

Greybox-Fuzzing

Ingeniería del Software 2

Fuzzing

- Idea: Testing a nivel de sistema (i.e. no controlamos el input)
- Buscamos aserciones violentadas, buffers overflows (vulnerabilidad), memory leaks (robustez)
- Casi siempre partimos de una semilla (i.e. conjunto inicial de inputs que serán “fuzzeados”)

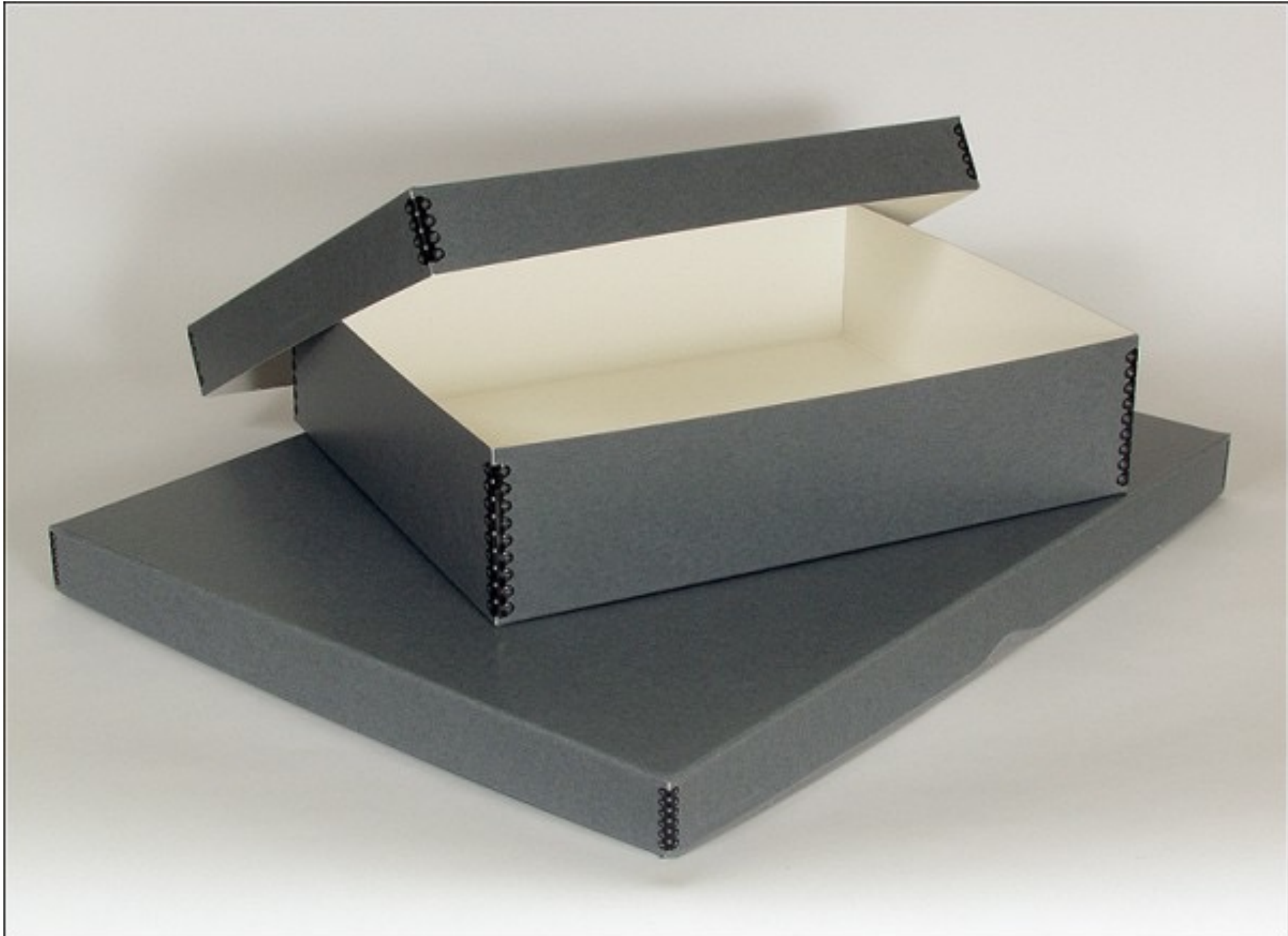
Blackbox fuzzing

- Partimos de un seed (conjunto de inputs)
- Se elige cada seed y se le aplican una cantidad aleatoria de mutaciones entre 0 y K.
- Algunas mutaciones posibles:
 - Insertar un caracter
 - Eliminar un caracter
 - “Flippear” un caracter

Blackbox Fuzzing

```
Def BlackBoxFuzz(SEED):  
    numberOfExecutedTests := 0  
    While Budget is not empty:  
        If numberOfExecutedTests < len(SEED)  
            Test := SEED[numberOfExecutedTests]  
        Else:  
            Choose randomly input from SEED  
            Test := mutate(input, K)  
        Endif  
        Run Test (Report if crashes/hangs/assertions)  
        numberOfExecutedTests += 1  
    EndWhile
```

Greybox Fuzzing



Greybox Fuzzing

- Idea: Si un input aportó cobertura, lo agrego al corpus de inputs para futuras mutaciones
- Para eso necesitamos medir la cobertura (ej: branches cubiertos, basic blocks cubiertos) del programa bajo test
- Energía: cada elemento del seed posee una “energía” (i.e. probabilidad de elegirlo)

Greybox Fuzzing

```
Def GreyBoxFuzz(SEED):  
    numberOfExecutedTests := 0  
    init_len := len(SEED)  
    While Budget is not empty:  
        If numberOfExecutedTests < init_len  
            Input := SEED[numberOfExecutedTests]  
            Execute Input (Report if crashes/hangs/etc)  
        Else:  
            Choose randomly input from SEED  
            NewInput := mutate(input, K)  
            Execute NewInput (Report if crashes/hangs/etc)  
            If NewInput adds new Coverage:  
                SEED += NewInput  
            Endif  
        Endif  
        numberOfExecutedTests+=1  
    EndWhile
```

Boosted Greybox fuzzing

- Idea: Aumentar la probabilidad de elegir un input de la semilla de acuerdo a las chances de descubrir otros caminos en el CFG
- La energía de un input s se define como $e(s)$ donde $p(s)$ es el camino que recorrió la ejecución de s y $f(p(s))$ es la frecuencia de apariciones de un camino en el test suite

$$e(s) = \frac{1}{f(p(s))^a}$$

Boosted Greybox Fuzzing

```
Def BoostedGreyBoxFuzz(SEED):  
    energy:= {}  
    init_len = len(SEED)  
    numberOfExecutedTests := 0  
    While Budget is not empty:  
        If numberOfExecutedTests < init_len  
            Input := SEED[numberOfExecutedTests]  
            Execute Input (Report if crashes/hangs/etc)  
        Else:  
            Choose input from SEED using energy(...)  
            newInput := mutate(input, K)  
            Execute NewInput (Report if crashes/hangs/etc)  
            If NewInput adds new Coverage:  
                SEED += NewInput  
            Endif  
        Endif  
        numberOfExecutedTests += 1  
        Update energy(s) for each s in SEED  
    Endwhile
```

Demo

- En <https://www.fuzzingbook.org/html/GreyboxFuzzer.html> encontrará el código de tres tipos distintos de fuzzers:
 - Blackbox fuzzer (MutationFuzzer)
 - Greybox fuzzer
 - Boosted greybox fuzzer

Demo

Ejecutar los 3 fuzzers sobre el HTMLParser de la librería de Python

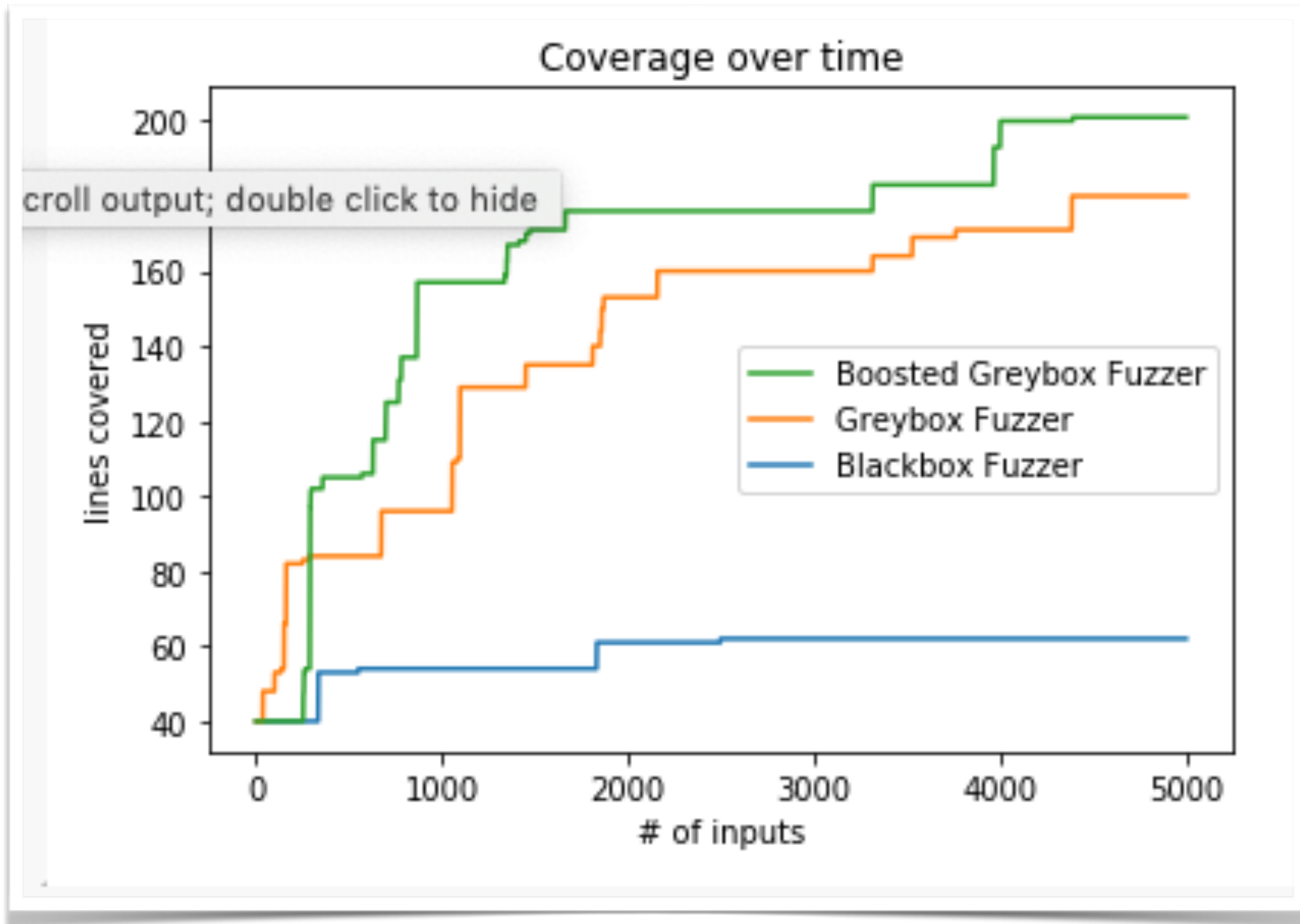
```
# create wrapper function
def my_parser(inp):
    parser = HTMLParser() # resets the HTMLParser object for every fuzz input
    parser.feed(inp)

n = 5000
seed_input = " " # empty seed
blackbox_fuzzer = MutationFuzzer([seed_input], Mutator(), PowerSchedule())
greybox_fuzzer = GreyboxFuzzer([seed_input], Mutator(), PowerSchedule())
boosted_fuzzer = CountingGreyboxFuzzer([seed_input], Mutator(), AFLFastSchedule(5))

start = time.time()
blackbox_fuzzer.runs(FunctionCoverageRunner(my_parser), trials=n)
greybox_fuzzer.runs(FunctionCoverageRunner(my_parser), trials=n)
boosted_fuzzer.runs(FunctionCoverageRunner(my_parser), trials=n)
end = time.time()

"It took all three fuzzers %0.2f seconds to generate and execute %d inputs." % (end - start, n)
```

Resultado esperado



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

favored 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

American Fuzzy Lop (AFL)

- AFL es uno de los dos fuzzers más populares
- AFL aplica una variante del algoritmo de greybox fuzzing
- AFL funciona sobre programas C/C++ que pueden ser traducidos a LLVM (para medir cobertura de bloques básicos)
- A diferencia del algoritmo de greybox fuzzing que vimos antes, AFL es determinístico.

Fuzzing con AFL

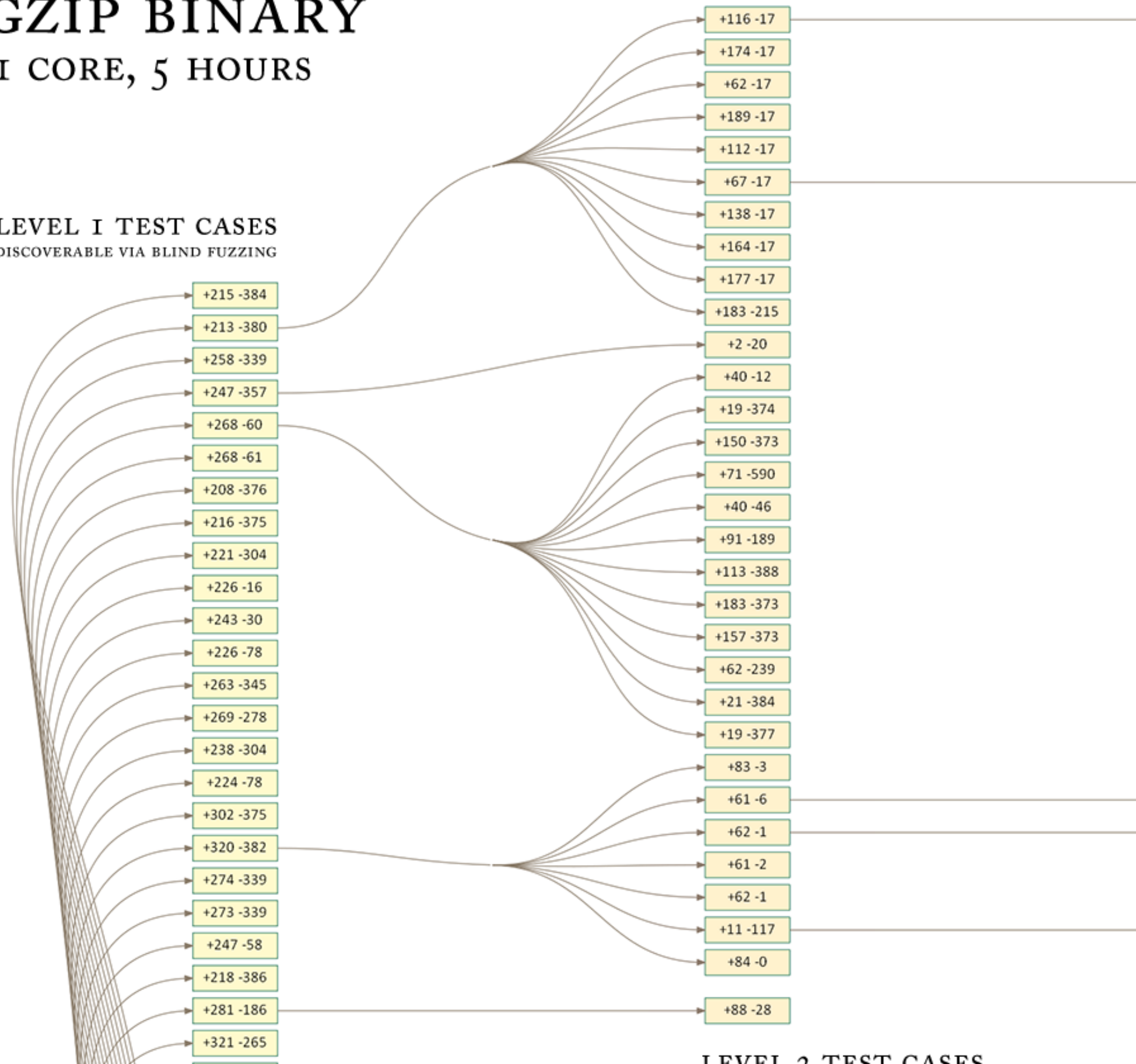
- 1) Encola **inputs iniciales** provistos por el usuario a la cola Q,
- 2) Tomar un input de la cola Q
- 3) Intentar **reducir** el input a su menor tamaño si alterar su capacidad de cobertura,
- 4) Aplicar un conjunto de mutaciones (cambios) al input utilizando una variedad de **estrategias tradicionales 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-FUZZ, GZIP BINARY

2,000 EXECs/SEC, 1 CORE, 5 HOURS

LEVEL 1 TEST CASES

DISCOVERABLE VIA BLIND FUZZING



AFL-FUZZ, GZIP BINARY
2,000 EXECS/SEC, 1 CORE, 5 HOURS

Initial test case

LEVEL 1 TEST CASES
DISCOVERABLE VIA BLIND FUZZING

- +215 -384
- +213 -380
- +258 -339
- +247 -357
- +268 -60
- +268 -61
- +208 -376
- +216 -375
- +221 -304
- +226 -16
- +243 -30
- +226 -78
- +263 -345
- +269 -278
- +238 -304
- +224 -78
- +302 -375
- +320 -382
- +274 -339
- +273 -339
- +247 -58
- +218 -386
- +281 -186
- +321 -265
- +263 -277
- +239 -43
- +228 -393
- +241 -60
- +273 -185
- +242 -60
- +260 -341
- +239 -60
- +322 -382
- +340 -194
- +228 -316
- +245 -73
- +275 -42
- +248 -40
- +248 -42
- +229 -63
- +262 -306
- +226 -80
- +222 -386
- +281 -60
- +244 -379
- +297 -205
- +244 -304
- +280 -382
- +227 -363
- +259 -341
- +306 -375
- +252 -382
- +294 -205
- +228 -361
- +238 -16
- +246 -285
- +232 -357
- +325 -382
- +224 -5
- +304 -205
- +227 -371
- +283 -339
- +228 -374
- +227 -368
- +306 -375
- +66 -372
- +219 -5
- +316 -198
- +326 -382
- +318 -382
- +239 -304
- +319 -196

LEVEL 2 TEST CASES
DERIVED BY SEEDING THE FUZZER
WITH TEST CASES ISOLATED ON
PREVIOUS LEVEL

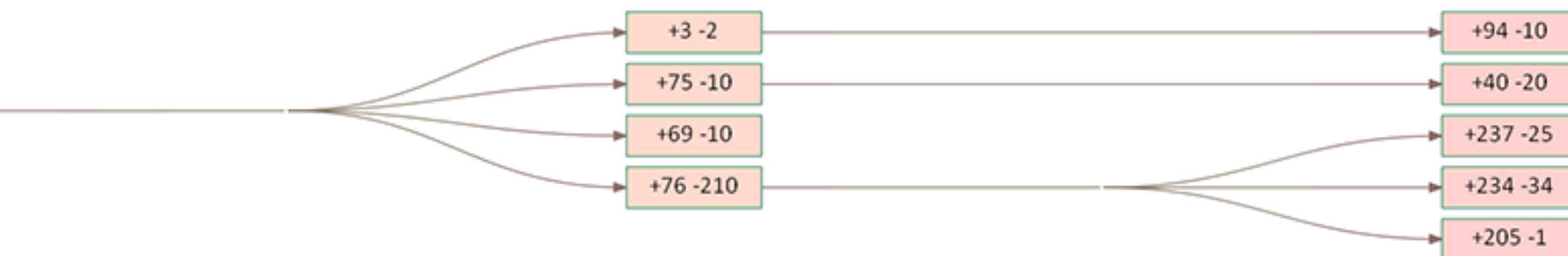
- +116 -17
- +174 -17
- +62 -17
- +189 -17
- +112 -17
- +67 -17
- +138 -17
- +164 -17
- +177 -17
- +183 -215
- +2 -20
- +40 -12
- +19 -374
- +150 -373
- +71 -590
- +40 -46
- +91 -189
- +113 -388
- +183 -373
- +157 -373
- +62 -239
- +21 -384
- +19 -377
- +83 -3
- +61 -6
- +62 -1
- +61 -2
- +62 -1
- +11 -117
- +84 -0
- +88 -28
- +47 -0
- +32 -66
- +92 -293
- +81 -220
- +16 -229
- +19 -55
- +66 -20
- +93 -2
- +100 -2
- +94 -2
- +74 -2
- +47 -8
- +101 -221
- +116 -183
- +23 -8
- +142 -188
- +24 -16

VALUES CORRESPOND TO THE NUMBER OF NEW
STATE TRANSITIONS (TUPLES) APPEARING IN THE
ISOLATED TEST CASE, COMPARED TO PARENT.

LEVEL 6 TEST CASES

- +3 -2
- +75 -10
- +69 -10
- +76 -210
- +94 -10
- +40 -20
- +237 -25
- +234 -34
- +205 -1

LEVEL 6 TEST CASES



Estrategias de mutación de AFL

- **Walking bit & byte flips:** Flipping un bit, dos bits, etc.
- **Simple arithmetics:** incrementar o decrementar valores enteros existents en el input
- **Known integers:** e.g., -1, 256, 1024, MAX_INT-1, MAX_INT
- **Stacked tweaks:** borrar, duplicar, insertar bloques

AFL Strategies

- <https://lcamtuf.blogspot.com.ar/2014/08/binary-fuzzing-strategies-what-works.html>
- Las estrategias de mutación de inputs de AFL son operadores de mutación que provienen de la comunidad de seguridad (no de la comunidad de lenguajes de programación).
- Entonces, en su mayoría son manipulaciones de bits.

Más Allá de los Crashes

- Un programa crashea únicamente si hay un problema serio
 - Hangs, Segmentation fault, assertion failure, etc.
- Pero puede sobrevivir a defectos **latentes**
 - Memory Leaks, Index Out of bounds, Incorrecto Manejo de Punteros, etc.

Sanitizers

- Son frameworks que instrumentan un programa para realizar chequeos en tiempo de ejecución
- Existen distintos sanitizers para C/C++
- <https://github.com/google/sanitizers>

AddressSanitizer (ASan)

- Framework de Instrumentación para detectar múltiples clases de corrupción del manejo de Memoria
 - Heap/stack buffer overflows, use-after-free bugs
- Se puede combinar con cualquier Fuzzer siempre y cuando podamos instrumentar el programa

Undefined Behaviour Sanitizer (UBSan)

- Detecta en runtime operaciones cuya semántica no está bien descripta en C/C++
- Ejemplos:
 - Oversized Shift Amounts
 - Dereferencing a NULL Pointer
 - Violating Type Rules
 - Use of an uninitialized variable
 - Signed integer overflow

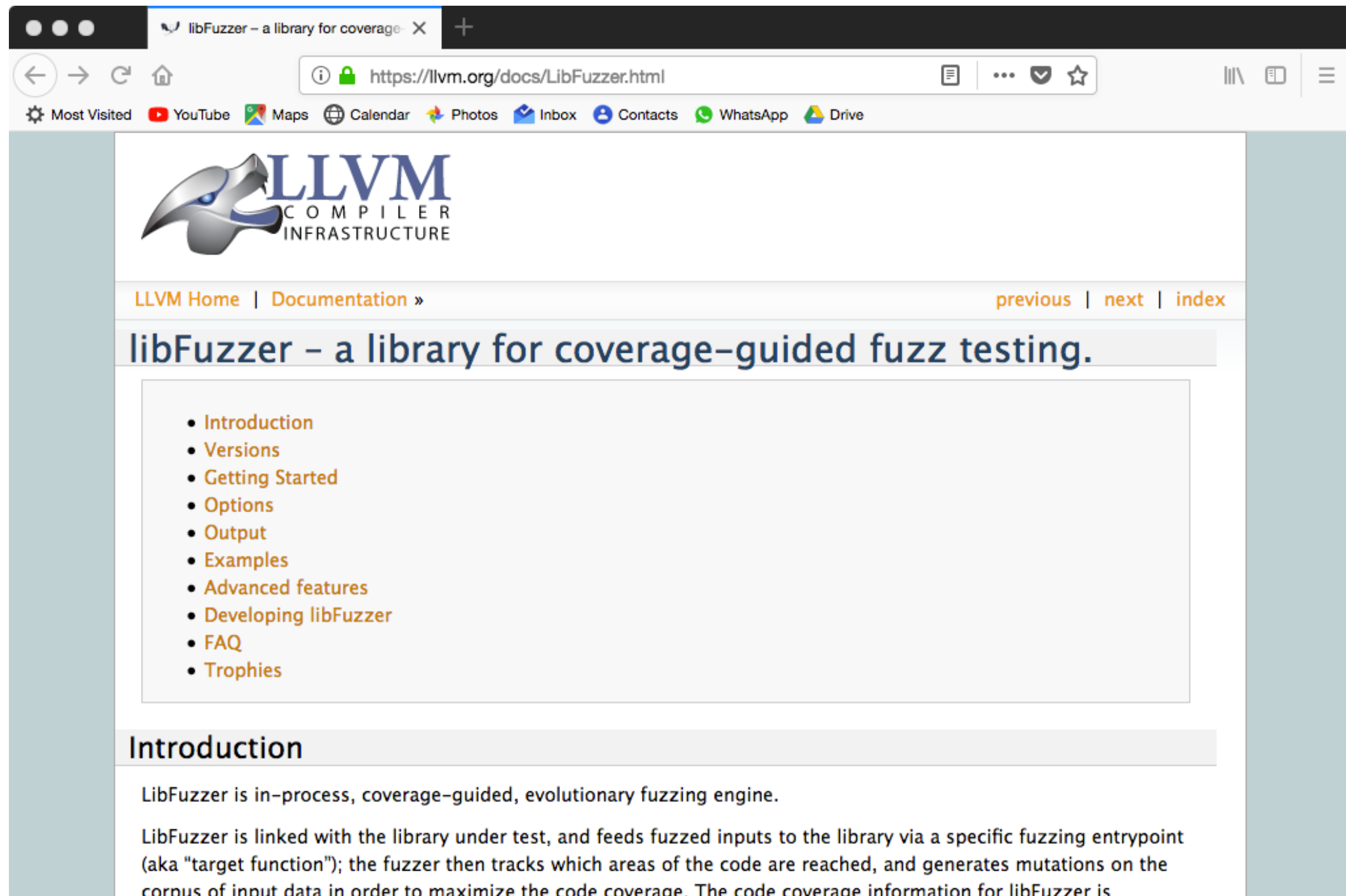
Memory Sanitizer (MSan)

- Detecta un-initialised reads en el heap
- Usualmente introduce un alentecimiento de 3x

Leak Sanitizer (LSan)

- Detecta memory leaks

LibFuzzer



The screenshot shows a web browser window with the address bar displaying `https://llvm.org/docs/LibFuzzer.html`. The page header features the LLVM logo and navigation links: [LLVM Home](#), [Documentation »](#), [previous](#), [next](#), and [index](#). The main heading is **libFuzzer – a library for coverage-guided fuzz testing.**

A list of links is provided for navigation:

- [Introduction](#)
- [Versions](#)
- [Getting Started](#)
- [Options](#)
- [Output](#)
- [Examples](#)
- [Advanced features](#)
- [Developing libFuzzer](#)
- [FAQ](#)
- [Trophies](#)

The **Introduction** section begins with the text: "LibFuzzer is in-process, coverage-guided, evolutionary fuzzing engine."

LibFuzzer is linked with the library under test, and feeds fuzzed inputs to the library via a specific fuzzing entrypoint (aka "target function"); the fuzzer then tracks which areas of the code are reached, and generates mutations on the corpus of input data in order to maximize the code coverage. The code coverage information for libFuzzer is

LibFuzzer

- AFL: Fuzzea solo sistemas enteros
- LibFuzzer: permite fuzzear una librería
- Necesita implementar una función que transforma un arreglo de bytes en un input válido de la librería

```
// fuzz_target.cc
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    DoSomethingInterestingWithMyAPI(Data, Size);
    return 0; // Non-zero return values are reserved for future use.
}
```

LibFuzzer

- El código debe poder tolerar cualquier input de data
- No debe haber llamados a `exit()`
- No debe utilizar threads/no determinismo
- Tan rápido como se pueda (el target)
- No modificar información global (static)

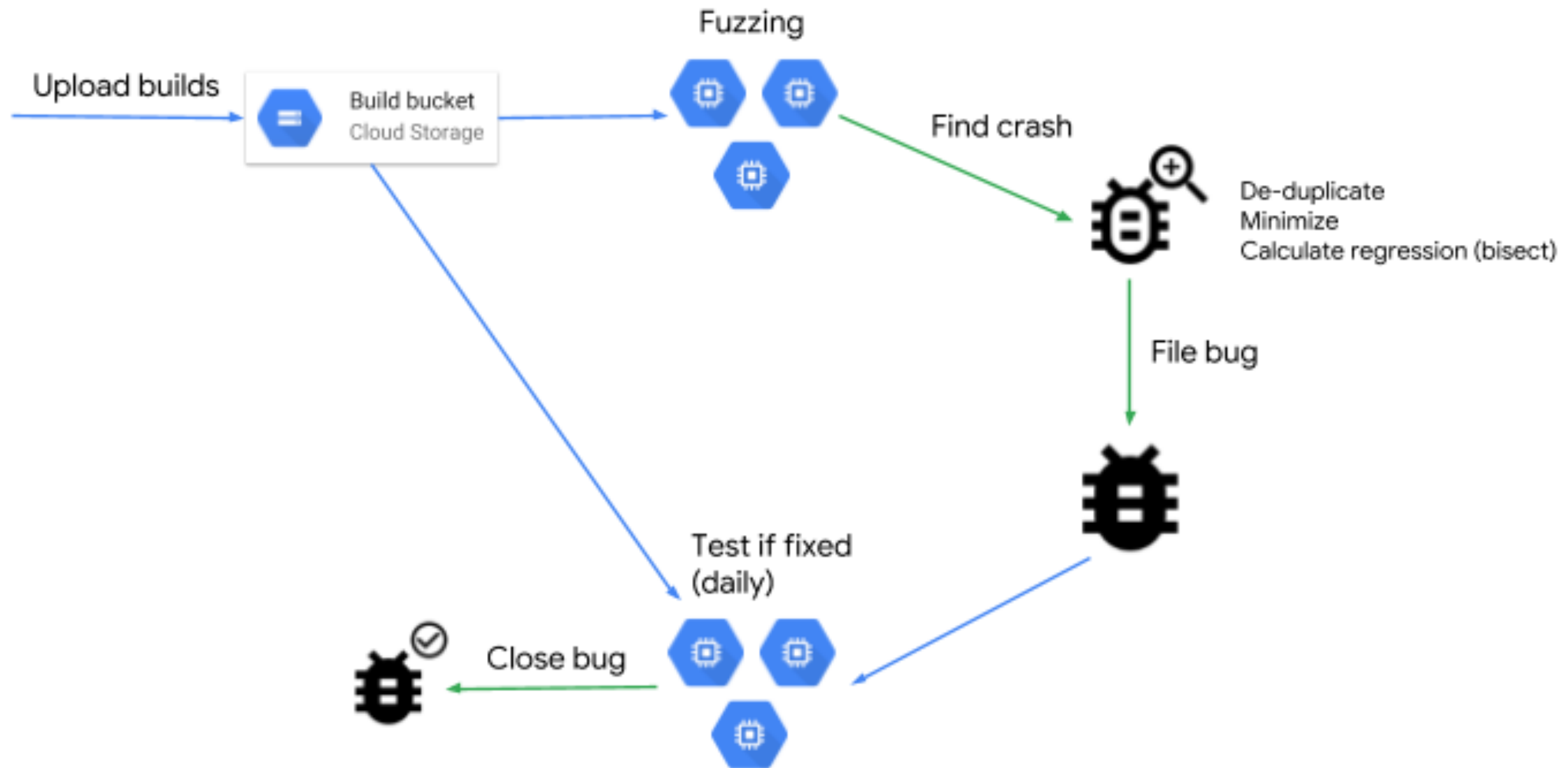


ClusterFuzz

- Infraestructura para ejecutar fuzzing sobre software
- Google usa ClusterFuzz para fuzzear Chrome Browser
- Scalable. El cluster fuzz de Google's corre sobre 25,000 máquinas.
- Enero 2019, ClusterFuzz detectó ~16,000 bugs en Chrome



ClusterFuzz



HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).



...this pages about "boats". User Erica requests
secure connection using key "4538538374224".
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H



...this pages about "boats". User Erica requests
secure connection using key "4538538374224".
User Meg wants these 6 letters: **POTATO**. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H



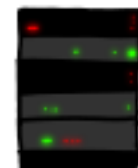
POTATO

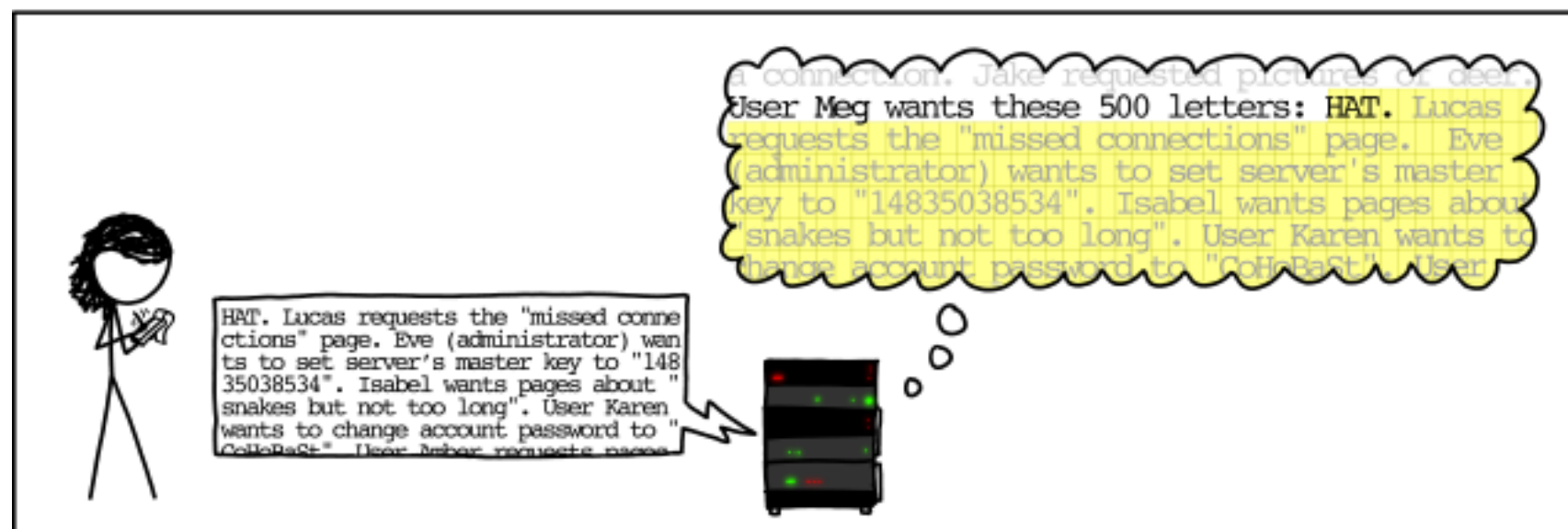
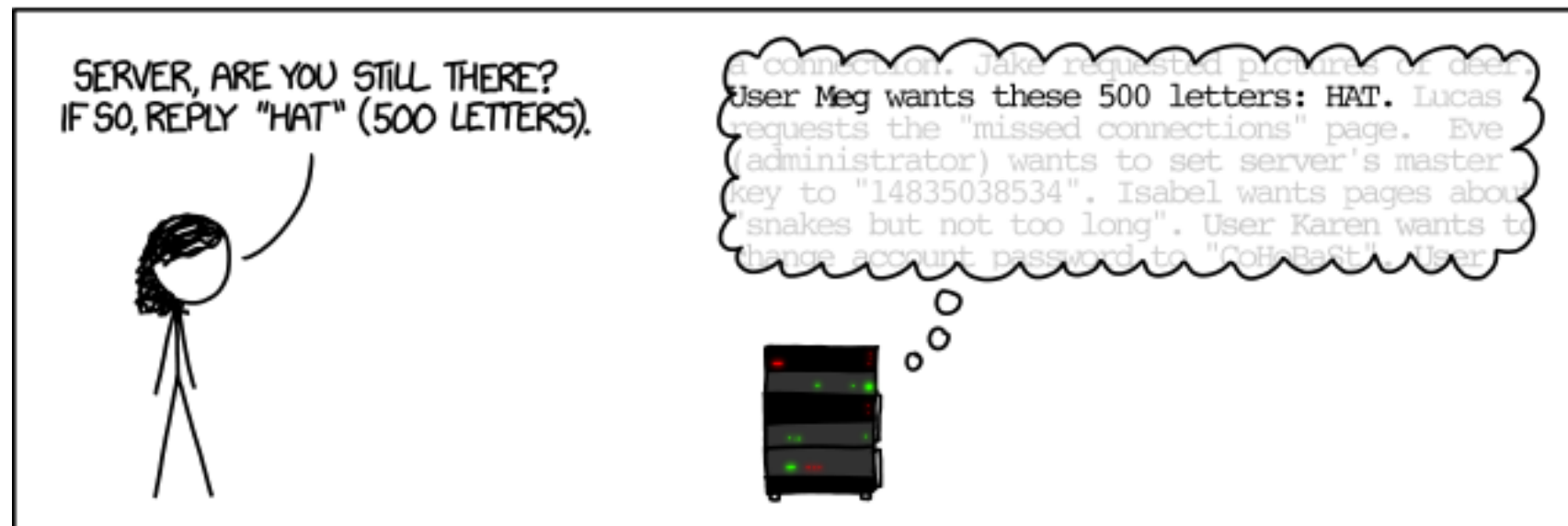
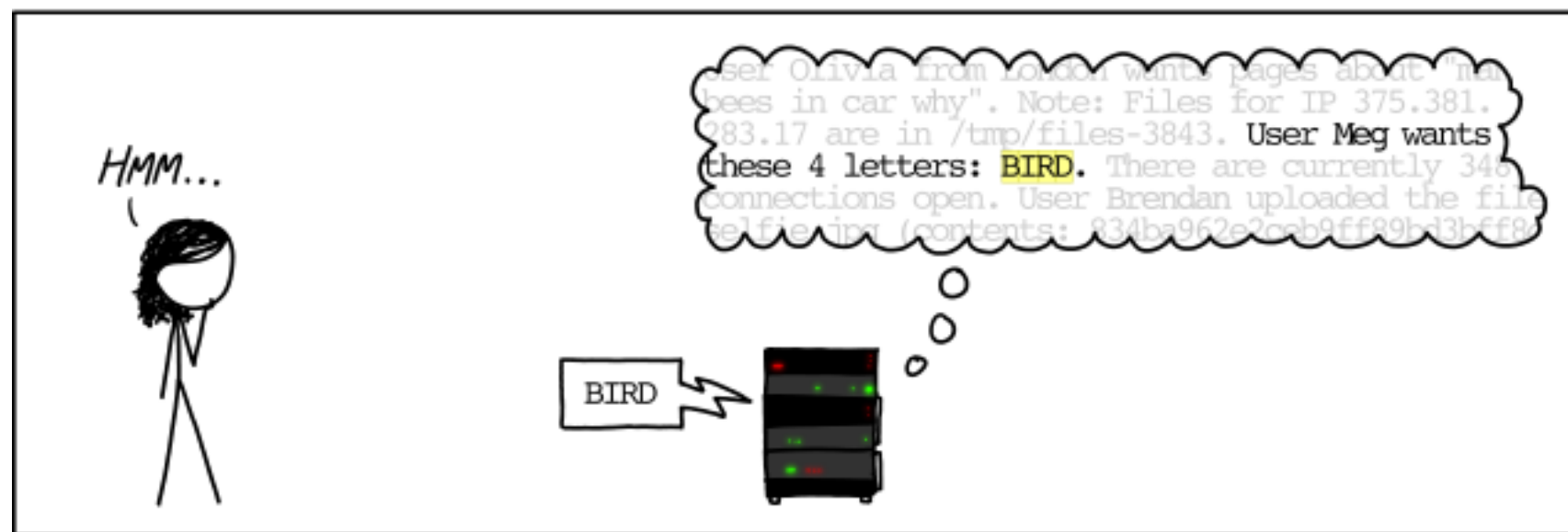


SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 348
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2c0eb9ff89b43b6f8





Heartbleed

- Dic. 2013: primer release de AFL
- Nov 2014: primer release de LibFuzzer
- Heartbleed:
 - AFL: 6 horas
 - Libfuzzer: 10 segs



Heartbleed

- Heartbleed se introduce en Dic. 2011
- Recien se detecta en Marzo 2014
- Si era fácil de detectar por AFL, ¿por qué no se detectó?



Heatbleed

- OpenSSL era un proyecto bien financiando
- Las herramientas de fuzzing existían
- El fuzzing no lo efectuaban los developers, sino los security researchers



OSS-Fuzz - Continuous Fuzzing for Open Source Software

Status: Stable. We are accepting applications from widely-used open source projects.

[FAQ](#) | [Ideal Fuzzing Integration](#) | [New Project Guide](#) | [Reproducing Bugs](#) | [Projects](#) | [Projects Issue Tracker](#) | [Glossary](#)

[Create New Issue](#) for questions or feedback about OSS-Fuzz.

Introduction

[Fuzz testing](#) is a well-known technique for uncovering various kinds of programming errors in software. Many of these detectable errors (e.g. [buffer overflow](#)) can have serious security implications.

We successfully deployed [guided in-process fuzzing of Chrome components](#) and found [hundreds](#) of security vulnerabilities and stability bugs. We now want to share the experience and the service with the open source community.

In cooperation with the [Core Infrastructure Initiative](#), OSS-Fuzz aims to make common open source software more secure and stable by combining modern fuzzing techniques and scalable distributed execution.

At the first stage of the project we use [libFuzzer](#) with [Sanitizers](#). More fuzzing engines will be added later. [ClusterFuzz](#) provides a distributed fuzzer execution environment and reporting.

Currently OSS-Fuzz supports C and C++ code (other languages supported by [LLVM](#) may work too).

OSS-Fuzz

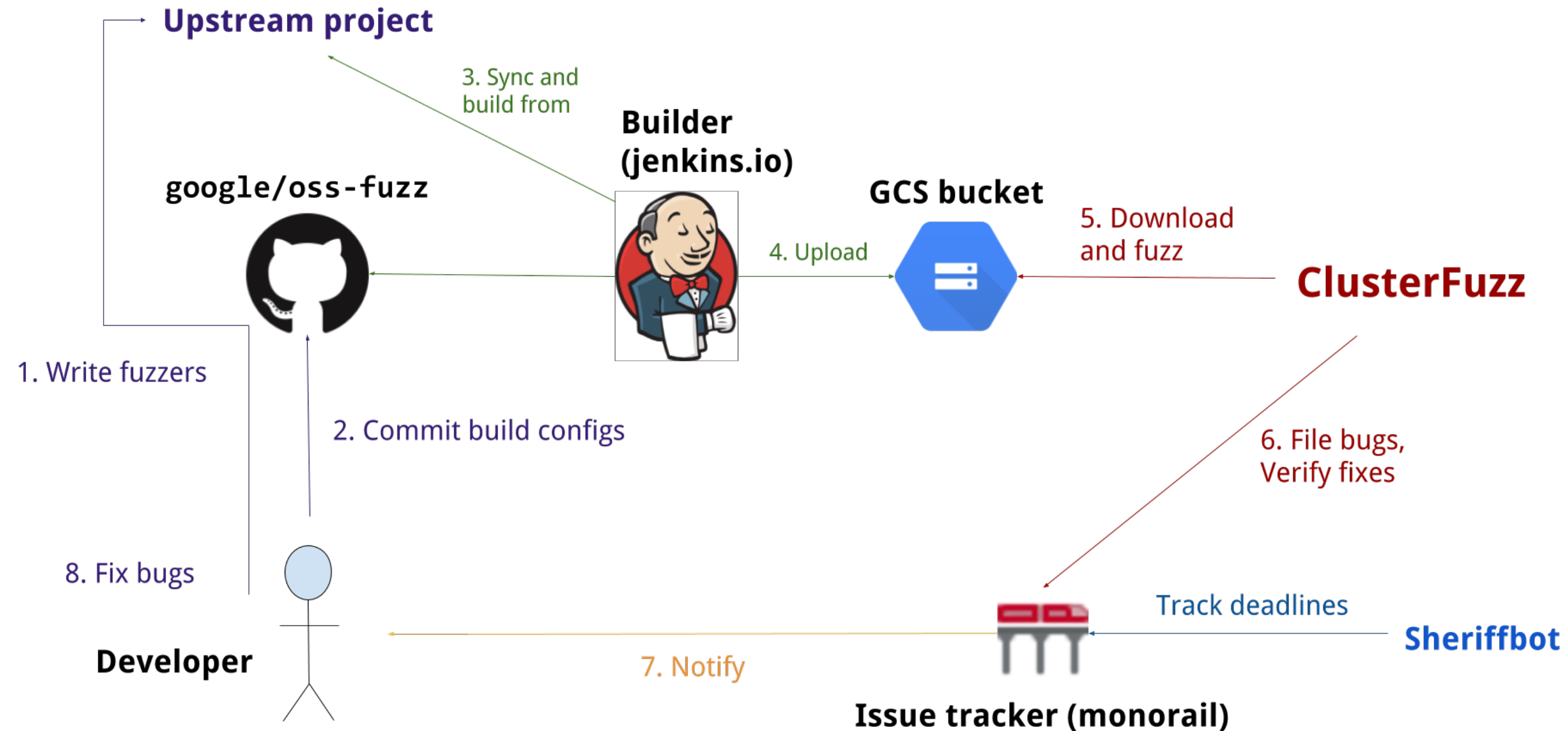
- Idea: suministrar infraestructura y tiempo de cómputo para fuzzear proyectos open-source importantes
- El proyecto Open-Source debe preparar el sistema para facilitar el fuzzing (libFuzzer/AFL)
- OSS-Fuzz fuzzea el proyecto y reporta vulnerabilidades y problemas detectados (ASan, UBSan, MSan)
- El proyecto Open-Source corrige el sistema

OSS-Fuzz: proyectos

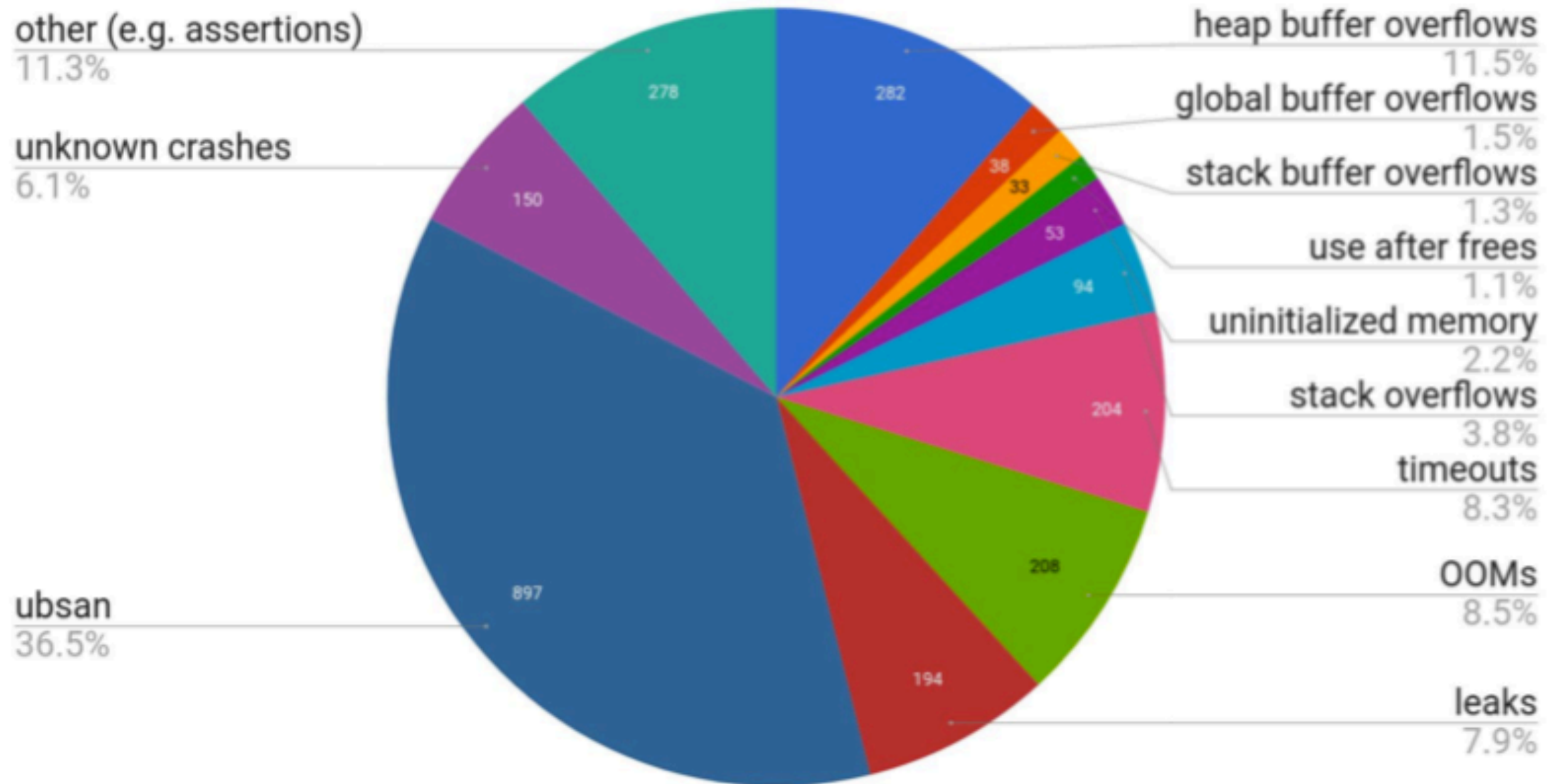
- Cpython2, cpython3
- Ffmpeg
- Coreutils
- Firefox
- Ghostscript
- Glib
- Golang
- Libmpeg2
- Libreoffice
- Lvm
- nodejs
- openssh
- openssl
- qt
- Sqlite3
- Syzkaller
- Tinyxml2
- Wget
- zlib
- ...

[https://github.com/google/oss-fuzz/tree/master/
projects](https://github.com/google/oss-fuzz/tree/master/projects)

OSS-Fuzz (2016)



OSS-Fuzz (>2000 bugs)



OSS-Fuzz: Ideal Integration

- Los tests targets son mantenidos por los programadores como parte de su codebase
- Existe un seed con buen coverage inicial
- Los fuzz targets son en general rápidos y no tienen OOM
- Provee un diccionario de fuzzing (constantes para ser usadas durante el fuzzing)



OSS-Fuzz: lifecycle

- AFL/Libfuzzer detectan una falla. Esta es de-duplicada contra los bugs existentes
- El input que produce la falla es minimizado
- El bug report es enviado privadamente a los programadores del proyecto
- Cada 24 horas el input es ejecutado contra la última versión del proyecto.
- Fixed: el bug report se hace público 30 días despues del fix
- Not-fixed: el bug report se hace público luego de 90 días

Bug report generado automáticamente

Issue 2445
Starred by 2 users

gdal: Heap-buffer-overflow in PCIDSK::CBandInterleavedChannel::ReadBlock
Project Member Reported by monor...@clusterfuzz-external.iam.gserviceaccount.com, Jul 1

Status: Verified
Owner: ----
Closed: Jul 2
Cc:  bi...@gmail.com
 m...@net
 s...@gmail.com
 e'...@gmail.co
Type: Bug-Security
ClusterFuzz
Stability-Memory-AddressSanitizer
Reproducible
ClusterFuzz-Verified
Engine-libfuzzer
OS-Linux
Proj-gdal
Reported-2017-07-01

Detailed report: <https://oss-fuzz.com/testcase?key=4766641567039488>
Project: gdal
Fuzzer: libFuzzer_gdal_filesystem_fuzzer
Fuzz target binary: gdal_filesystem_fuzzer
Job Type: libfuzzer_asan_gdal
Platform Id: linux
Crash Type: Heap-buffer-overflow READ 4
Crash Address: 0x602000008855
Crash State:
PCIDSK::CBandInterleavedChannel::ReadBlock
PCIDSK2Band::IReadBlock
GDALRasterBand::GetLockedBlockRef
Sanitizer: address (ASAN)
Recommended Security Severity: Medium
Regressed: https://oss-fuzz.com/revisions?job=libfuzzer_asan_gdal&range=201705291647:201705301648
Reproducer Testcase: https://oss-fuzz.com/download?testcase_id=4766641567039488

OSS-Fuzz

[oss-fuzz](#) [New issue](#) [All issues](#)

1 - 100 of 14194 [Next >](#) [List](#) [Grid](#)

ID ▼	Type ▼	Component ▼	Status ▼	Proj ▼	Reported ▼	Owner ▼	Summary + Labels ▼	Opened ▼
17098	Bug	----	New	llvm	2019-09-09	----	llvm:llvm-itanium-demangle-fuzzer: Out-of-memory in llvm_llvm-itanium-demangle-fuzzer ClusterFuzz Reproducible	20 hours ago
17091	Bug	----	New	llvm	2019-09-09	----	llvm:llvm-opt-fuzzer--x86_64-strength_reduce: ASSERT: !BaseRegs.empty() && "1*reg => reg, should not be needed." ClusterFuzz Reproducible	31 hours ago
17067	Build-Failure	----	New	iroha	----	----	iroha: Coverage build failure	2 days ago
17066	Build-Failure	----	Verified	uwebsockets	----	----	uwebsockets: Fuzzing build failure	2 days ago
17065	Build-Failure	----	New	php	----	----	php: Fuzzing build failure	2 days ago
17064	Build-Failure	----	Verified	perfetto	----	----	perfetto: Fuzzing build failure	2 days ago
17063	Build-Failure	----	New	openvswitch	----	----	openvswitch: Fuzzing build failure	2 days ago
17062	Build-Failure	----	New	cryptofuzz	----	----	cryptofuzz: Fuzzing build failure	2 days ago
17049	Bug	----	New	llvm	2019-09-07	----	llvm:llvm-itanium-demangle-fuzzer: ASSERT: Parser->TemplateParams.size() >= OldNumTemplateParamLists ClusterFuzz Reproducible	3 days ago
17048	Build-Failure	----	Verified	minizinc	----	----	minizinc: Fuzzing build failure	3 days ago
17047	Build-Failure	----	New	grpc	----	----	grpc: Fuzzing build failure	3 days ago
17046	Build-Failure	----	New	chakra	----	----	chakra: Fuzzing build failure	3 days ago

Agosto 2019: OSS-Fuzz encontró > **14,000** bugs en **200** proyectos open acreditados.

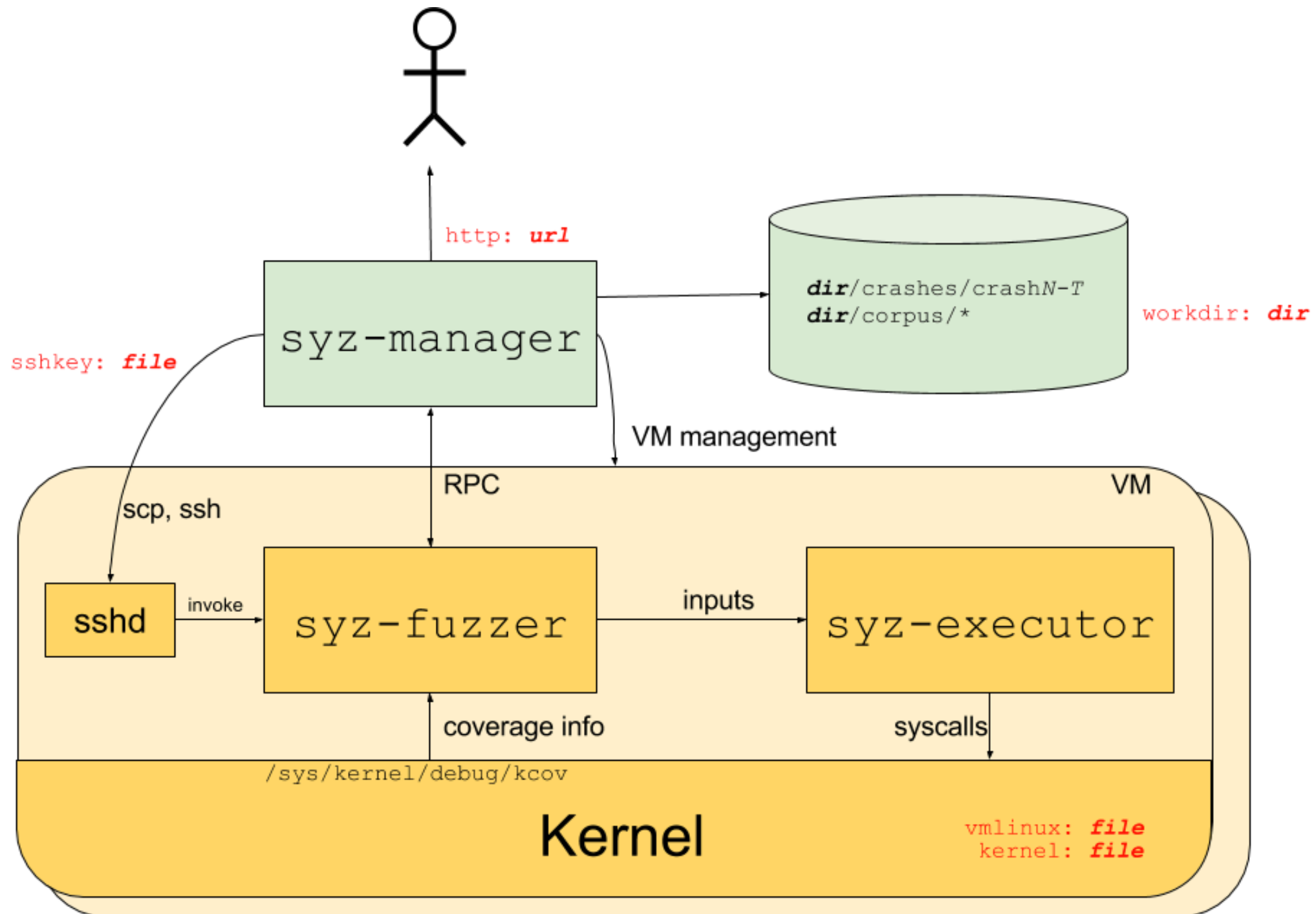
Patch reward programs

- Hasta \$20,000: ideal integration con OSS-Fuzz
- \$10,000 por bug-fixes complicados de alto impacto en vulnerabilidad
- \$5,000 por bug-fixes moderados
- \$1,337 por submitir fixes de complejidad baja
- \$500 "one-liner specials" para fixes sencillos con impacto en seguridad

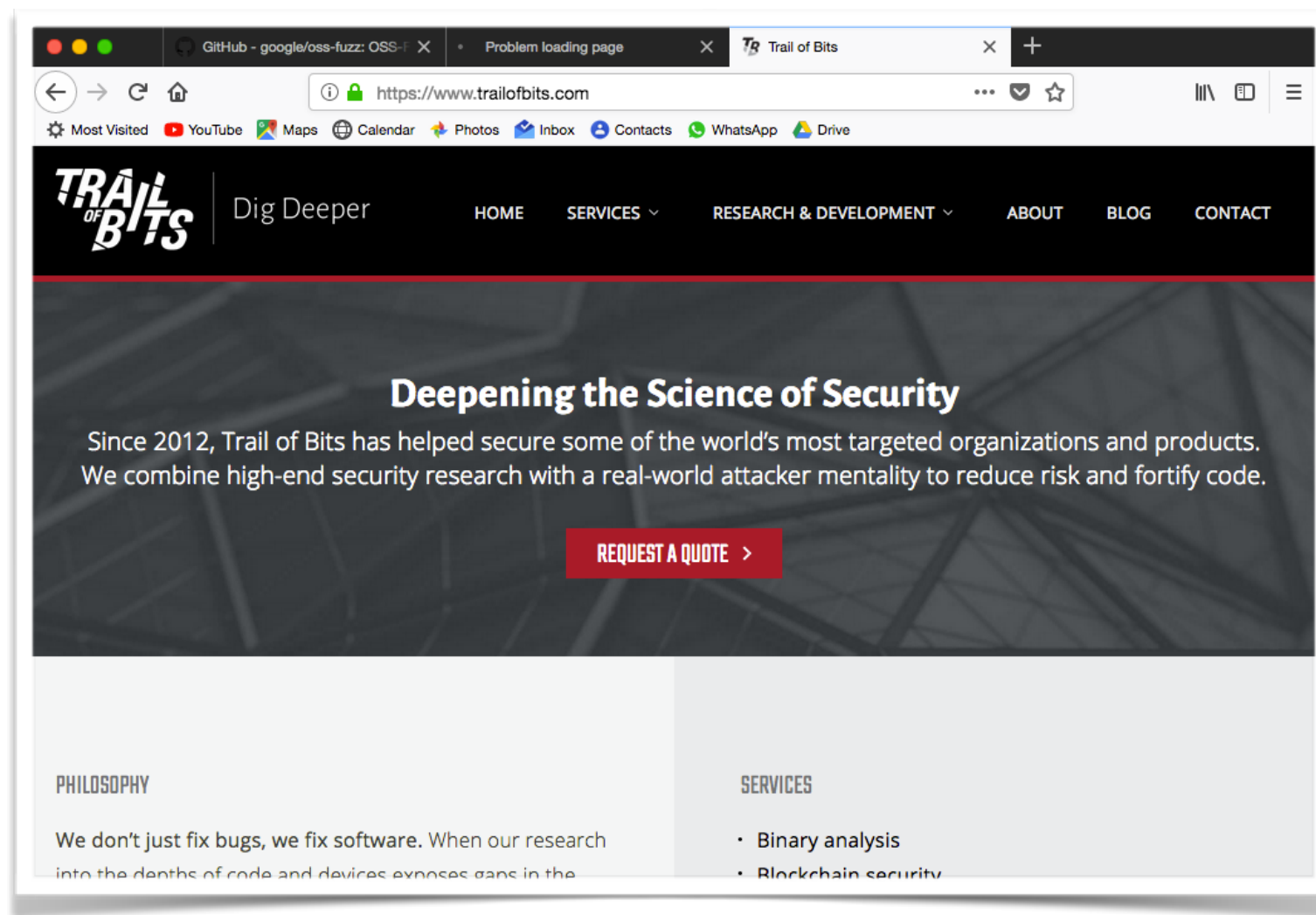
Syzkaller

- AFL/Libfuzzer fuzzean código de una aplicación
- ¿Qué pasa si la vulnerabilidad no reside en un defecto en el programa, sino en el sistema operativo (i.e. kernel)?
- Syzkaller es un fuzzer especializado para hacer coverage-based fuzzing de kernel code
 - ¿Cómo medimos cobertura en código kernel?
 - ¿Cómo instrumentamos el sistema operativo?
 - Usa VMs para invocar al sistema operativo objetivo

Syzkaller



- Fuzzing no se realiza únicamente en Google, Mozilla. También hay empresas especializadas en fuzzing (ejemplo: Trail of Bits)
- <https://www.trailofbits.com>
- <https://twitter.com/trailofbits>



Recap

- Blackbox Fuzzing
- Greybox Fuzzing
- Boosted Greybox Fuzzing
- AFL / LibFuzzer (LLVM/C) + ASan + asserts
- ClusterFuzz
- OSS-Fuzz

Fuzz-Driven Development

- 2003: Test Driven Development (Kent Beck)
 - Insuficiente para seguridad
- 2017: Fuzz-Driven Development (Kostya Serebryany)
 - Cada API es un fuzz target
 - Tests == “Seed” para fuzzear
 - CI incluye Continuous Fuzzing

Bibliografía

- The Fuzzing Book (<https://www.fuzzingbook.org>) by Zeller, Gopinath, Boehme, Fraser and Holler
- Capítulo 2 Lexical Fuzzing - Greybox Fuzzing