Fuzzing@Home: *Distributed Fuzzing on Untrusted Heterogeneous Clients*

-The 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID2022)

Daehee Jang, Ammar Askar, Insu Yun, Stephen Tong, Yiqin Cai, Taesoo Kim

SSLab @GeorgiaTech

Large-Scale Fuzzing

- **There are so many codes to fuzz/test**
  - OSSFuzz has more than 300 open-source projects ported for fuzzing
  - Google use ClusterFuzz: immense distributed fuzzing infrastructure
    - ✓ Mainly inspired from ClusterFuzz

..	
abseil-cpp	Fill in main_repo for several projects. (#4816)
alembic	Fill in main_repo for several projects. (#4816)
apache-commons	Fix builds after Jazzer breaking change (#6622)
apache-httpd	apache-httpd: fix build (#6626)
arduinojson	Populate a bunch of main_repo values. (#4815)
arrow	[arrow] Add contact (#5033)
aspell	Populate a bunch of main_repo values. (#4815)
assimp	assimp: switch to new base builder (#6448)
astc-encoder	Fill in main_repo for several projects. (#4816)
augeas	Populate a bunch of main_repo values. (#4815)
avahi	Fill in main_repo for several projects. (#4816)

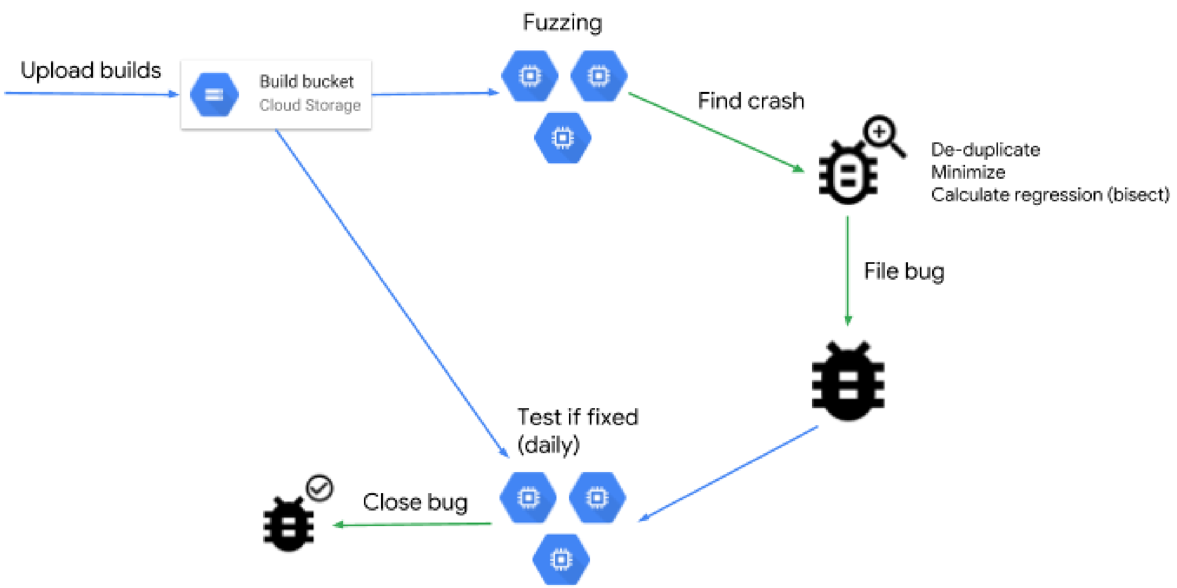
Background - ClusterFuzz

❖Google's Large-Scale Distributed Fuzzing System

- ~ 30,000 VM Instances
- ~ 340 open source fuzz targets running
- ~ 25,000 bugs discovered.

❖Designed as Private Infrastructure

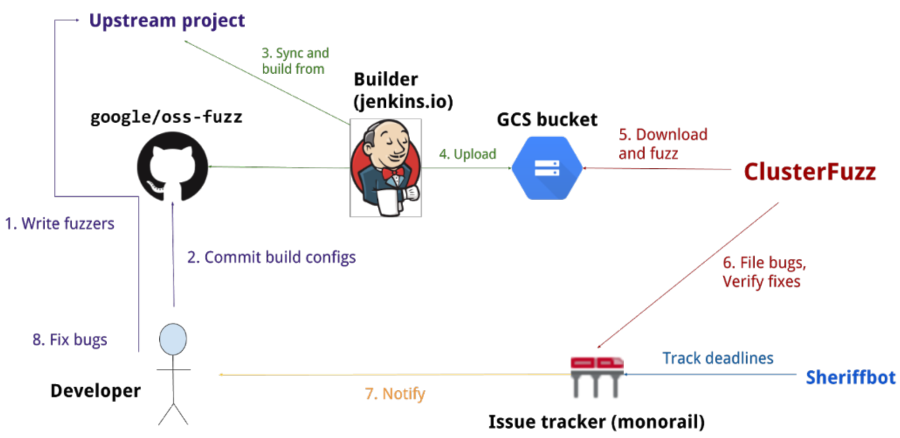
- Single owner (Google) controls overall infrastructure/results



Fuzzing@Home - Motivation

❖ Why not apply "@home" idea to fuzzing?

- Fuzzing works better in parallel
- People can utilize spare computing power for fuzzing
- Organizations can collaborate for fuzz-testing their product
  - ✓ Multiple companies develop software together
  - ✓ Multiple companies do bug-bounty together

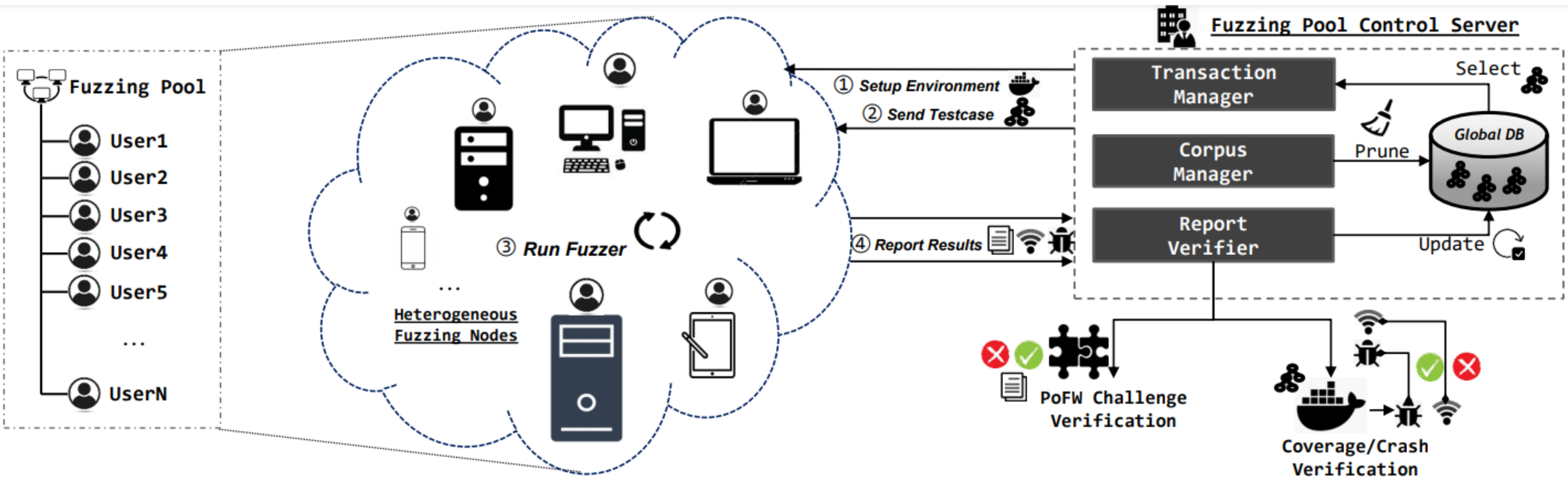


FOLDING @HOME

SETI HOME

Introduction & Design

Fuzzing@Home Overview



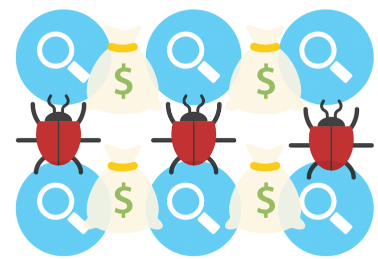
❖ Components

- Fuzzing Pool: Group of people (nodes) fuzzing the same target
- Fuzzing Node: Organization/People's computing device (PC, laptop, mobile, …)
  - ✓ Heterogeneous, Untrusted
- Control Server: Fuzzing pool master
  - ✓ Verification, Deduplication, Scheduling optimization…

Fuzzing@Home – Security Problem

❖ **Collaborative *"public"* network infrastructure for fuzzing**

- Collaborating participants are untrusted
- Fuzzing may involve money



- How do we tell if a participant is working?
  - ✓ -> Goofing Problem

❖ **Solution: Proof-of-Work (PoW) for fuzzing**

- Design Proof-of-Fuzzing-Work (PoFW)

Fuzzing@Home – Security Problem

❖PoW vs PoFW?

- Existing PoW computations have estimated time to get result
  - ✓ E.g., Breaking RSA-XXX with CPU-YYY usually takes ZZZ hours.
- Existing PoW computations gives output data as a computing result (challenge user)
  - ✓ E.g., Bitcoin mining (hash)
  - ✓ E.g., Cryptographic algorithm (decrypted data)

- Fuzzing has no estimated time to get result
  - ✓ E.g., Crashing chrome-v8 with CPU-YYY usually takes ZZZ hours..??
- Fuzzing do not yield result output data in its execution (can't challenge user)
  - ✓ E.g, *void* function
- Idea: Use code-coverage as proof-of-work in fuzzing
  - ✓ Fuzzing always takes input data -> produce code-coverage

Proof-of-Work tailored for Fuzzing

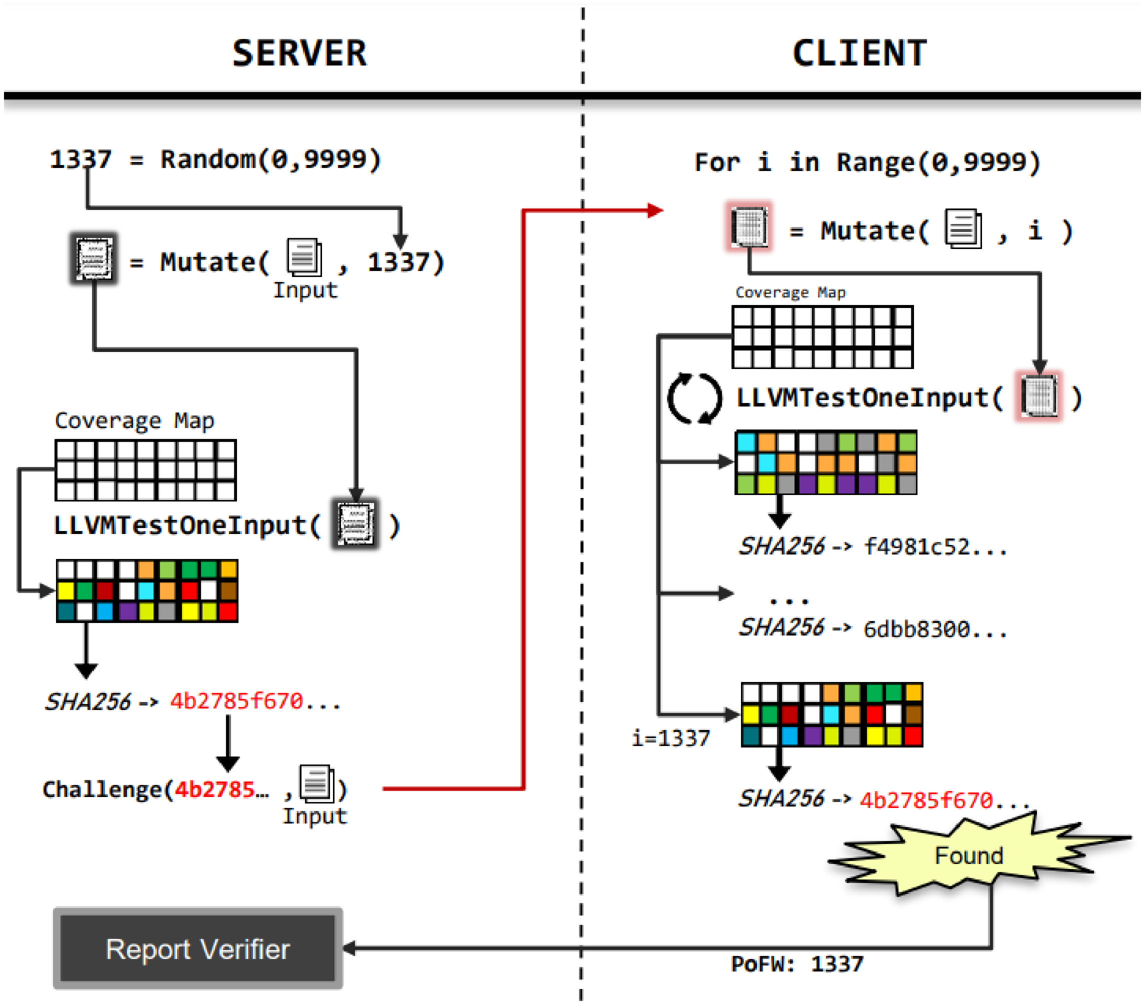
❖ Proof of Fuzzing Work?

- Hash code-coverage information into a single SHA512 string
- "execution hash", use it as fingerprint
  - ✓ SHA512 of code coverage information

❖ Steps

- 1. Control server randomly picks a seed number and initial fuzzing input
- 2. Control server pre-calculate a single "execution hash"
- 3. Control server challenge a node to find the same seed number as an answer
  - ✓ range of seed number and fuzzing input is given
- 4. Node exhaustively search possible seed numbers
  - ✓ Finding seed number is guaranteed if all numbers are tried
  - ✓ Control server verify result in O(1) time/memory complexity

PoFW Overview



Face two problems in “execution hash”: Hash collision, Non-determinism

SSLab @GeorgiaTech

Challenge in PoFW design

- **Hash Collision**
  - Different input, but same code coverage
  - Depends on "complexity" of target application
    - ✓ Need evaluation

- **Non-Determinism**
  - Same input but different code coverage
  - Also depends on "complexity" of target application
    - ✓ Need evaluation

- **PoFW needs**
  - Low collision rate
  - Low non-determinism rate

Evaluation – PoFW Hash Collision

Project	1st	2nd	3rd	Project	1st	2nd	3rd
arrow	7.3%	6.6%	5.9%	lame	1.6%	1.0%	0.1%
binutils	21.5%	14.7%	13.3%	libmpeg2	0.3%	0.2%	0.1%
capstone	0.8%	0.4%	0.1%	libpcap	37.1%	5.6%	2.2%
c-ares	33.8%	5.6%	1.8%	libpng-proto	11.6%	0.9%	0.5%
eigen	32.4%	18.6%	14.6%	libtiff	10.0%	3.6%	2.8%
ffmpeg	0.6%	0.2%	0.1%	libzip	1.7%	0.8%	0.4%
flac	6.2%	5.4%	3.0%	lodepng	26.8%	23.8%	17.3%
freeimage	1.4%	1.2%	1.0%	matio	25.5%	8.1%	7.0%
gfwx	32.6%	5.4%	3.4%	mruby	1.5%	0.2%	0.1%
giflib	31.4%	9.8%	2.8%	ntp	26.7%	6.4%	5.6%
htslib	2.1%	0.3%	0.1%	php	18.3%	2.9%	0.3%
jansson	4.1%	4.0%	3.2%	wavpack	2.2%	0.1%	0.1%
kcodec	0.6%	0.4%	0.1%	zlib	0.2%	0.1%	0.1%

1st: Highest percentage of duplicated hashes
2nd: 2nd Highest percentage of duplicated hashes
3rd: 3rd Highest percentage of duplicated hashes

Table 1. Three highest hash-duplication-ratios among 1M executions. Inputs are auto-generated by libfuzzer mutation from empty corpus. If the change of input is too small, program will take exact same code path; producing same coverage map.

Evaluation – PoFW Nondeterminism

Project	# execution	Project	# execution
arrow	63K	lame	16K
binutils	125K	libmpeg2	14K
capstone	54K	libpcap	387K
c-ares	unseen	libpng-proto	492K
eigen	unseen	libtiff	318K
ffmpeg	233K	libzip	404K
flac	unseen	lodepng	unseen
freeimage	69K	matio	341K
gfwx	516K	mruby	23K
giflib	582K	ntp	unseen
htslib	462K	php	93K
jansson	unseen	wavpack	65K
kcodecs	7K	zlib	120K

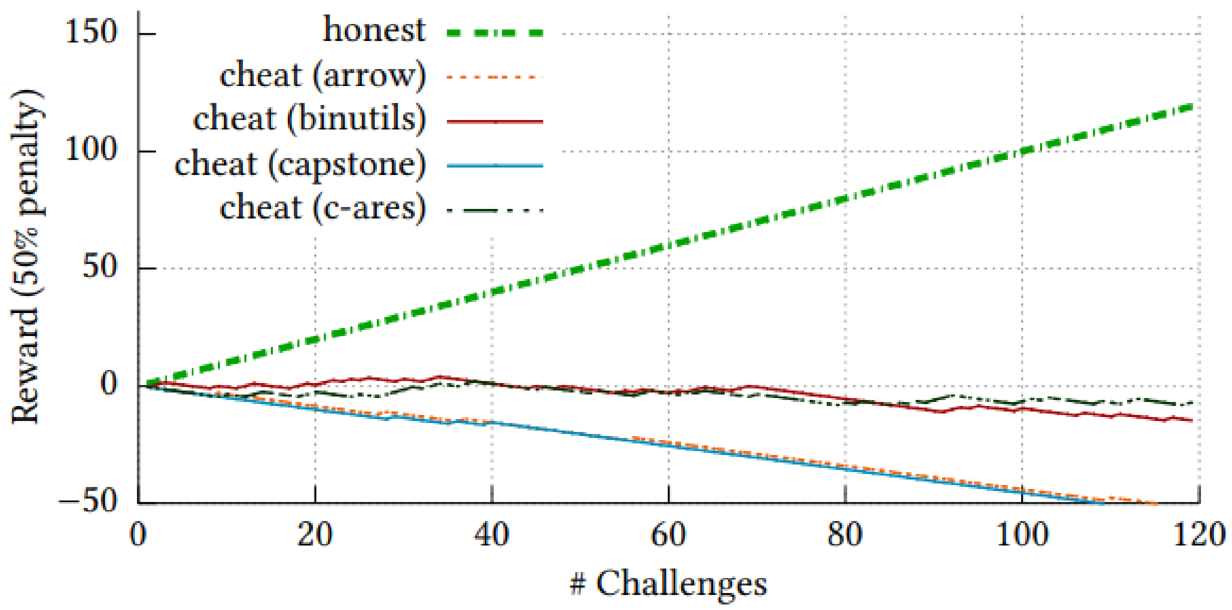
 # execution: Number of executions until first hash deviation is observed.
unseen: Deviation not observed within 1M executions.

Table 2. Due to the non-determinism, a program could yield different coverage map even with the same condition.

SSLab @GeorgiaTech

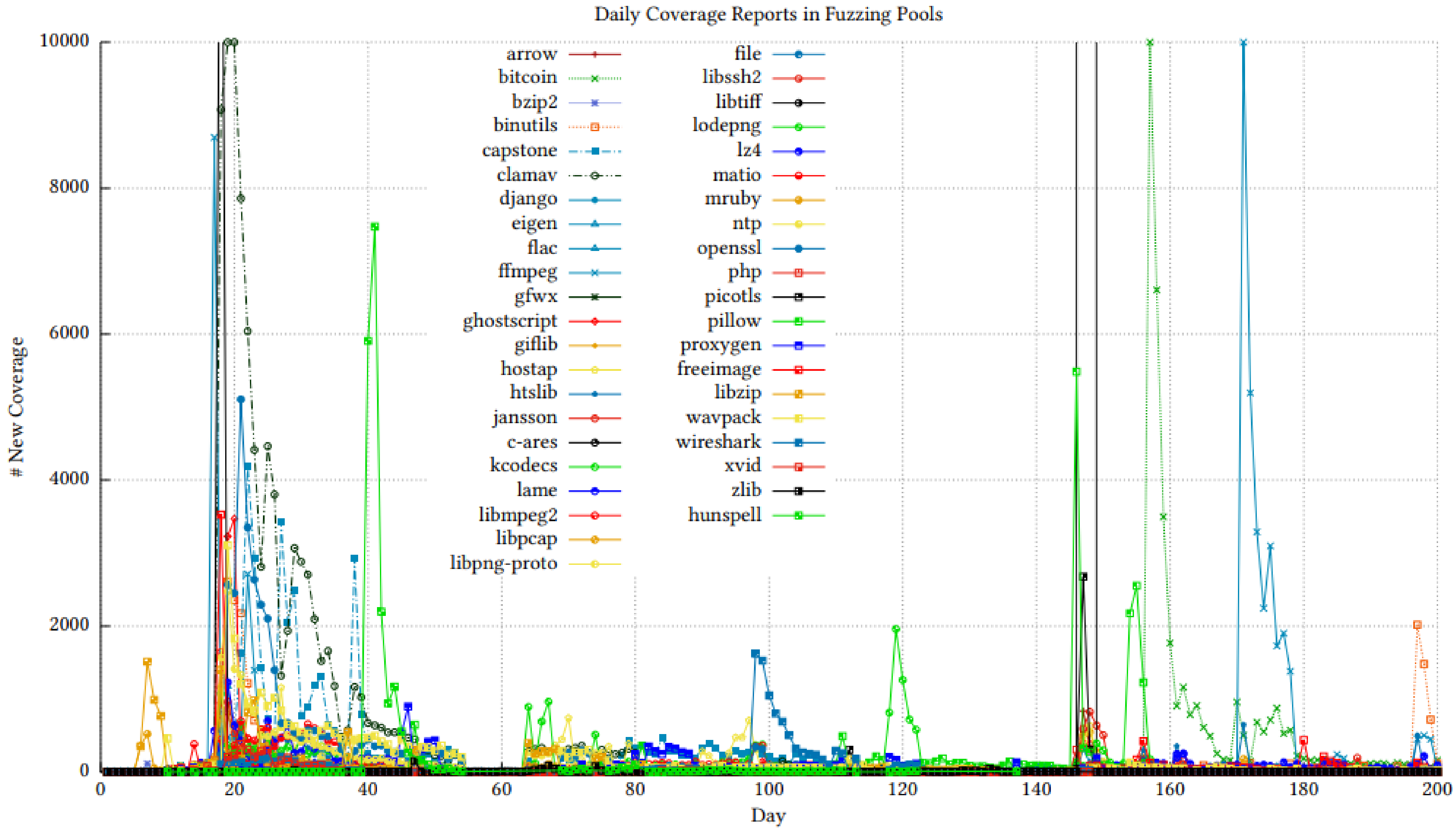
Evaluation – Cheat Prevention (simulation)

Solution: make system more beneficial to honest users!



Deployment & Evaluation

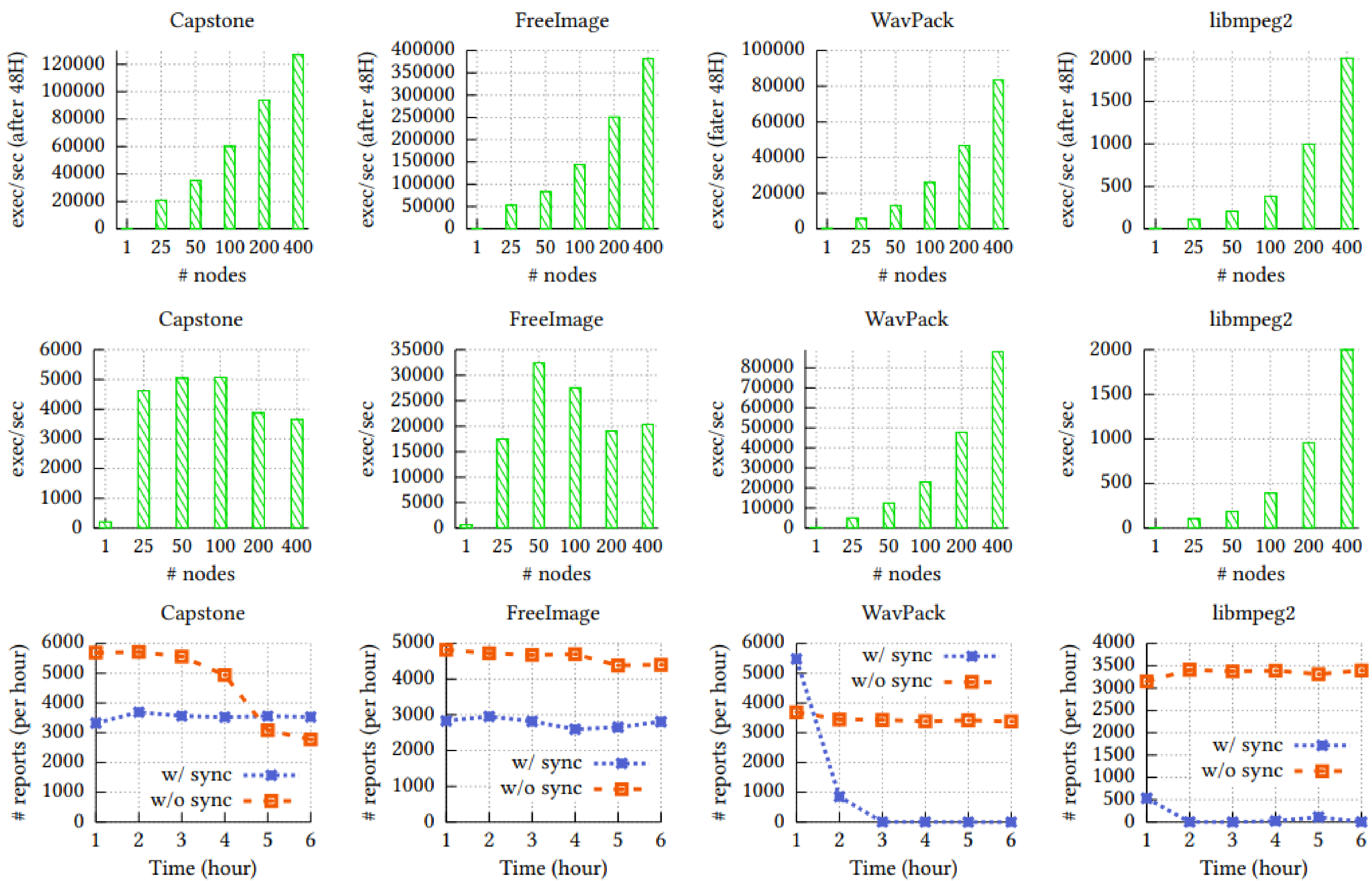
Test Deployment (7~800 beta testers)



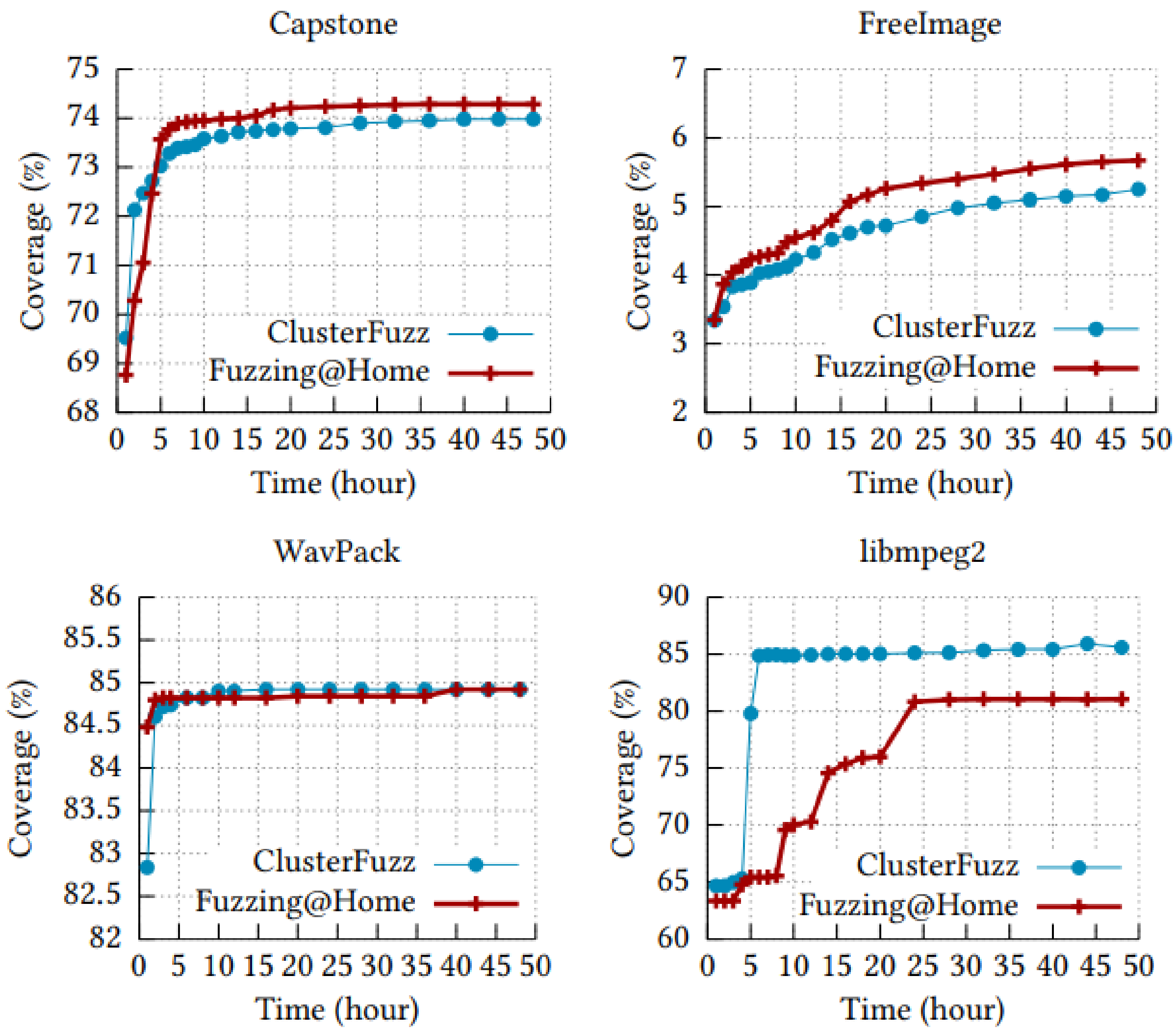
Evaluation Environment

- **Distributed Servers up to #1,000 cores**
  - Large-Scale pool evaluation
    - ✓ Coverage Saturation
    - ✓ State Synching
    - ✓ Other performances…
- **ClusterFuzz**
  - comparison evaluation
  - Used 100 cores

Evaluation - Scalability

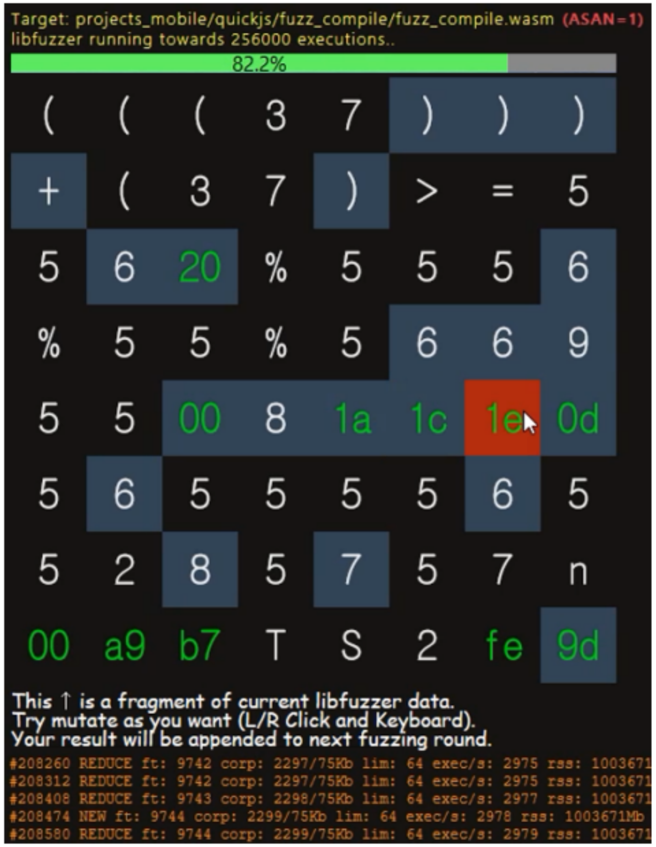


Evaluation – ClusterFuzz Comparison



WASM Fuzzer Running Example

http://fuzzcoin.gtisc.gatech.edu:8000/



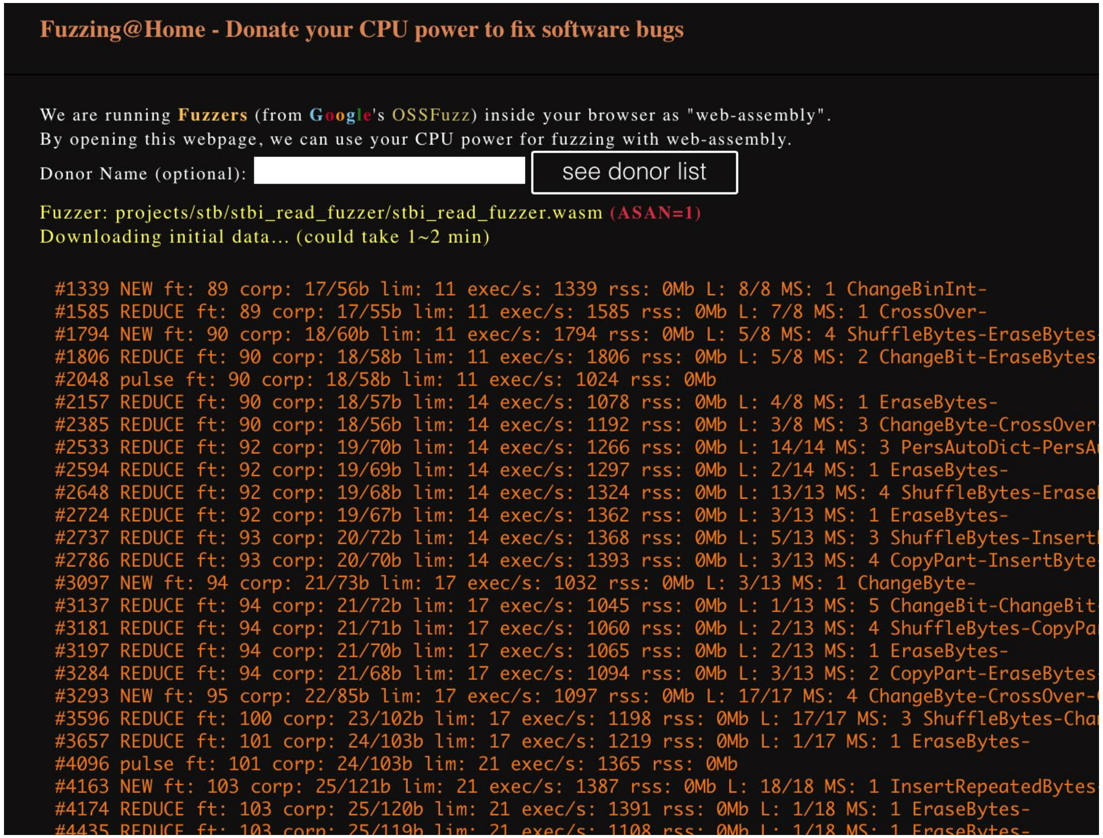


Figure 12. WASM-fuzzer running inside Chrome. The WASM-fuzzer randomly picked one test case and displayed it as a hex-dump. Black tiles are unchanged bytes, and grey tiles are mutated ones by the user.

Discovered Bugs (as in ClusterFuzz)

Project	# Unique Bugs	Description
Apache Arrow	1	null pointer dereference
ClamAV	2	heap-read-buffer-overflow null pointer dereference
FreeImage	5	stack-write-buffer-overflow out-of-memory allocation-size-too-big heap-write-buffer-overflow global-read-buffer-overflow
Capstone	1	global-read-buffer-overflow
htslib	1	out-of-memory
libtiff	1	out-of-memory
matio	21	calloc-overflow allocation-size-too-big out-of-memory SEGV on unknown address (9) stack-write-buffer-overflow heap-read-buffer-overflow (5) heap-write-buffer-overflow memcpy-param-overlap floating point exception
Samba	1	heap-read-bufferoverflow
Xvid	1	heap-read-bufferoverflow
mruby	1	out-of-memory
stb	1	heap-read-buffer-overflow
quickjs	1	heap-read-buffer-overflow
Total	37	unique bugs found

Other Issues (see paper)

- **Discovery Stashing Problem**
  - Collaborator selectively not reporting findings
- **Performance Optimization**
  - How to optimize work verification loads?
- **Implementation Details**
  - How to integrate fuzzer for Fuzzing@Home?
- **WASM-based fuzzer**
  - What are the benefits/limitations?

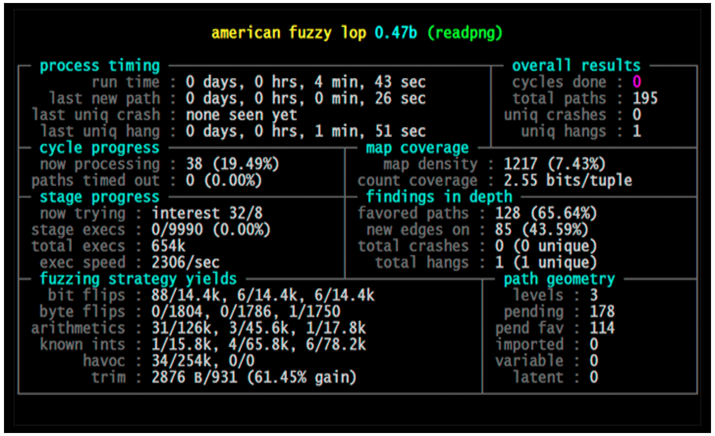
Future Work/Ideas..

❖Utilize Proof-of-Fuzzing-Work for block-chain?

- As in bitcoin PoW which is a lot of electricity waste

❖Fuzzing + Bitcoin?

- Bitcoin miners find hash collision
- Fuzzcoin miners find errors



+

❖Utilize fuzzing to quantify bug-bounty?

- Difficult to find crash -> more rewards for bug-bounty?

Thank you