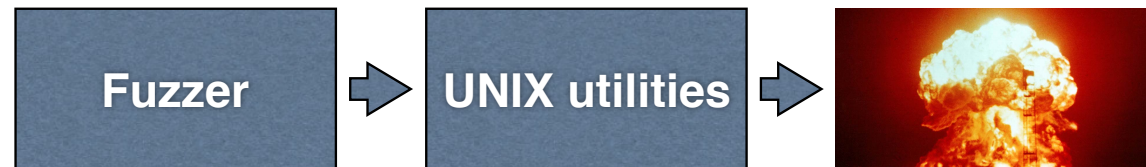


Taller #7: AFL

Generación Automática de Casos de Test - 2018

Fuzzing

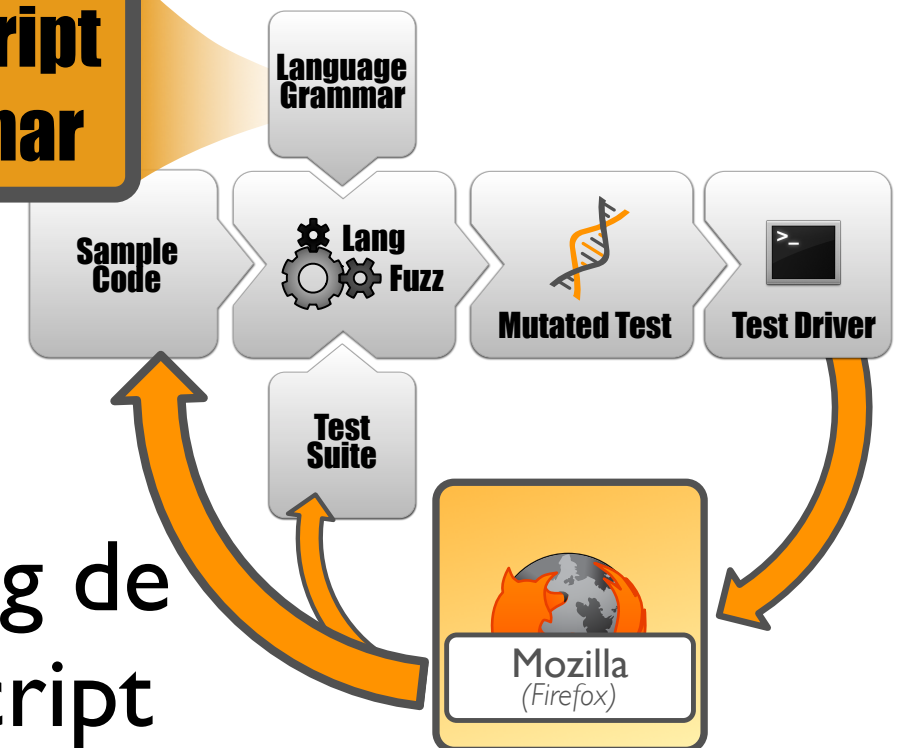


“ab’d&gfdfggg” grep • sh • sed ... 25%–33%

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

JavaScript Grammar

Fuzzing de JavaScript



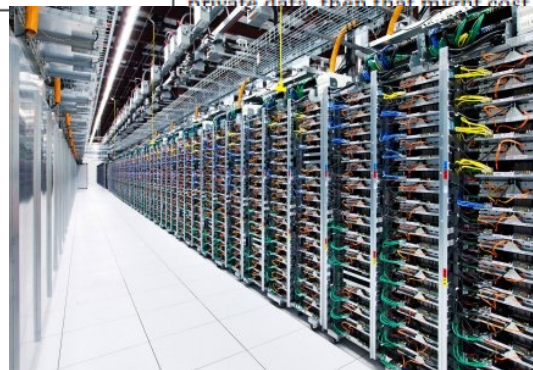
DOI:10.1145/2093548.2093564

Article development led by 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 y private data, then that might cost

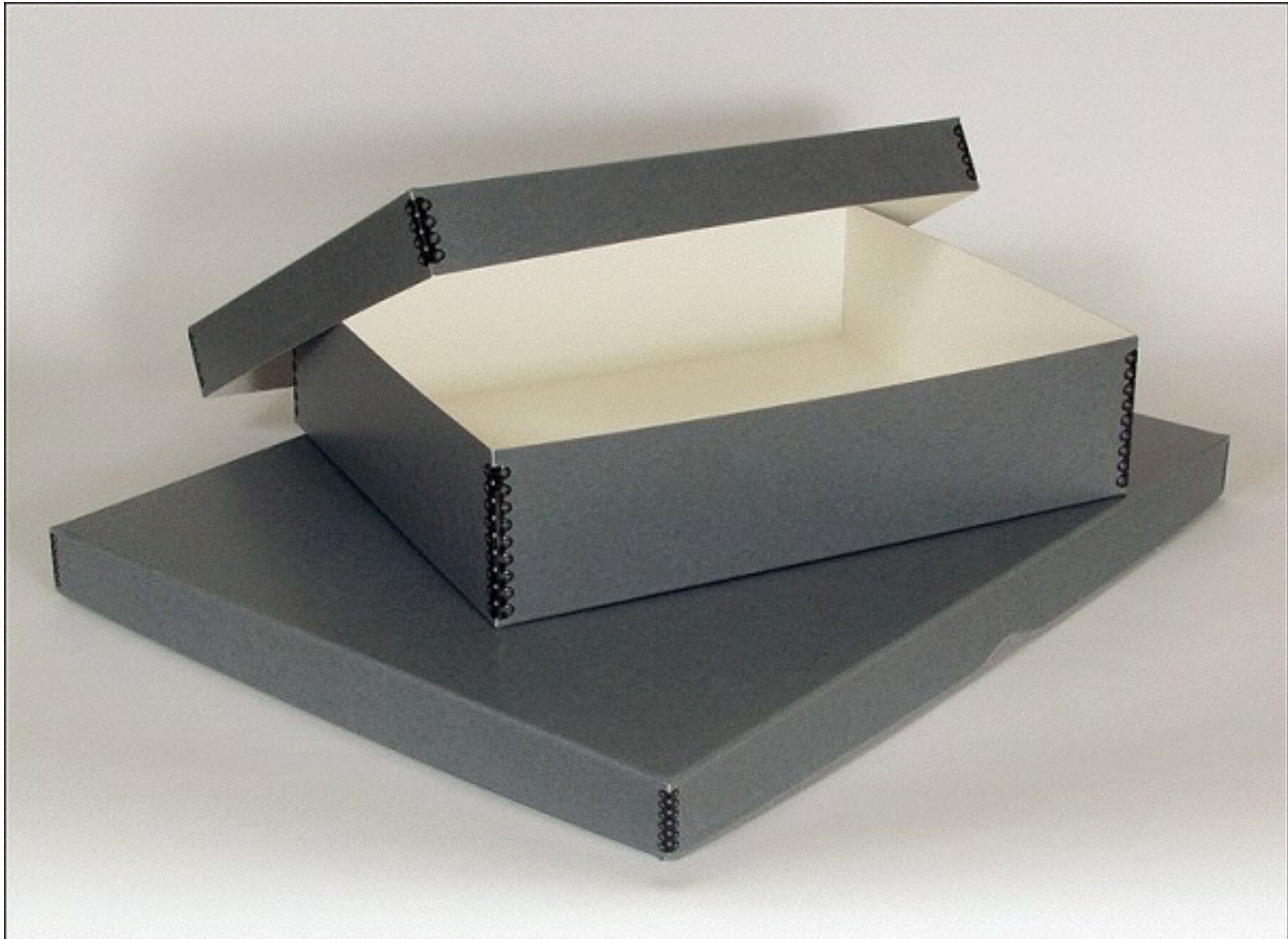
your operating system) has read t image data, decoded it, created n data structures with the decoded da and passed those to the graphics ce in your computer. If the code imp menting that jpg parser contains bug such as a buffer overflow that c be triggered by a corrupted jpg ima then the execution of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

American Fuzzy Lop

american fuzzy lop 0.47b (readpng)

process timing		overall results	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
cycle progress		map coverage	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
stage progress		findings in depth	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
fuzzing strategy yields		path geometry	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 8/931 (61.45% gain)	latent	: 0

Repaso: Grey-box Fuzzing



Repaso: American Fuzzy Lop

american fuzzy lop 0.47b (readpng)

process timing

run time : 0 days, 0 hrs, 4 min, 43 sec
last new path : 0 days, 0 hrs, 0 min, 26 sec
last uniq crash : none seen yet
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec

cycle progress

now processing : 38 (19.49%)
paths timed out : 0 (0.00%)

stage progress

now trying : interest 32/8
stage execs : 0/9990 (0.00%)
total execs : 654k
exec speed : 2306/sec

fuzzing strategy yields

bit flips : 88/14.4k, 6/14.4k, 6/14.4k
byte flips : 0/1804, 0/1786, 1/1750
arithmetics : 31/126k, 3/45.6k, 1/17.8k
known ints : 1/15.8k, 4/65.8k, 6/78.2k
havoc : 34/254k, 0/0
trim : 2876 B/931 (61.45% gain)

overall results

cycles done : 0
total paths : 195
uniq crashes : 0
uniq hangs : 1

map coverage

map density : 1217 (7.43%)
count coverage : 2.55 bits/tuple

findings in depth

favorable paths : 128 (65.64%)
new edges on : 85 (43.59%)
total crashes : 0 (0 unique)
total hangs : 1 (1 unique)

path geometry

levels : 3
pending : 178
pend fav : 114
imported : 0
variable : 0
latent : 0

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 crashes) 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 crashes) y la minimización si la hubiere.