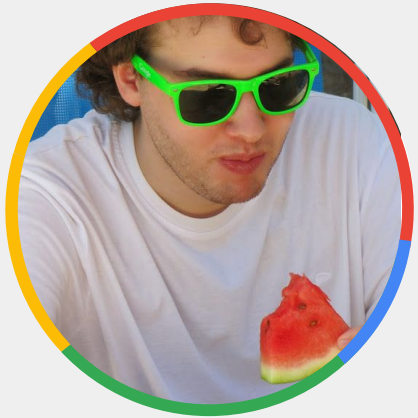
Create
Design
Code
Build
for
everyone

~~Lightning~~ Talks
with Google

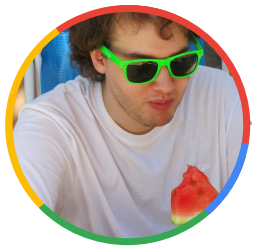
Fuzzing: el arte de romper software



Meet Marco Vanotti

Mountain View

Path to Google



Marco Vanotti

Open Source Dev
Argentum Online

Teaching Assistant
University of Buenos Aires

Developer
Monits

Intern
Google Accelerator

Intern
Microsoft Azure Security

Intern
Google Accelerator

SWE
Google NBU

SWE
Fuchsia Security

Fuzzing

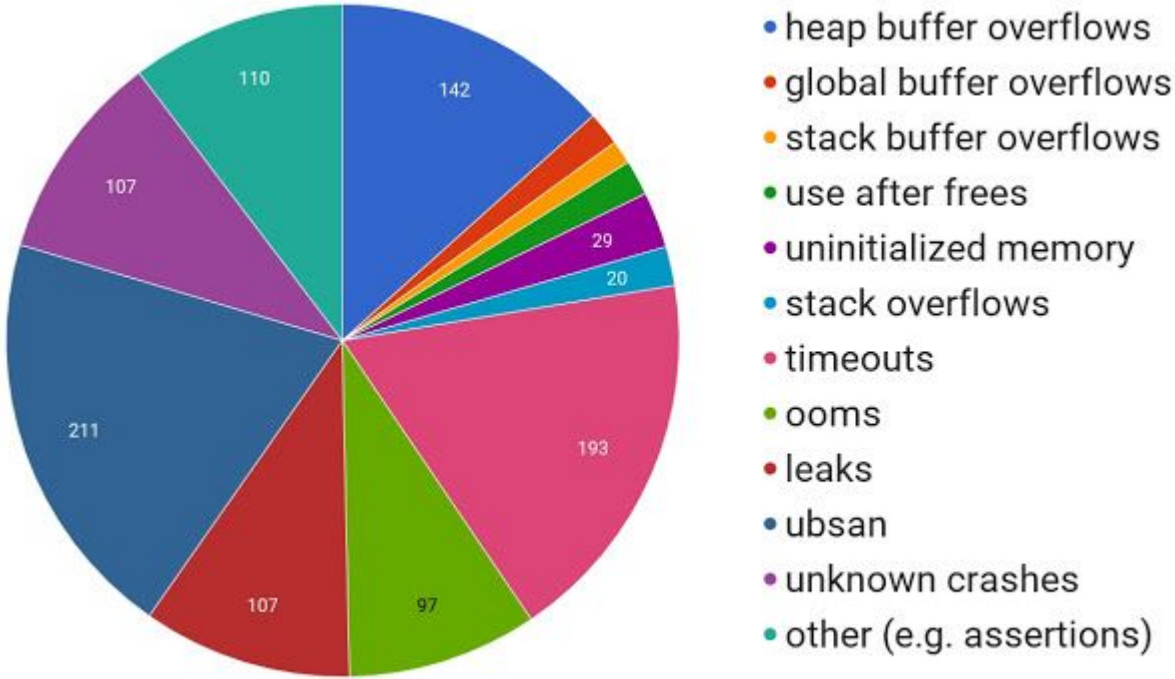
El arte de romper software

Qué, cómo y por qué

Fuzzing es un proceso para testear APIs usando datos autogenerados.

- Generamos un input
- Corremos el programa con ese input
- Miramos si explota

Útil para hallar potenciales bugs de seguridad...



Desglose de tipos de bugs encontrados por OSS-Fuzz en 47 proyectos open source el 2017 luego de 5 meses de fuzzing.

Algunos ejemplos...

- RADAMSA: Input Generation

```
$ echo "Fuzztron 2000" | radamsa --seed 4
Fuzztron 4294967296
```

```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
18446744073709551615 + 4)))
1 + (2 + (3 + 170141183460469231731687303715884105727))
```

Algunos ejemplos...

- Scapy: Network Fuzzing

```
>>> fuzz(Dot11()).show()
###[ 802.11 ]###
  subtype= <RandNum>
  type= <RandNum>
  proto= <RandNum>
  FCfield= retry+wep+order
  ID= <RandShort>
  addr1= 13:d2:e0:ee:d9:77
  addr2= fb:98:ab:83:8f:51
  addr3= 00:00:00:00:00:00
  SC= <RandShort>
```

```
>>> ejemplo = fuzz(IP())
>>> ejemplo.show()
###[ IP ]###
  version= <RandNum>
  ihl= None
  tos= 63
  len= None
  id= <RandShort>
  flags= MF+evil
  frag= <RandNum>
  ttl= <RandByte>
  proto= <RandByte>
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
```

Historia: Los comienzos

- Tarjetas Perforadas ~**1950**
- “The Monkey” por Steve Capps en **1981**
- Universidad de Wisconsin, tp de sistemas operativos **1988**
  “An empirical study of the reliability of unix utilities.”

Mejoras: Detectar Anomalías

- Assert()

- Valgrind
  - Para el binario antes de que arranque y simula todo el código.
  - Detecta accesos inválidos a memoria, memory leaks, etc.

- Libdislocate
  - Reemplaza libc allocators, detecta corrupciones de memoria.

- Sanitizers (ASAN, UBSAN, MSAN, TSAN, HWASAN)
  - Compila el binario con instrumentación especial para detectar accesos inválidos a memoria, problemas de concurrencia, comportamiento indefinido, etc

AddressSanitizer (ASAN)

Creado por Kostya Serebryani et al.

github.com/google/sanitizers

```
==25739==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000b494
WRITE of size 4 at 0x60200000b494 thread T0
    #0 0x527823 in Knapsack::maxBenefitDP() /home/martin/Desktop/repos/algo3-201
    #1 0x5286d7 in Knapsack::maxBenefit() /home/martin/Desktop/repos/algo3-2019-
    #2 0x529037 in LLVMFuzzerTestOneInput /home/martin/Desktop/repos/algo3-2019-
    #3 0x42d88a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigne
    #4 0x42d085 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, b
    #5 0x42f039 in fuzzer::Fuzzer::MutateAndTestOne() (/home/martin/Desktop/repo
    #6 0x42fd05 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<c
> > const&) (/home/martin/Desktop/repos/algo3-2019-2c-tp1/src/fuzz+0x42fd05)
    #7 0x425900 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char con
    #8 0x447c72 in main (/home/martin/Desktop/repos/algo3-2019-2c-tp1/src/fuzz+0
    #9 0x7f9b20a3ab6a in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x26
    #10 0x41eaa9 in _start (/home/martin/Desktop/repos/algo3-2019-2c-tp1/src/fuz

0x60200000b494 is located 0 bytes to the right of 4-byte region [0x60200000b490,
allocated by thread T0 here:
    #0 0x522a82 in operator new(unsigned long) (/home/martin/Desktop/repos/algo3
```

```
void foo(T *a) {
  *a = 0x1234;
}
```

8-byte store:

```
clang -O2 -faddress-sanitizer a.c -c -DT=long
```

```
push   %rax
mov    %rdi,%rax
shr    $0x3,%rax
mov    $0x100000000000,%rcx
or     %rax,%rcx
cmpb   $0x0,(%rcx) # Compare Shadow with 0
jne    23 <foo+0x23>  # To Error
movq   $0x1234,(%rdi) # Original store
pop    %rax
retq
callq  __asan_report_store8 # Error
```

Mejoras: Obtener Cobertura

- SanitizeCoverage (SANCOV)
  - Otro sanitizer, instrumenta el binario agregando una llamada a una función `__sanitizer_cov_trace_pc` al final de cada bloque básico / eje.

- DynamoRIO (drcov)
  - Copia de a un bloque básico, lo modifica de forma dinámica y ejecuta nativamente.

- Intel Processor Trace (IPT)
  - El procesador «monitorea» un rango de memoria y escribe en memoria la lista de las ramas tomadas.

SanitizerCoverage

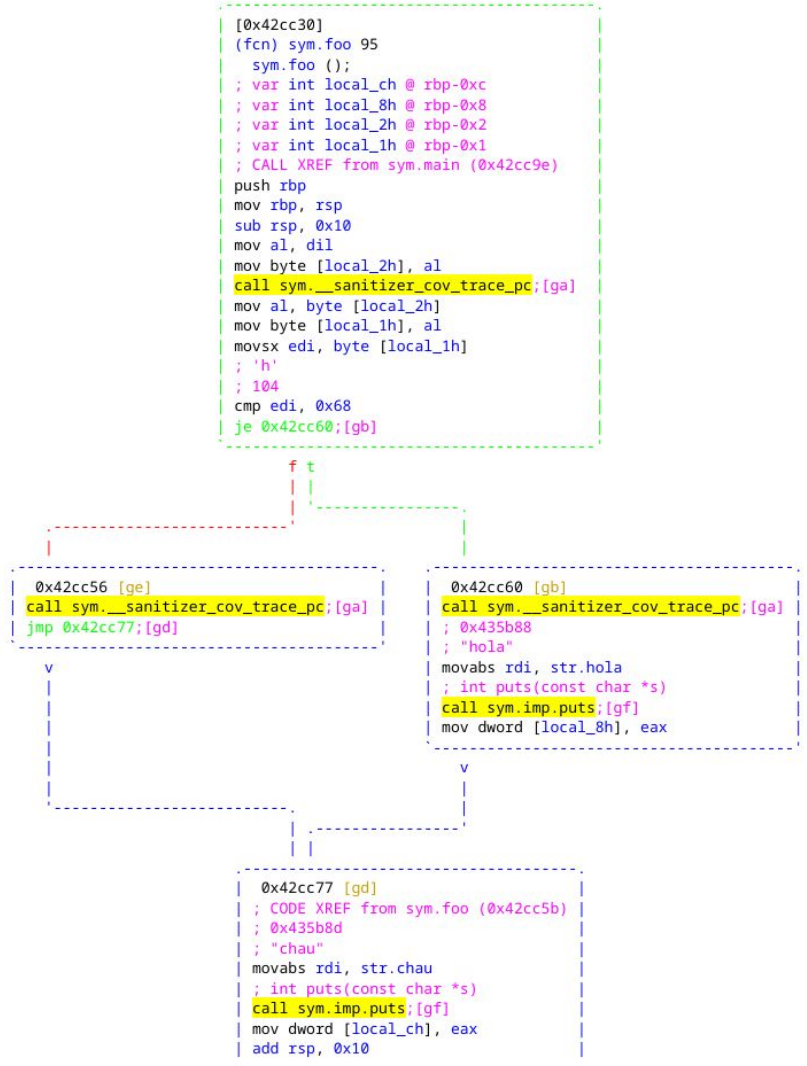
```c
#include <stdio.h>

void foo(char c) {
  if (c == 'h') {
    puts("hola");
  }
  puts("chau");
}

int main(void) {
  foo('c');
}

void __sanitizer_cov_trace_pc(void) {}
```

```
clang-8 -fsanitize-coverage=trace-pc -O0 ejemplo.c -o ejemplo
```



Manejo de Corpus

- Agregar inputs interesantes al corpus
  - Nuevos caminos recorridos
  - Nuevos crashes

- Minimizar Corpus
  - Usar cobertura para tratar de achicar el corpus
  - Mergear Corpus de distintas corridas

Mutaciones

- Generalmente son heurísticas
- Invertir Bits/Bytes
- Operaciones Aritméticas / Known Ints
- Copiar, Mover, Insertar, Borrar bloques de bits/bytes
- Tomar valores de diccionarios
- Combinar corpus

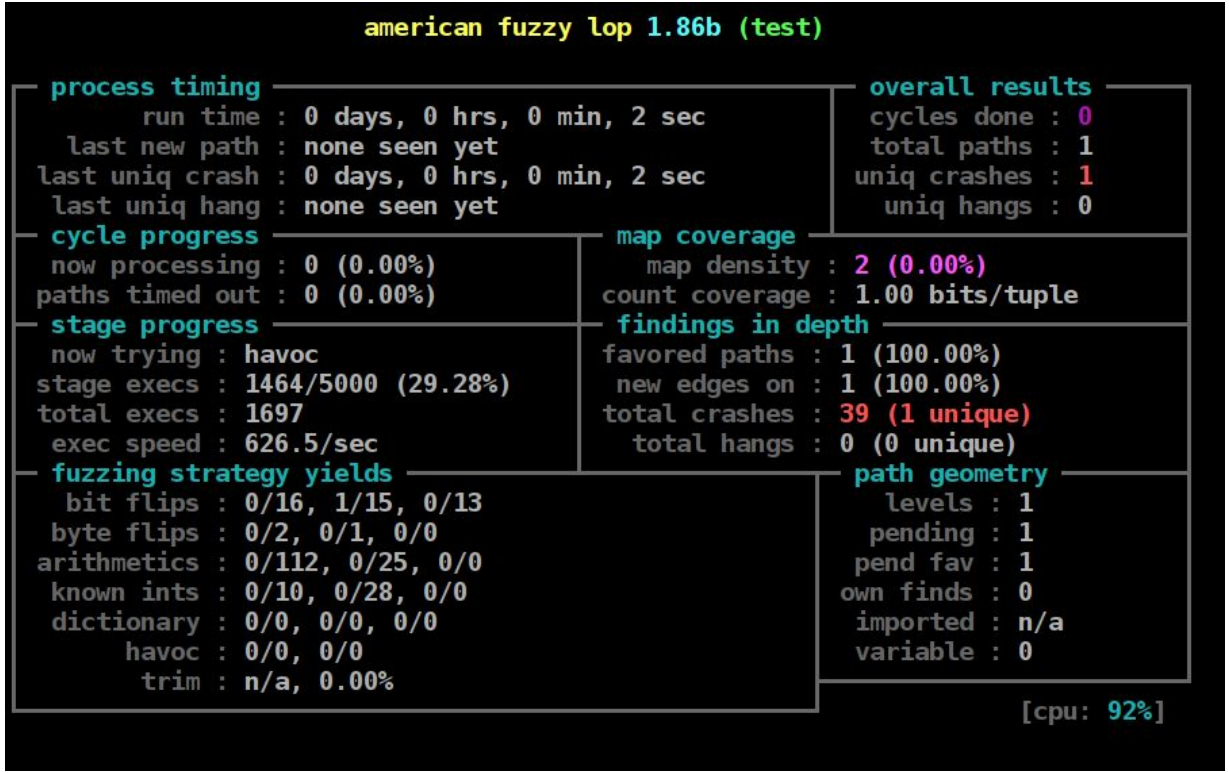
American Fuzzy Lop (AFL)

Creado por Michał Zalewski (lcamtuf@)

github.com/google/AFL

- Muy fácil de usar y efectivo
- Guiado por cobertura
  - Con soporte para black-box fuzzing
- Soporte para ForkServer e in-process fuzzing
- Soporte para diccionarios, manejo de corpus, paralelismo

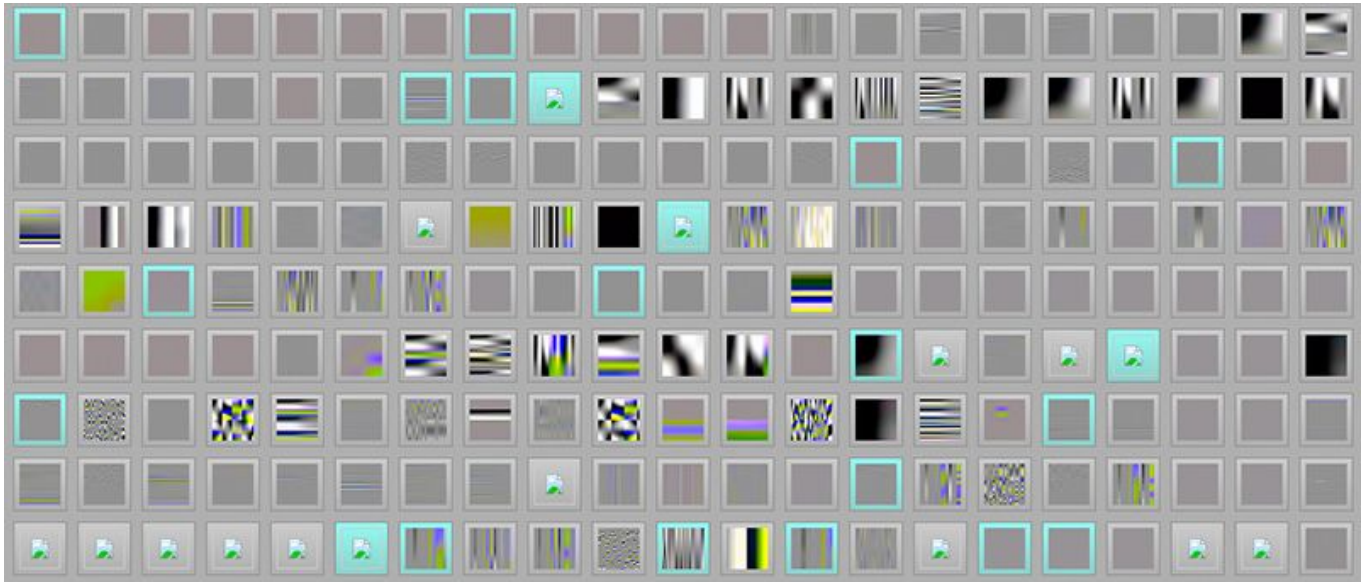
American Fuzzy Lop (AFL)



American Fuzzy Lop (AFL)

Sacando imágenes de la galera

https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html



American Fuzzy Lop (AFL)

Cobertura

- Compila programas con su propia versión de GCC/Clang
- Mantiene una lista de ejes usando una heurística:

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

- Para black-box fuzzing, usa QEMU instrumentado para obtener cobertura.

American Fuzzy Lop (AFL)

Extra Features

- Explorador de Crashes *(peruvian were-rabbit mode)*
  - Dado un Crash, ejecuta el fuzzer varias veces y se queda con inputs que sigan crasheando pero que ejecuten otros caminos.
  - Útil para entender cómo explotar bugs.

- Analisis de Corpus

LibFuzzer

Creado por Kostya Serebryani et al.
llvm.org/docs/LibFuzzer.html

- In-Process Fuzzer
- Guiado por Cobertura (sancov)
- Permite fuzzear cualquier función
- Hace uso de otros Sanitizers (ASAN, MSAN, UBSAN, etc)
- Soporta diccionarios, minimización de corpus, paralelismo.
- Facil de usar, pero requiere programación...

LibFuzzer

¿Requiere Programación?

Solo hay que crear una función que transforme un array de bytes en un input válido para tu API.

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
  static_assert(kMaxBufSize <= INT_MAX);
  if (size > INT_MAX) {
    return 0;
  }
  LZ4_decompress_safe(reinterpret_cast<const char*>(data), dstBuffer, static_cast<int>(size),
                      static_cast<int>(kMaxBufSize));
  return 0;
}
```

LibFuzzer

```
INFO: Seed: 3918206239
INFO: Loaded 1 modules (14 guards): [0x73be00, 0x73be38),
INFO: Loaded 1 PC tables (7 PCs): 7 [0x52f8c8,0x52f938),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#0      READ units: 1
#1      INITED cov: 3 ft: 3 corp: 1/1b exec/s: 0 rss: 26Mb
#8      NEW    cov: 4 ft: 4 corp: 2/29b exec/s: 0 rss: 26Mb L: 28 MS: 2 InsertByte-InsertRepeatedBytes-
#3405   NEW    cov: 5 ft: 5 corp: 3/82b exec/s: 0 rss: 27Mb L: 53 MS: 4 InsertByte-EraseBytes-...
#8664   NEW    cov: 6 ft: 6 corp: 4/141b exec/s: 0 rss: 27Mb L: 59 MS: 3 CrossOver-EraseBytes-...
#272167 NEW    cov: 7 ft: 7 corp: 5/201b exec/s: 0 rss: 51Mb L: 60 MS: 1 InsertByte-
==================================================================
==2335==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000155c13 at pc 0x0000004ee637...
READ of size 1 at 0x602000155c13 thread T0
    #0 0x4ee636 in FuzzMe(unsigned char const*, unsigned long) FTS/tutorial/fuzz_me.cc:10:7
    #1 0x4ee6aa in LLVMFuzzerTestOneInput FTS/tutorial/fuzz_me.cc:14:3
...
artifact_prefix='./'; Test unit written to ./crash-0eb8e4ed029b774d80f2b66408203801cb982a60
...
```

Syzkaller

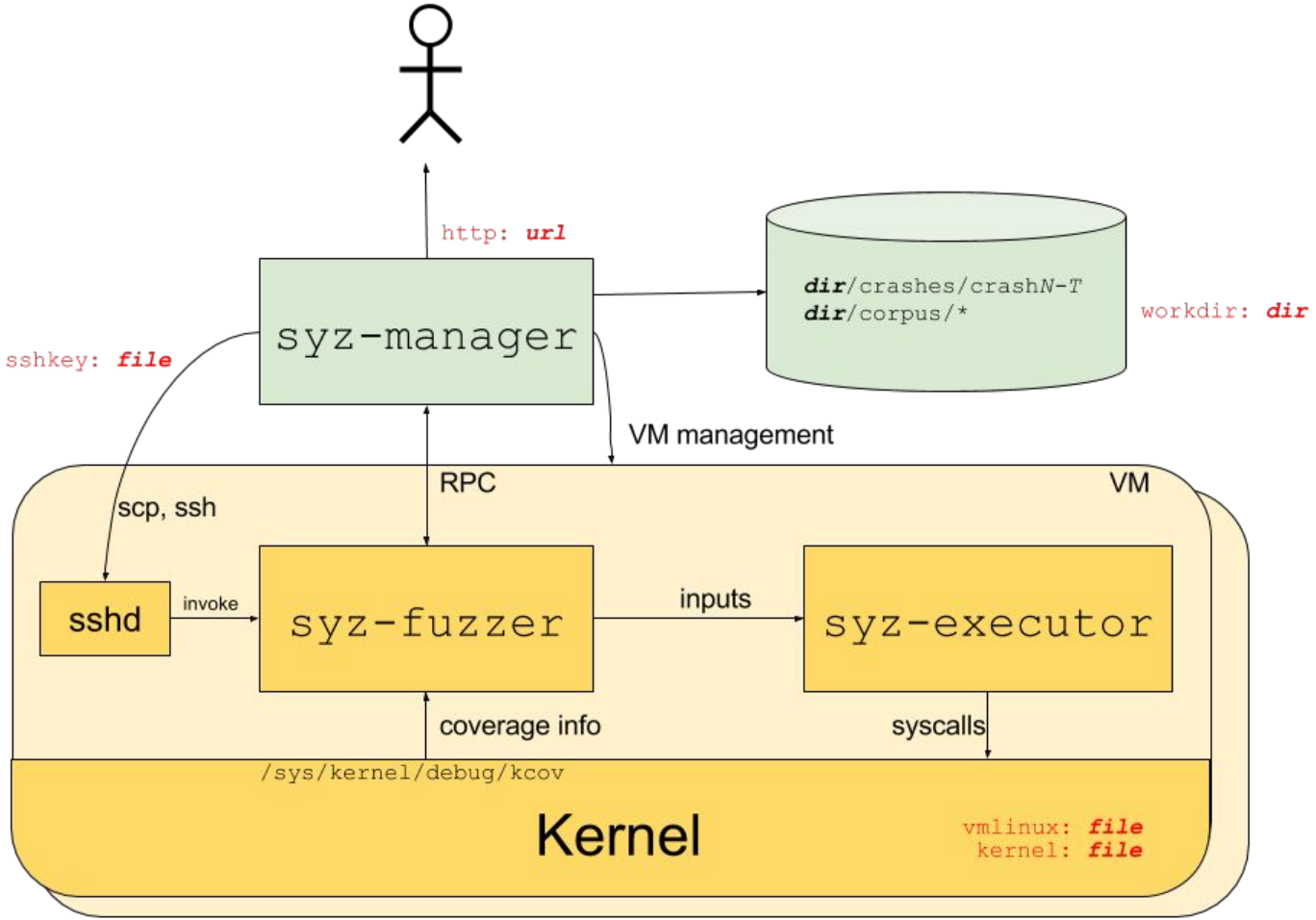
Creado por Dmitry Vyukov

github.com/google/syzkaller

- Kernel Fuzzer
- Generacional, Mutacional
- Guiado por Cobertura (KCOV)
- Crea “reproducers” para bugs hallados
- Hace uso de Kernel Sanitizers (KASAN, KMSAN, KTSAN)
- Bug Bisection

Syzkaller

Estructura



Syzkaller

Formato de Syscalls

```
resource zx_vmo[zx_handle]

zx_vmo_create(size int64, options flags[vmo_create_options], out ptr[out, zx_vmo])
zx_vmo_op_range$ZX_VMO_OP_COMMIT(handle zx_vmo, op const[ZX_VMO_OP_COMMIT], offset i
zx_vmo_op_range$ZX_VMO_OP_DECOMMIT(handle zx_vmo, op const[ZX_VMO_OP_DECOMMIT], offs
zx_vmo_op_range$ZX_VMO_OP_CACHE_SYNC(handle zx_vmo, op const[ZX_VMO_OP_CACHE_SYNC],
```

Formato de Programas

```
mmap(&(0x7f0000000000), (0x1000), 0x3, 0x32, -1, 0)
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
```

Otros Fuzzers de Kernels

- kAFL (RUB-SysSec)
  - Obtiene cobertura usando IPT
    - Versión modificada de Linux y QEMU para agregar soporte.
    - Corre varias veces los mismos casos para detectar cobertura no relevante.
    - NO es un fork de AFL.
- TriforceAFL (nccgroup)
  - Forkea AFL y sus modificaciones a QEMU para soportar full-system virtualization.

Problemas Comunes

- Checksums - Números Mágicos
  - Se puede hacer post-procesamiento del input.
  - Instrumentar Comparaciones
  - Lo mejor es modificar el código bajo test.

- Codepaths no alcanzados
  - Revisar visualmente la cobertura y fijarse dónde no anda.
  - Mejorar el corpus del fuzzer con datos reales.

- Protocolos que requieren mantener un estado
  - Muchas veces es necesario un fuzzer ad-hoc.

- Es difícil encontrar bugs en la lógica de negocio.

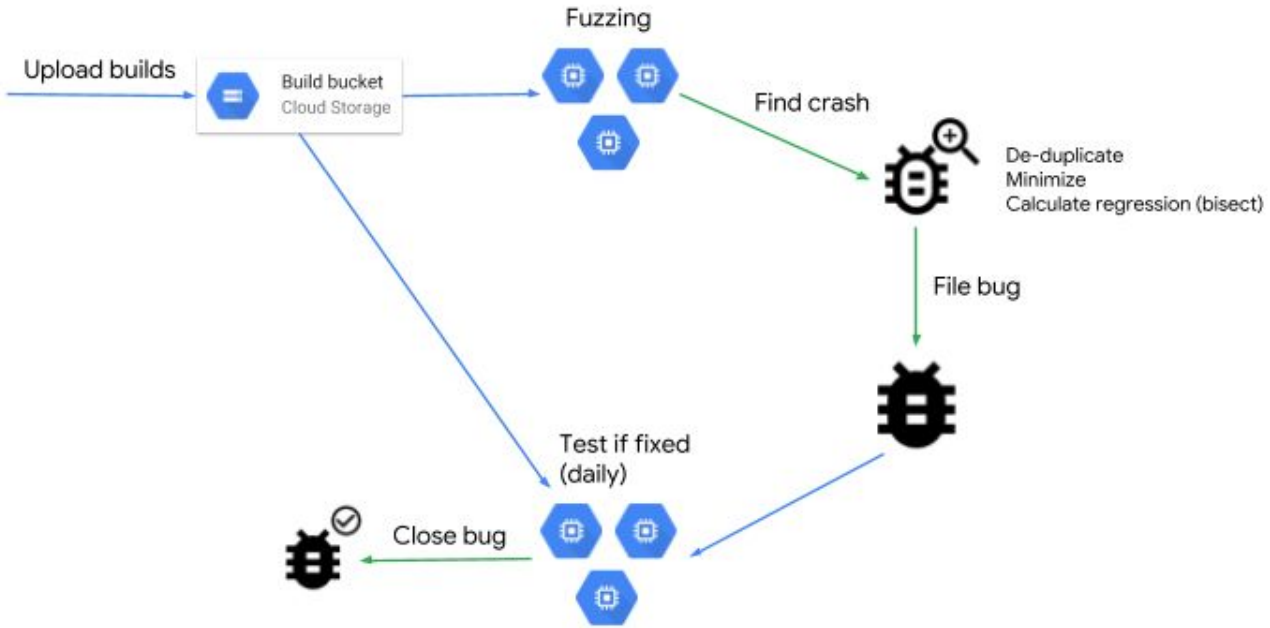
Fuzzing Targets

- Todo lo que maneje input de usuarios/no confiable
- Usar unit tests para crear fuzzers (libFuzzer)
- Parsers de cualquier tipo (json, archivos, etc)

Fuzzeando a gran escala

ClusterFuzz
github.com/google/clusterfuzz

Open Source
Orig. usado en Chrome
25000 vms internas
AFL & libFuzzer
Estadisticas,
Visualizaciones,
Bisección.



Recompensas por escribir Fuzzers

Si escribís un fuzzer para Chrome, Google lo deja corriendo en ClusterFuzz continuamente contra nuevas versiones del código.

Los bugs que encuentre tu fuzzer serán recompensados por Google, con premios de hasta $1000 dólares.

Más info: google.com/about/appsecurity/chrome-rewards/#fuzzerprogram

Conclusiones

- Usar memory-safe languages.
- Fuzzing está bueno y es fácil.
- Complementar otros tipos de testing con fuzzing.
- Testear usando AddressSanitizer.
- Hay mucho para mejorar.

Thank you!
Questions?