THE UNIVERSITY OF MELBOURNE

Expand the reach of Fuzzing

Fuzzing & Software Security Summer School @NUS 2024

Thuan Pham

ARC DECRA Fellow & Senior Lecturer in Cyber Security

About me

- Senior Lecturer (U.S equiv. Associate Professor) in Cybersecurity at University of Melbourne
- A fuzzing enthusiast
- Co-author of open-sourced tools including AFLFast, AFLGo, AFLSmart, AFLNet, AFLTeam, EDEFuzz, and ProFuzzBench
- Founder & Lead of the **Melbourne Fuzzing Hub**

The ***<u>Fuzzed</u>*** Outline

- Introduction to Fuzzing

<u>Tutorial 1:</u> Beyond Well-tested Applications

- ➢ Fuzzing stateful network protocol implementations
- ➢ Fuzzing graph algorithm implementations

<u>Tutorial 2:</u> Beyond Crash Oracles

- ➢ Introduction to diff. & metamorphic fuzzing
- ➢ Fuzzing Web APIs for excessive data exposures

\+ Discussion: Beyond the Coverage Plateau

Acknowledgements













Acknowledgements

Australian Government

Australian Research Council

Google

aws

THE UNIVERSITY OF MELBOURNE

Beyond Well-tested Applications

AFLNET: A Greybox Fuzzer for Network Protocols

Van-Thuan Pham
Monash University
thuan.pham@monash.edu

Marcel Böhme
Monash University
marcel.boehme@monash.edu

Abhik Roychoudhury
National University of Singapore
abhik@comp.nus.edu.sg

***Abstract*—Server fuzzing is difficult. Unlike simple command-line tools, servers feature a massive state space that can be traversed effectively only with well-defined sequences of input messages. Valid sequences are specified in a *protocol*. In this paper, we present AFLNET, the first greybox fuzzer for protocol implementations. Unlike existing protocol fuzzers, AFLNET takes a mutational approach and uses state-feedback to guide the fuzzing process. AFLNET is seeded with a corpus of recorded message exchanges between the server and an actual client. No protocol specification or message grammars are required. AFLNET acts as a client and replays variations of the original sequence of messages sent to the server and retains those variations that were effective at increasing the coverage of the code or state space. To identify the server states that are exercised by a message sequence, AFLNET uses the server's response codes. From this feedback, AFLNET identifies progressive regions in the state space, and systematically steers towards such regions. The case studies with AFLNET on two popular protocol implementations demonstrate a substantial performance boost over the state-of-the-art. AFLNET discovered two new CVEs which are classified as critical (CVSS score CRITICAL 9.8).**

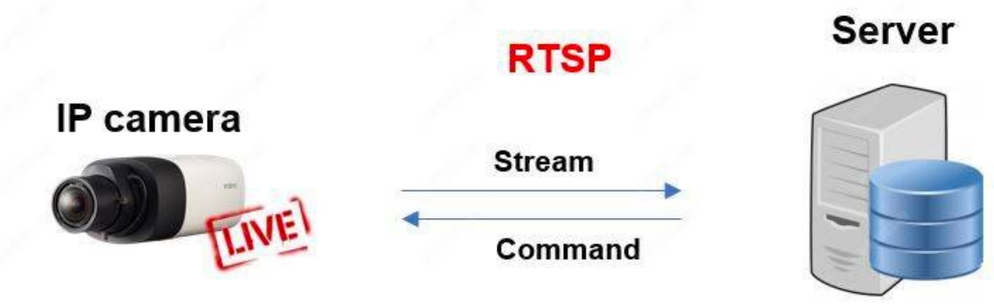
> *"One of the things that I struggle with is the limitation AFL seems to have, in that it only performs fuzzing with one input (a file). For many systems such as network protocols, it would be useful if fuzzing could be done on a sequence of inputs. This sequence of inputs might be for example messages necessary to complete a handshake in TLS/TCP."*
>
> — Paul (a member of the AFL's user group) [8]

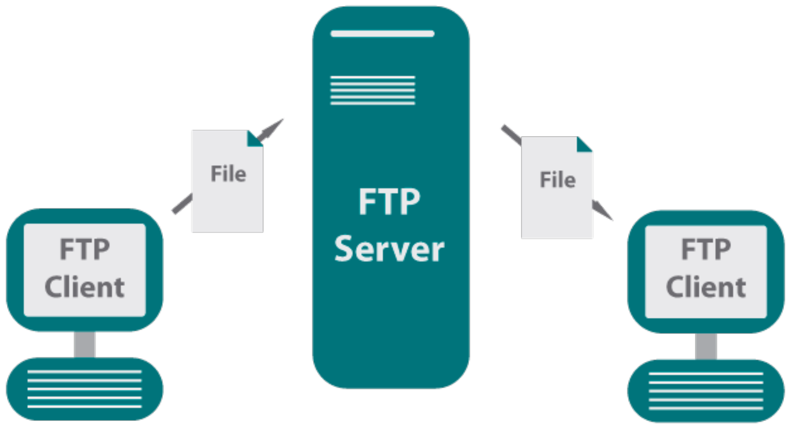
> *"I'm interested in doing something fairly non-traditional and definitely not currently supported by AFL. I would like to perform fuzzing of a large and complex external server that cannot easily be stripped down into small test cases."*
>
> — Tim Newsham (a member of the AFL's user group) [8]

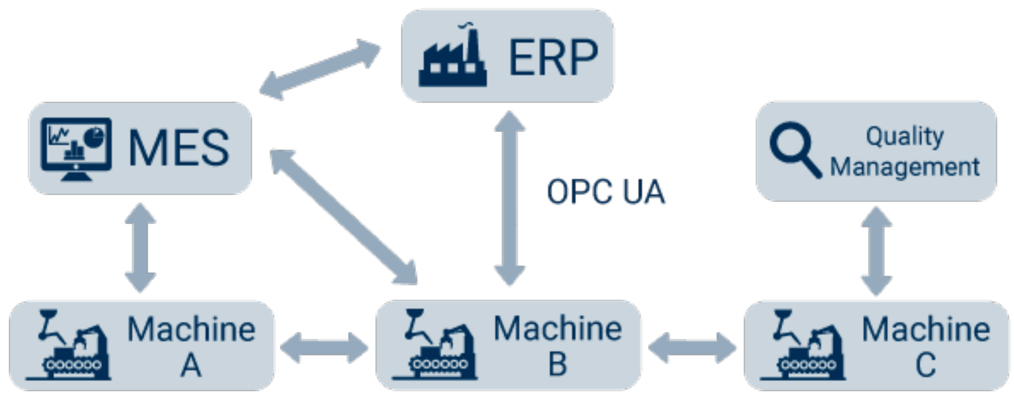
Fig. 1. Requests from AFL's users asking for stateful fuzzing support

Greybox-Fuzzing for Stateful Network Protocols

Why should we care about network protocols?







Fuzzing stateful protocol implementations is challenging

- Server accepts *sequences of messages*
- Server behaviour depends on the *current input & current program state*
  - *message order matters*
  - *Knowing current server state & how to drive the search towards a specific state is important*

```
220 FTP Server ready
USER foo
331 User foo OK. Password required
PASS foo
230 User logged in, proceed.
MKD demo
257 Directory created.
CWD demo
250 Requested file action okay, completed.
STOR test.txt
150 File status okay
226 Transfer complete
QUIT
221 Goodbye!
```

A sample FTP session to upload a file (test.txt) to a new folder (demo) on the server

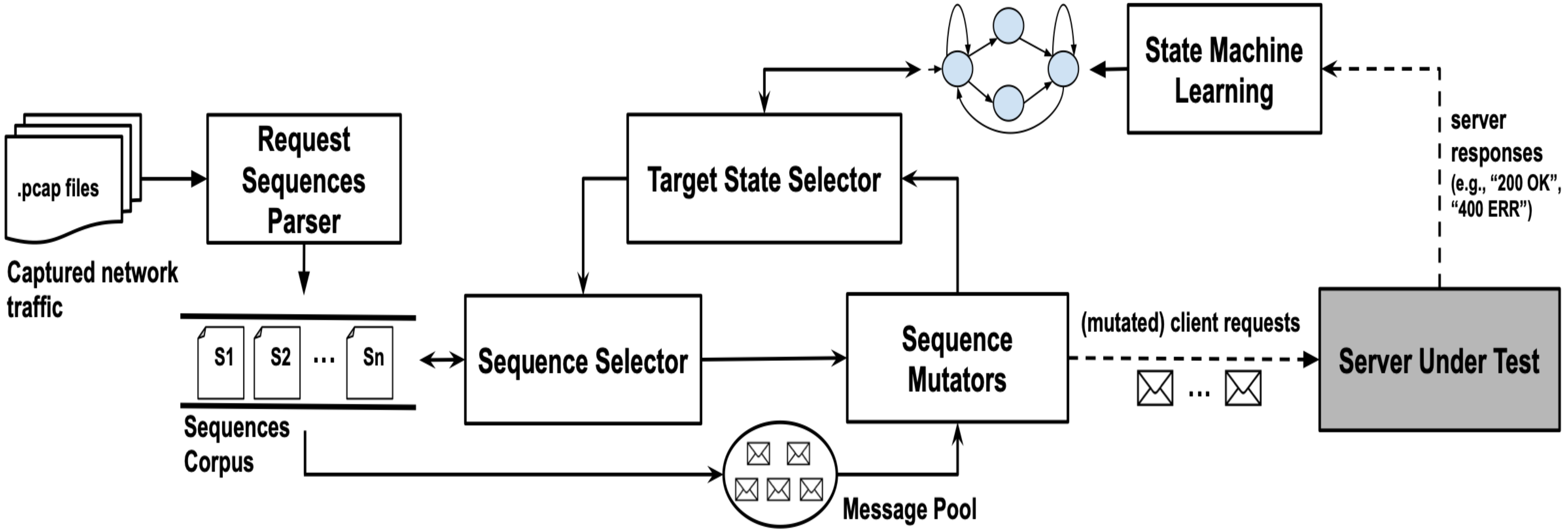
Existing widely-used approaches

1) Stateful Black-box Fuzzing, e.g., Peach, BooFuzz
   - Require manual constructed state machine/model
   - With no/limited feedback
2) Stateless Grey-box Fuzzing: AFL

> *"One of the things that I struggle with is the limitation AFL seems to have, in that it only performs fuzzing with one input (a file). For many systems such as network protocols, it would be useful if fuzzing could be done on a sequence of inputs. This sequence of inputs might be for example messages necessary to complete a handshake in TLS/TCP."*
>
> — Paul (a member of the AFL's user group) [8]

> *"I'm interested in doing something fairly non-traditional and definitely not currently supported by AFL. I would like to perform fuzzing of a large and complex external server that cannot easily be stripped down into small test cases."*
>
> — Tim Newsham (a member of the AFL's user group) [8]

Architecture of AFLNet

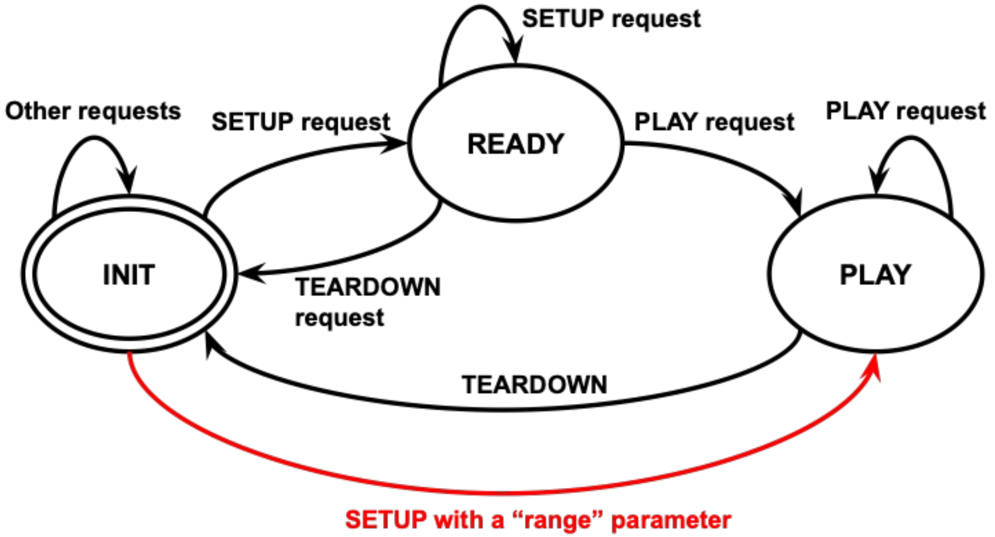


AFLNet takes a ***mutational approach*** and uses ***state-feedback*** to *dynamically* construct ***protocol state machine*** and guide the fuzzing process, together with code coverage feedback.

- AFLNet acts as a client.
- AFLNet is seeded with a corpus of recorded message exchanges.
- AFLNet monitors server behaviours (e.g., code coverage) and its responses

Automated State Machine Inference

- Time consuming
- Require domain knowledge
- Implemented protocol could be different from the standard specification



Manual & static approach

- Not time consuming
- Capture the exact implemented protocol

Automatic & dynamic approach

220 FTP Server ready

USER foo

331 User foo OK. Password required

PASS foo

230 User logged in, proceed.

MKD demo

257 Directory created.

CWD demo

250 Requested file action okay, completed.

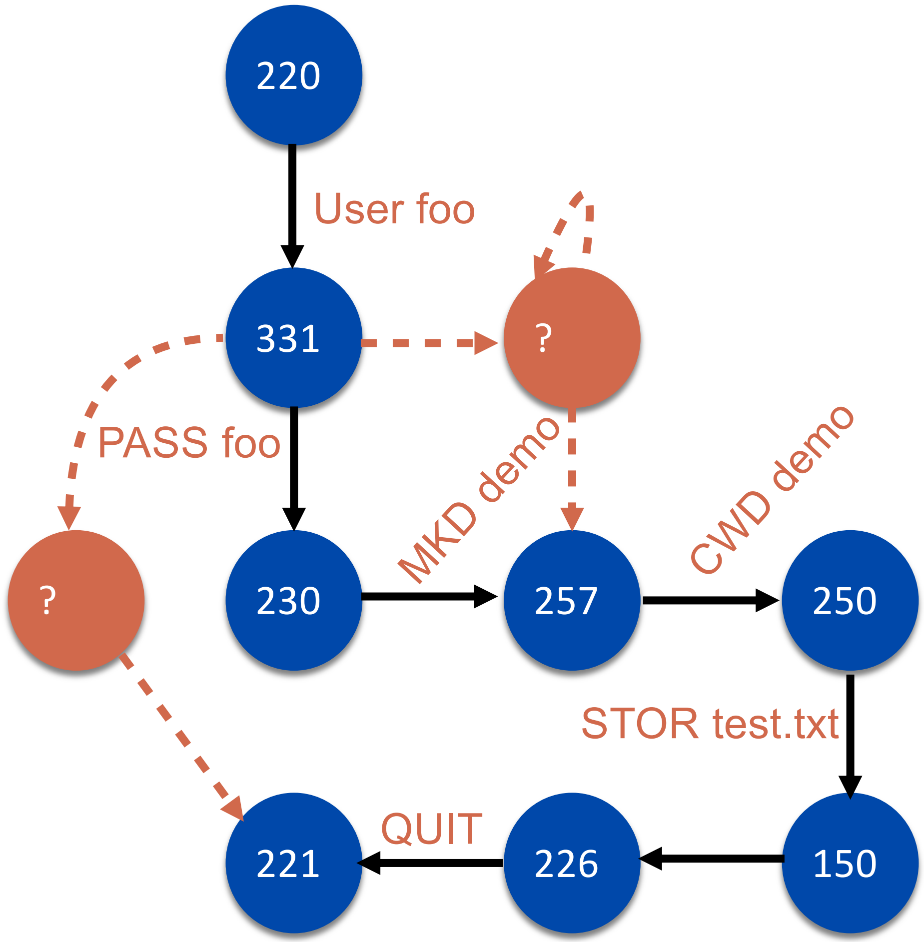
STOR test.txt

150 File status okay

226 Transfer complete

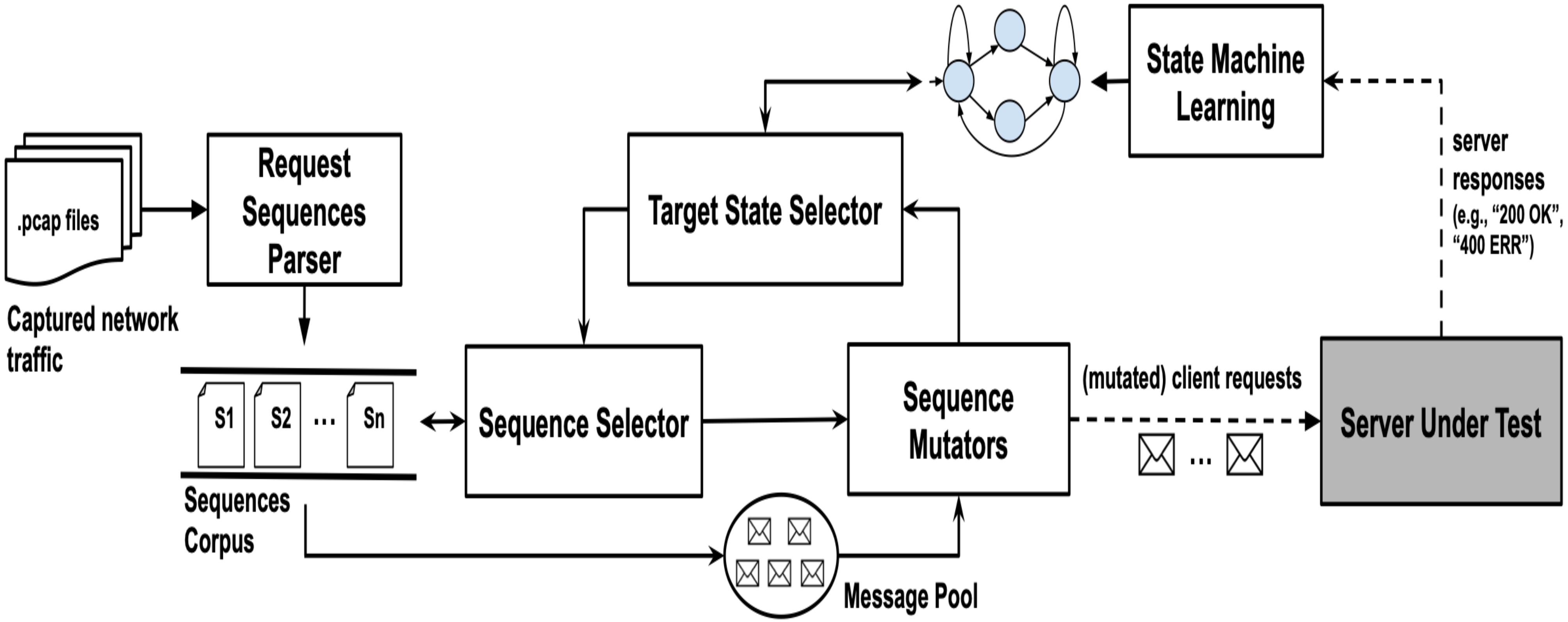
QUIT

221 Goodbye!



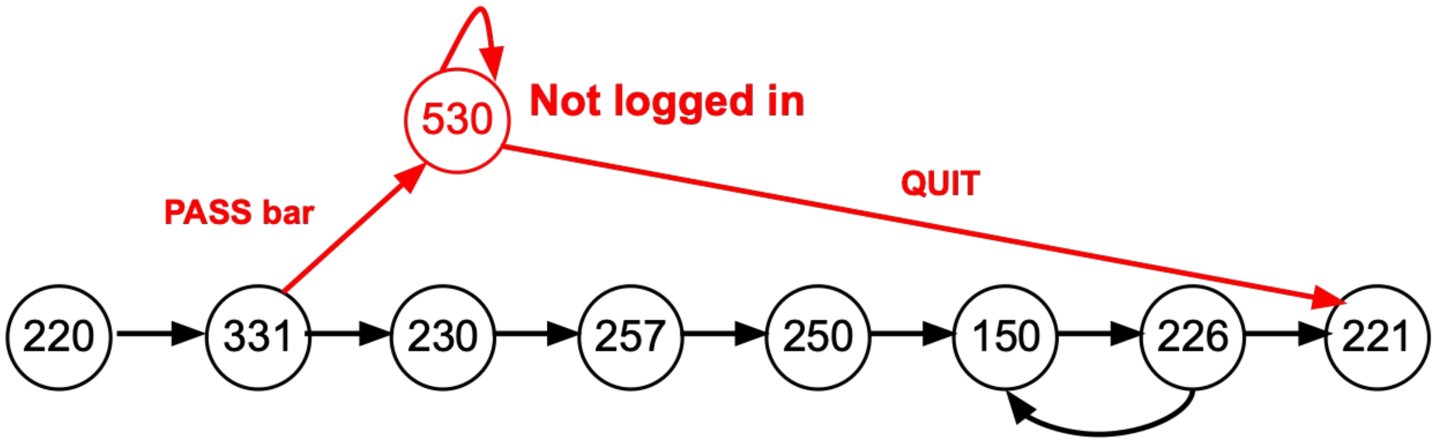
AFLNet workflow

Prioritise *“progressive” states* that contribute more towards increased code coverage.



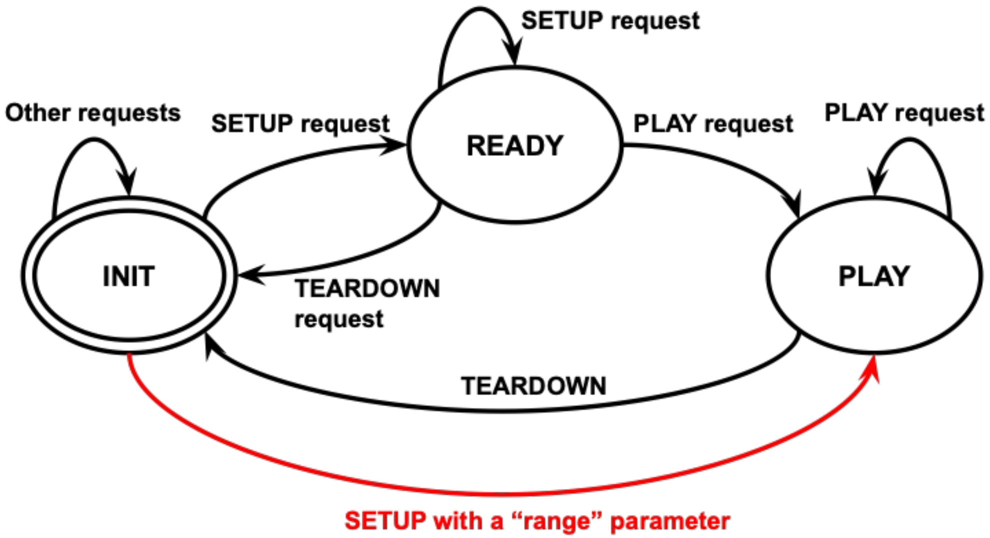
Original message sequence (i.e., seed input)

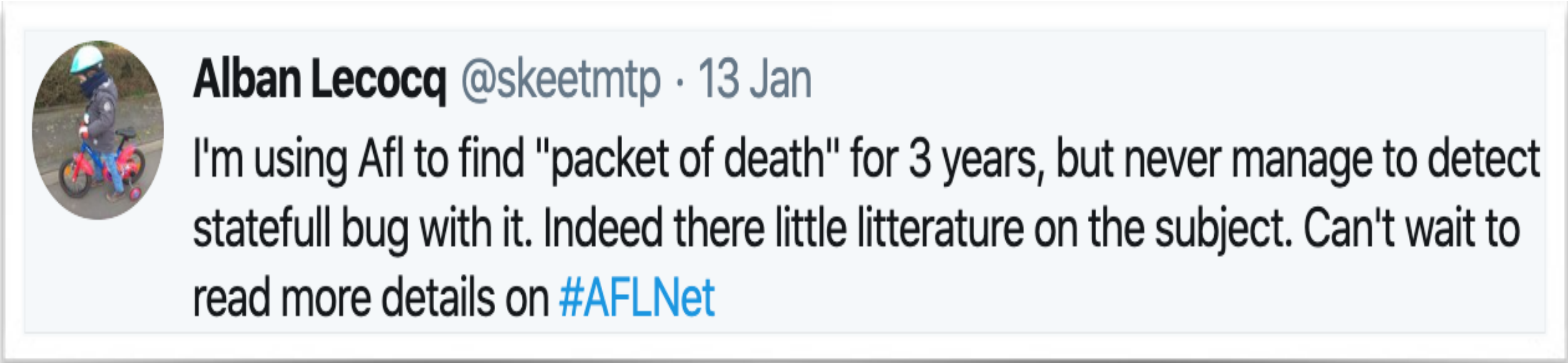
And then, state 331 (User OK) is targeted



inferred state machine

CVE-2019-7314 - liblivemedia in Live555 before 2019.02.03 mishandles the termination of an RTSP stream after RTP/RTCP-over-RTSP has been set up, which could lead to a Use-After-Free error that causes the RTSP server to crash (Segmentation fault) or possibly have unspecified other impact.





AFLNet keeps evolving
https://github.com/aflnet/aflnet

Mar 2020

Released on GitHub

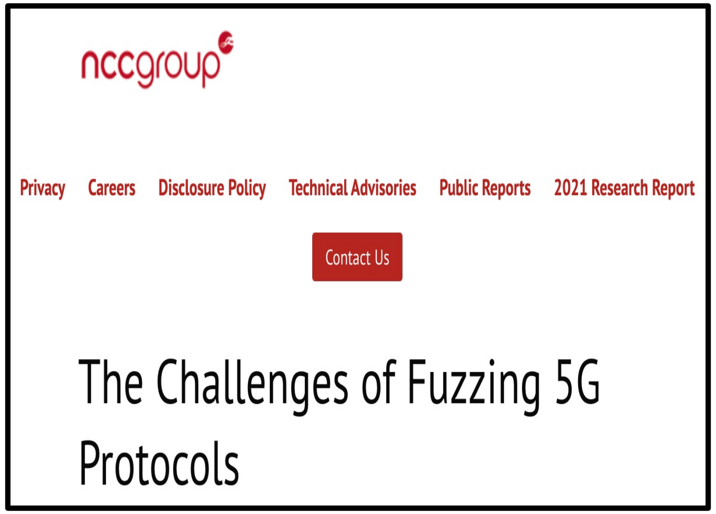
May 2024

200+ citations, 800+ stars, 180+ forks

FTP, RTSP

+ SSH, TLS, SMTP,

DTLS, DICOM, DNS, SIP, MODBUS …



ProFuzzBench: A Benchmark for Stateful Protocol Fuzzing

https://github.com/profuzzbench/profuzzbench

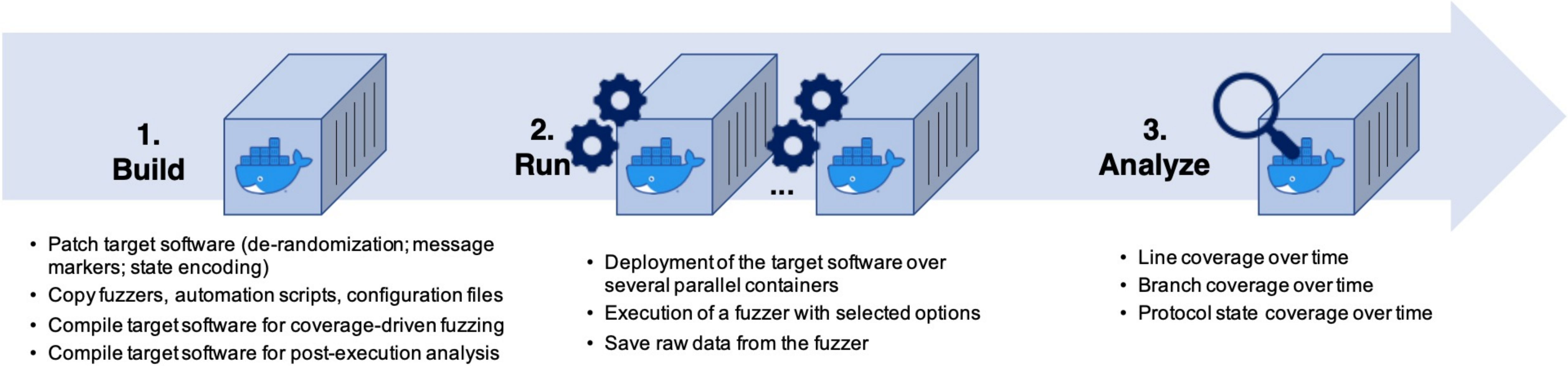


Figure 1: Workflow of benchmark automation.

Activity: Code Understanding & Discussions

- How to support a new protocol?
- What are the limitations of AFLNet and potential solutions?

Reading recommendation

Nyx-Net: Network Fuzzing with Incremental Snapshots

Sc … z[1]

Large Language Model guided Protocol Fuzzing

Ruijie Meng*, Martin Mirchev*, M

*National Un

FitM: Binary-Only Coverage-Guided Fuzzing for Stateful Network Protocols

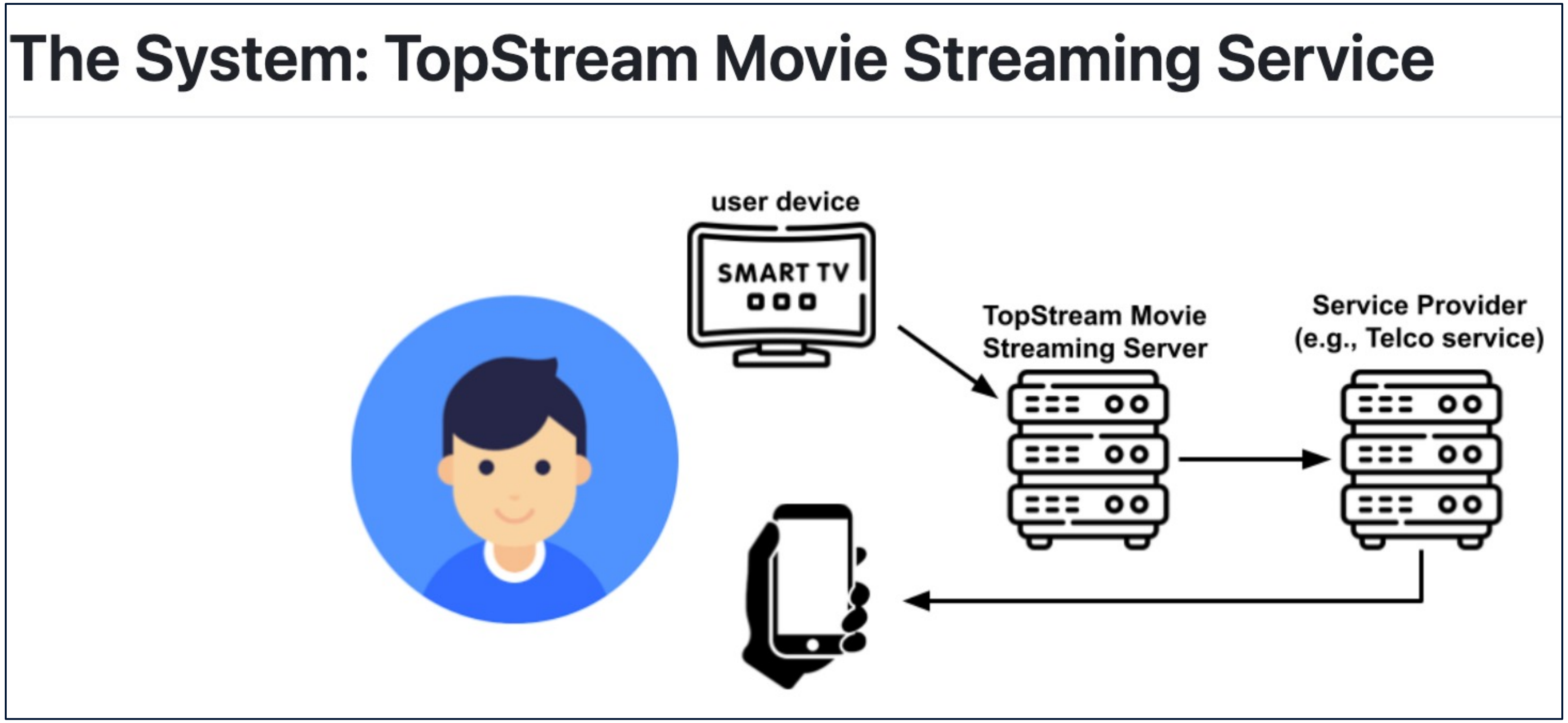
Julian Beier
TU Berlin
j.beier@tu-berlin.de

Marc Munier
TU Berlin
m.munier@campus.tu-berlin.de

NSFuzz: Towards Efficient and State-Aware Network Service Fuzzing

SHISONG QIN, Tsinghua University, China
FAN HU, State Key Laboratory of Mathematical Engineering and Advanced Computing, China
ZHEYU MA, BODONG ZHAO, TINGTING YIN, and CHAO ZHANG, Tsinghua University, China

Have fun with AFLNet ☺



SCAN ME

https://github.com/
SWEN90006-2023/
SWEN90006-assignment-2

GraphFuzz: Automated Testing of Graph Algorithm Implementations with Differential Fuzzing and Lightweight Feedback

Wenqi Yan, Manuel Rigger, Tony Wirth, Van-Thuan Pham

***Abstract*—Graph algorithms, such as shortest path finding, play a crucial role in enabling essential applications and services like infrastructure planning and navigation, making their correctness important. However, thoroughly testing graph algorithm implementations poses several challenges, including their vast input space (i.e., arbitrary graphs). Moreover, through our preliminary study, we find that just a few automatically generated graphs (less than 10) could be enough to cover the code of many graph algorithm implementations, rendering the code coverage-guided fuzzing approach—one of the state-of-the-art search algorithms—less efficient than expected.**

To tackle these challenges, we introduce *GraphFuzz*, the first automated feedback-guided fuzzing framework for graph algorithm implementations. Our key innovation lies in identifying lightweight and algorithm-specific feedback signals to combine with or completely replace the code coverage feedback to enhance the diversity of the test corpus, thereby speeding up the bug-finding process. This novel idea also allows *GraphFuzz* to effectively work in both black-box (i.e., no code coverage instrumentation/collection is required) and grey-box setups. *GraphFuzz* applies differential testing to detect both crash-triggering bugs and logic bugs. Our evaluation demonstrates the effectiveness of *GraphFuzz*. The tool has successfully discovered 12 previously unknown bugs, including 6 logic bugs, in 9 graph algorithm implementations in two popular graph libraries, NetworkX and iGraph. All of them have been confirmed and and 11 bugs have been rectified by the libraries' maintainers.

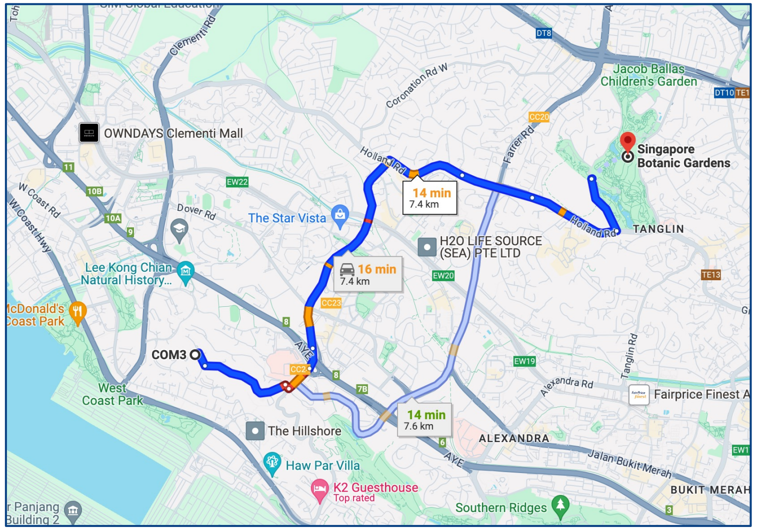
> *"This morning I was talking to a guy in Microsoft Research who is pretty famous. He described a neat project he worked on that essentially boiled down to a very complex graph shortest-path problem. That got my mind churning on the general problem of graph algorithms and testing graph algorithm implementations.[...] I hadn't looked at a graph problem in a while so I decided to code up the more-or-less canonical example: Dijkstra's algorithm [...]. Even with a tiny graph it's easy for a human to make a mistake. So, what about testing graph algorithms? In some ways this is a classic problem: a large number of inputs, possibility of bad arguments, many assumptions, and so on."*
>
> – J. D. McCaffrey (Research Software Engineer at Microsoft) [2]

Fig. 1: **A researcher's view on the challenges of testing graph algorithm implementations.**

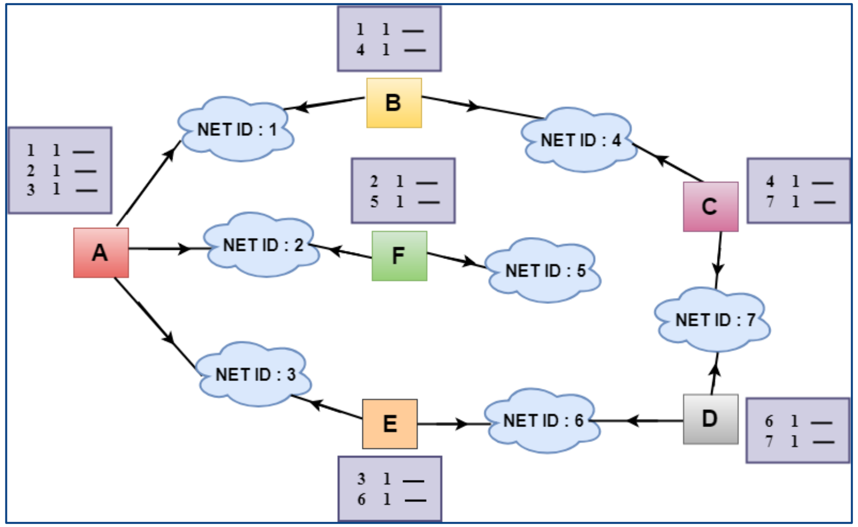
weights that are non-negative, and an arbitrary source vertex, s, as its inputs. The algorithm is supposed to find shortest paths from s to all other vertices on G. With G having M edges and N vertices, this renders the input space exceptionally vast to thoroughly test an implementation of the algorithm, surpassing the capacity of human testers. In particular, we would need to select a random graph G and a random vertex

Graph algorithms & their applications

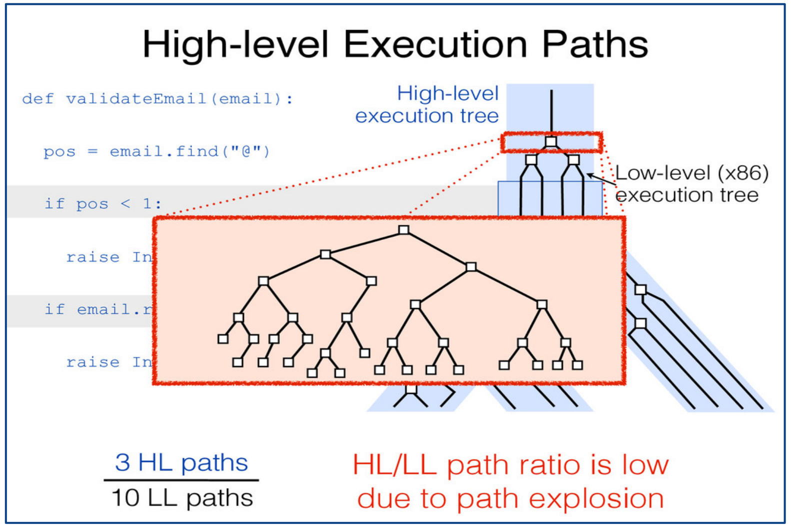
https://memgraph.com/blog/graph-algorithms-applications



Google Maps



Network Routing



Symbolic Execution

How would you test graph algorithms impl. And what could be the challenges?

C1. Vast Input Space

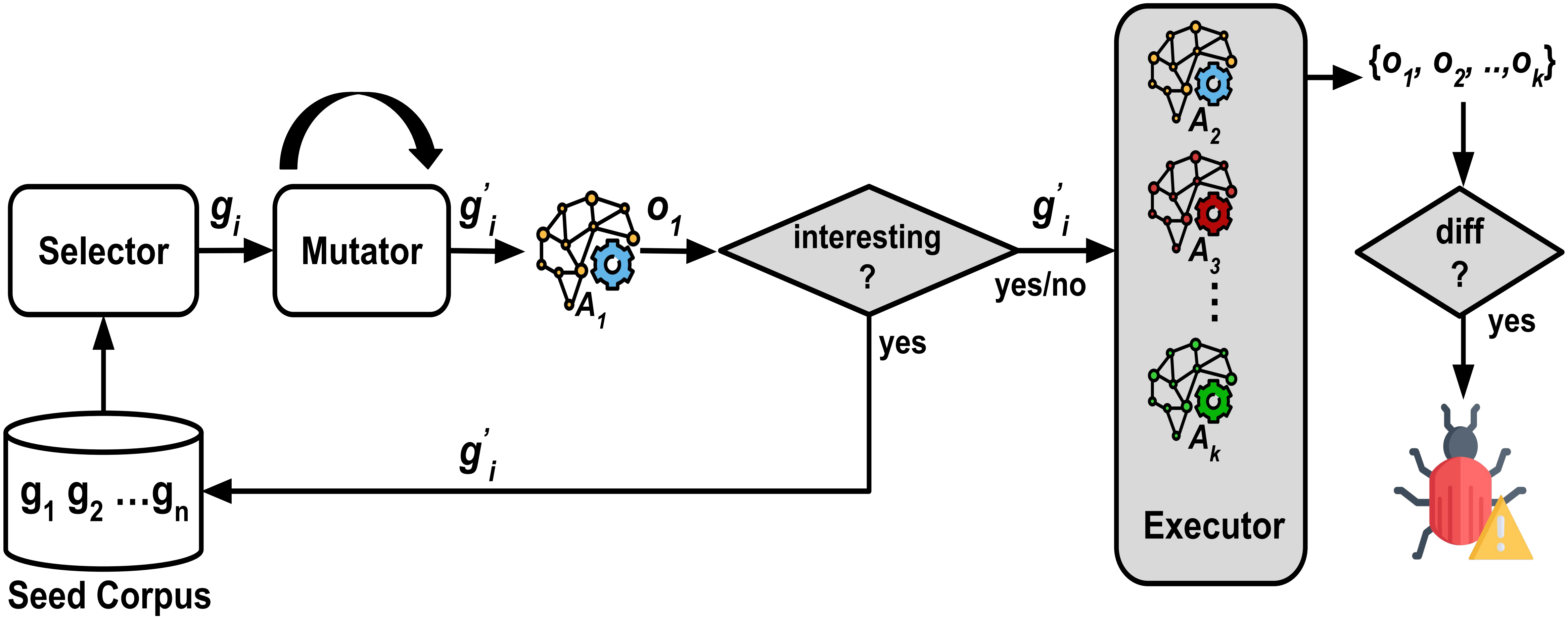
C2. Test Oracle Problem

"Given an input for a system, the challenge of distinguishing the corresponding desired, correct behaviour from potentially incorrect behavior is called the test oracle problem" [Earl et al. 2015]

> *"This morning I was talking to a guy in Microsoft Research who is pretty famous. He described a neat project he worked on that essentially boiled down to a very complex graph shortest-path problem. That got my mind churning on the general problem of graph algorithms and testing graph algorithm implementations.[...] I hadn't looked at a graph problem in a while so I decided to code up the more-or-less canonical example: Dijkstra's algorithm [...]. Even with a tiny graph it's easy for a human to make a mistake. So, what about testing graph algorithms? In some ways this is a classic problem: a large number of inputs, possibility of bad arguments, many assumptions, and so on."*
>
> – J. D. McCaffrey (Research Software Engineer at Microsoft) [2]

https://jamesmccaffrey.wordpress.com/2010/01/12/testing-graph-algorithms/

GraphFuzz: Code Coverage-Guided Differential Fuzzing



Algorithms curated for accessing GraphFuzz

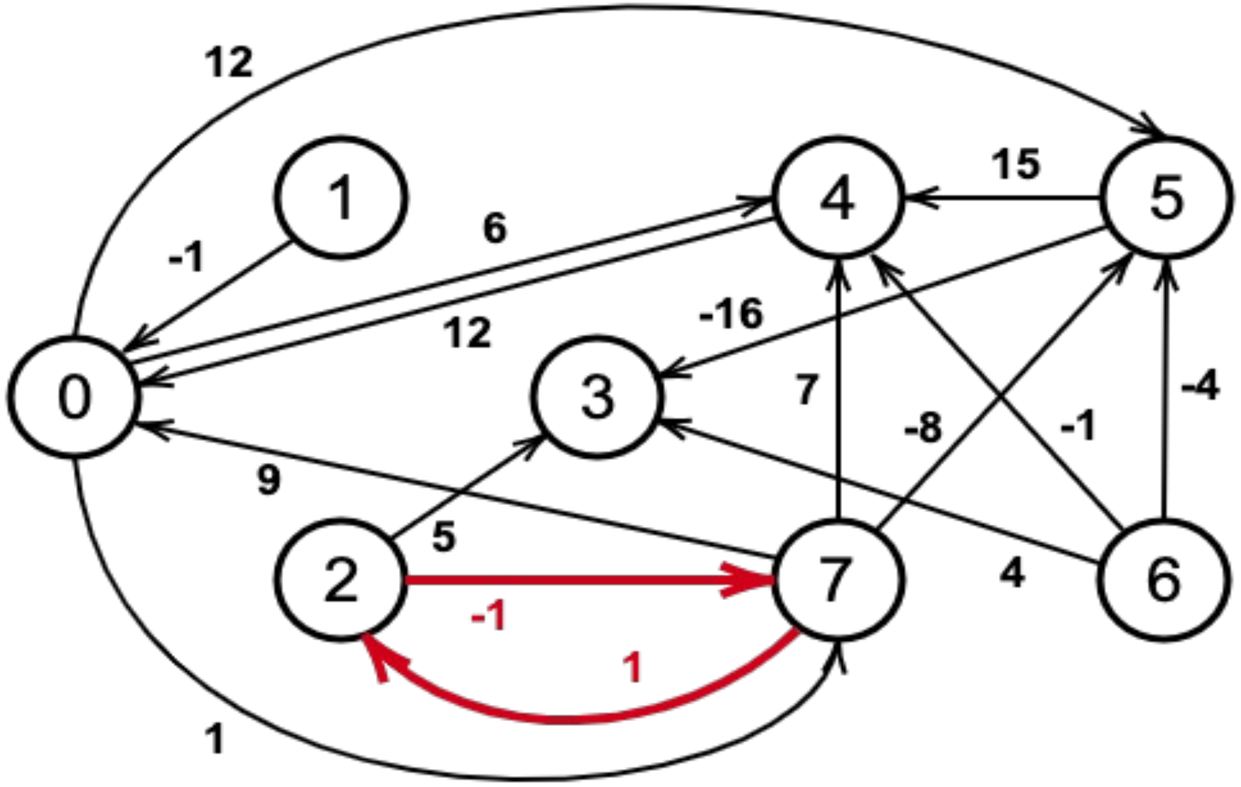
- 2 popular graph libraries
- 7 categories of graph algorithms
- 9 graph problems
- 21 graph algorithm implementations

Category	Problem	Algorithm	Implementation	
			NX	IG
Path Finding and Search	Shortest Path Finding	Bellman Ford [24]	✓	✓
		Goldberg Radzik [25]	✓	
		Dijkstra [3]	✓	
	Minimum Spanning Tree	Prim [41]	✓	✓
		Kruskal [42]	✓	
		Boruvka [43]	✓	
Community Detection	Strongly Connected Component	Tarjan [44]	✓	✓
		Tarjan (recurisve version)	✓	
		Kosaraju [45]	✓	
	Bi-Connected Component	[46]	✓	✓
Centrality and Importance	Harmonic Centrality	[47]	✓	✓
Similarity	Jaccard Similarity	[48]	✓	✓
	Max Matching	Hopcroft Karp [49]	✓	
		Eppstein	✓	
		Push-Relabel [50]		✓
Heuristic Link Prediction	Adamic Adar	[48]	✓	✓
Others	Max Flow Value	Goldberg Tarjan [51]	✓	✓
		Dinitz [52]	✓	
		Boykov Komogorov [53]	✓	

Preliminary results

<u>**6 logic bugs**</u> found in implementations of

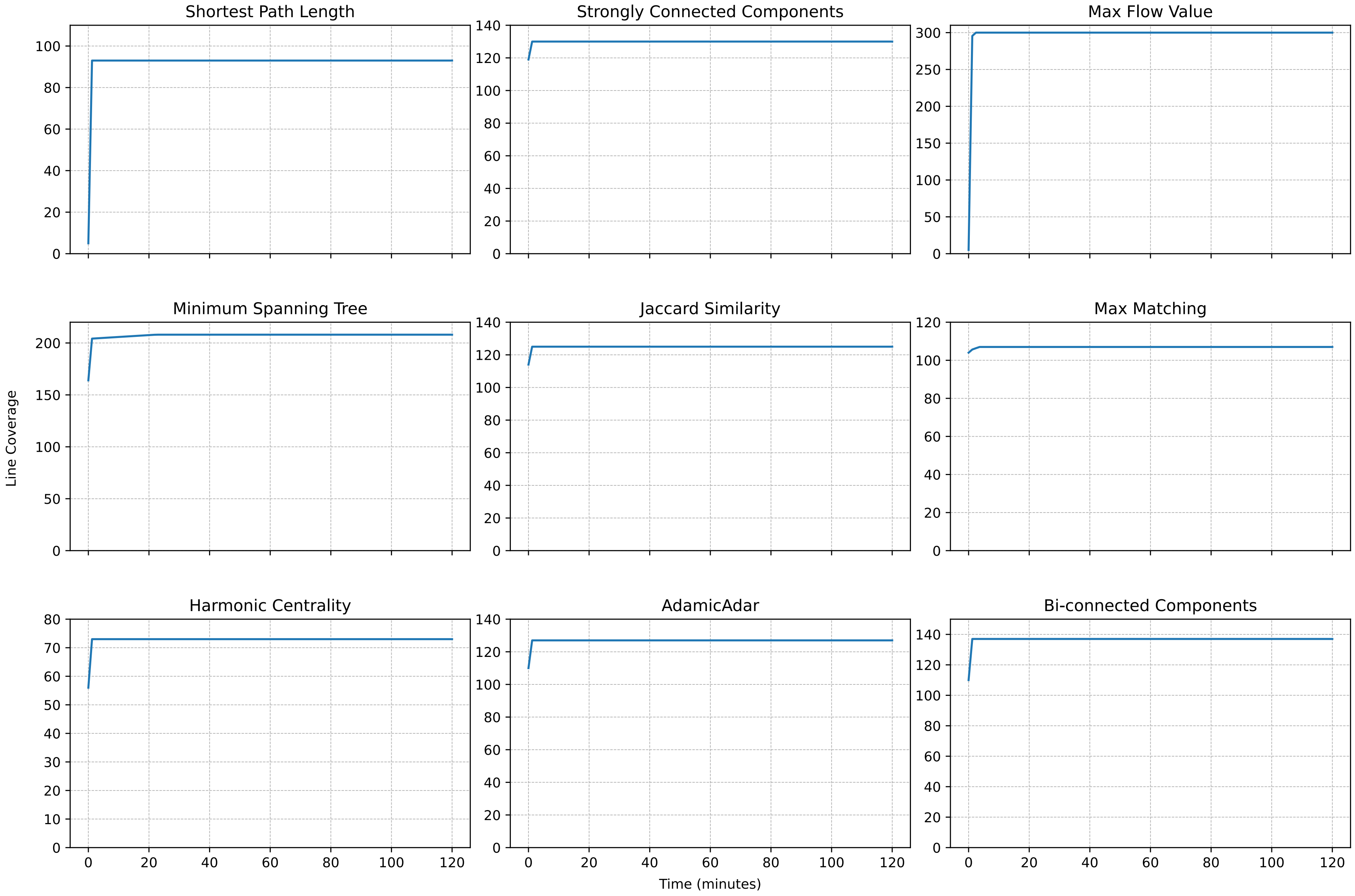
1) Goldberg-Radzik path finding
2) Tarjan strongly connected components
3) Jaccard similarty
4) Max flow algorithm
5) Adamic-Adar link prediction



"This is puzzling – (so perhaps a bug). [...] Manually checking these shows no negative cycles, but the [2,7] cycle has weight 0. I suspect that is the trouble."

— Prof. Dan Schult (The creator of NETWORKX)

However ...


Shortest Path Length
0 20 40 60 80 100
Strongly Connected Components
0 20 40 60 80 100 120 140
Max Flow Value
0 50 100 150 200 250 300
Minimum Spanning Tree
0 50 100 150 200
Jaccard Similarity
0 20 40 60 80 100 120 140
Max Matching
0 20 40 60 80 100 120
Harmonic Centrality
0 10 20 30 40 50 60 70 80
AdamicAdar
0 20 40 60 80 100 120 140
Bi-connected Components
0 20 40 60 80 100 120 140
Line Coverage
0 20 40 60 80 100 120
Time (minutes)

Observation & Insights

C3. Quickly saturated code coverage with small corpus size

We need other feedback signals

C4. Multi-language codebase or binary-only libraries

We may go back to black-box fuzzing, but *<u>can we do better</u>* than that?

Criteria to design new feedback signals

- Be able to support feedback-guided fuzzing of ***multi-language cod***ebases or ***binary-only libraries***
- Be ***lightweight*** => negligible overhead
- Be ***distinguishable***
- Be able to produce ***manageable seed corpus***

Algorithm-specific signals

Graph problem	Full Output	Feedback	Explanation
STP	The length of the shortest path between two specified vertices	(l)	l: shortest path length
SCC	A list of all strongly connected components	(n, s)	n: number of components s: size of the largest component
MFV	The value of the max flow from the source to the sink vertex	(v)	v: flow value
MST	A minimum spanning tree	(w, e)	w: total weigh of the MST e: number of edges on the MST
JC	A list of scores between all pairs of vertices	(v)	v: the highest score
MM	A list of matching pairs	(n)	n: the number of matching pairs
HC	A dictionary of vertices with harmonic centrality as the value	(d)	d: the difference between the two smallest scores
AA	A list of scores between all pairs of vertices	(v)	v: the highest score
BCC	A list of all bi-connected components	(s)	s: size of the largest biconnected component

Research Questions

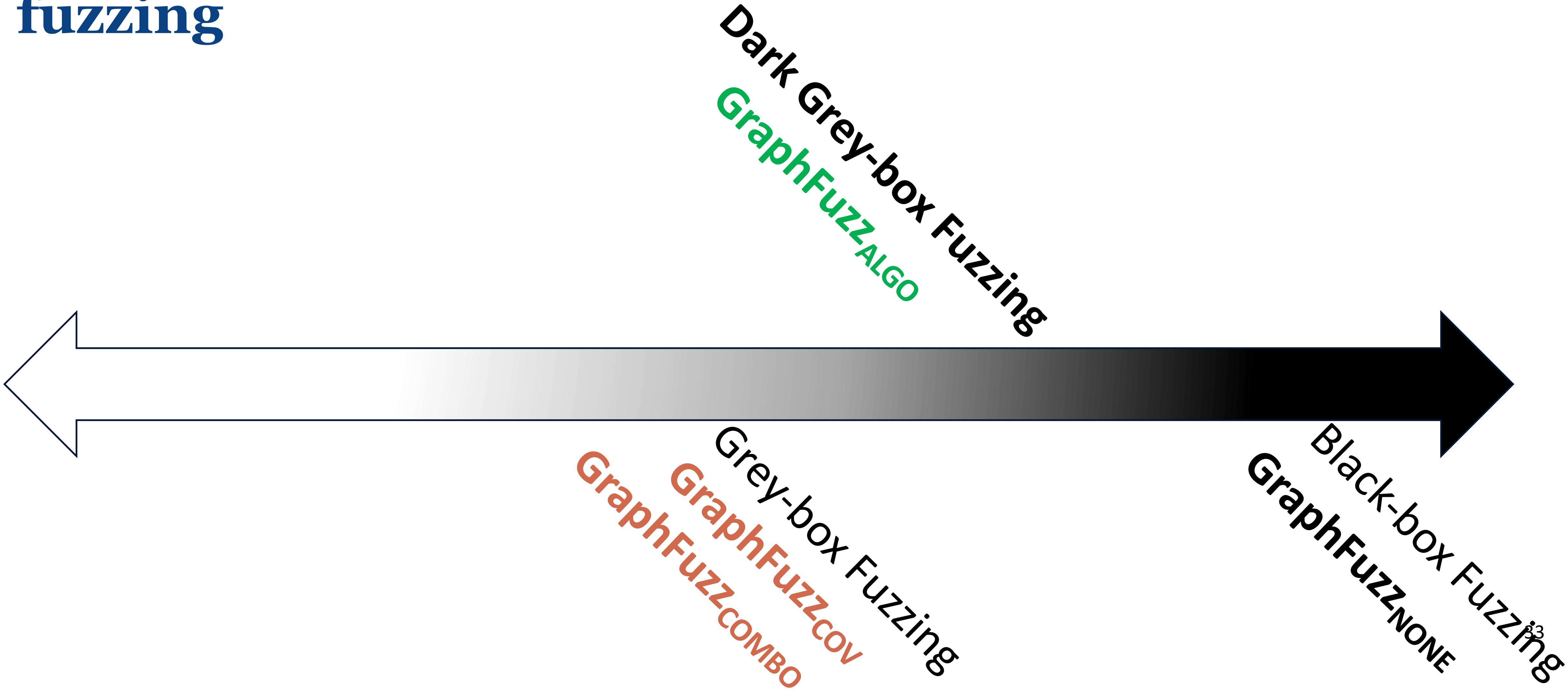
RQ-1. Does adding the ***algorithm-specific feedback signals*** help improve GraphFuzz's performance

RQ-2. How does GraphFuzz perform if ***no feedback signal*** is in use?

RQ-3. How does GraphFuzz perform in a ***larger testing campaign***?

RQ-1. Effectiveness & Efficiency of the new signals

RQ-2. Effectiveness & Efficiency of pure black-box fuzzing



Results – Bug finding

- The new feedback signals are effective
- GraphFuzz$_{ALGO}$ ranks 1st, followed by GraphFuzz$_{COMBO}$
- GraphFuzz$_{NONE}$ outperforms GraphFuzz$_{COV}$

	Small Initial Corpus			Bigger Initial Corpus		
	none	combo	algo	none	combo	algo
Bug-1	*35.96*	14.92	**39.55**	4.77	*7.79*	**20.43**
Bug-2	2.14	*21.21*	**122.91**	11.06	*23.21*	**43.86**
Bug-3	4.23	*5.50*	**6.58**	2.45	*6.19*	**13.85**
Bug-4	5.75	*17.98*	**18.94**	11.92	*13.27*	**107.51**
Bug-5	**306.69**	100.84	*167.43*	3.38	*3.69*	**3.71**
Bug-6	11.40	*15.62*	**26.61**	**35.48**	7.32	*13.72*

TABLE III: Bug discovery speed up (based on the mean values) of $GraphFuzz_{\text{NONE}}$, $GraphFuzz_{\text{COMBO}}$, and $GraphFuzz_{\text{ALGO}}$ over $GraphFuzz_{\text{COV}}$. The best result for each bug and corpus is highlighted in bold, while the second-best result is highlighted in italics.

Explanation

	Fuzzing throughput				Corpus size	
	none	cov	combo	algo	combo	algo
STPL	9153076	42064	55706	102073	1406	2033
SCC	40967765	55028	76903	170366	7792	9675
MFV	650820	25037	33736	82953	1526	1776
MST	19977653	43176	73095	182035	22787	33726
JS	2210589	50368	52263	57900	307	325
MM	43627857	201736	230785	5136005	112	145
HC	52783065	42375	79821	113156	5036	6673
AA	375026	51283	53208	57035	1775	2276
BCC	12266650	52039	79805	2061555	412	577

TABLE IV: The mean value of fuzzing throughput (i.e., number of graphs generated and evaluated over time) and corpus size of *GraphFuzz* in different setups, for each 2-hour experiment.

RQ-3. Bug finding capability of $\text{GraphFuzz}_{\text{ALGO}}$

- Longer runs: 24 hrs
- 16 more graph problems (e.g., Dice Similarity, All Node Cut, Simple Cycle)
- **Results:**
  - 18 additional bugs including 3 logic bugs and 15 crashes

$\text{GraphFuzz}_{\text{ALGO}}$ highly applicable and effective. Thus far, we have tested 26 different graph problems, and our tool has discovered bugs in 22 of them, resulting in a **success rate of 84.6%.**

Reading recommendation

Testing Database Engines via Query Plan Guidance

Jinsheng Ba
National University of Singapore

Manuel Rigger
National University of Singapore

DynSQL: Stateful Fuzzing for Database Management Systems with Complex and Valid SQL Query Generation

Zu-Ming Jiang
ETH Zurich

Jia-Ju Bai
Tsinghua University

Zhendong Su
ETH Zurich

GLeeFuzz: Fuzzing WebGL Through Error Message Guided Mutation

Hui Peng
Purdue University

Zhihao Yao
UC Irvine

Ardalan Amiri Sani
UC Irvine

Dave (Jing) Tian
Purdue University

Mathias Payer
EPFL

Beyond Crash Oracles

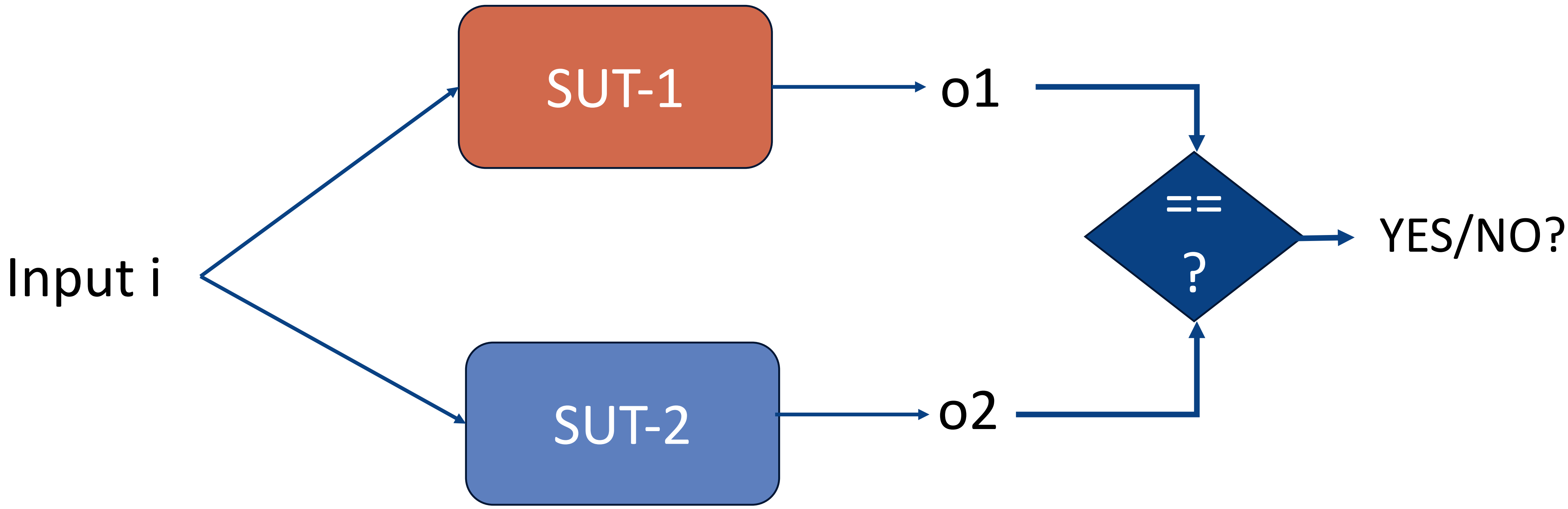
RECALL: The Test Oracle problem

"Given an input for a system, the challenge of distinguishing the corresponding desired, correct behaviour from potentially incorrect behavior is called the test oracle problem" **[Earl et al. 2015]**

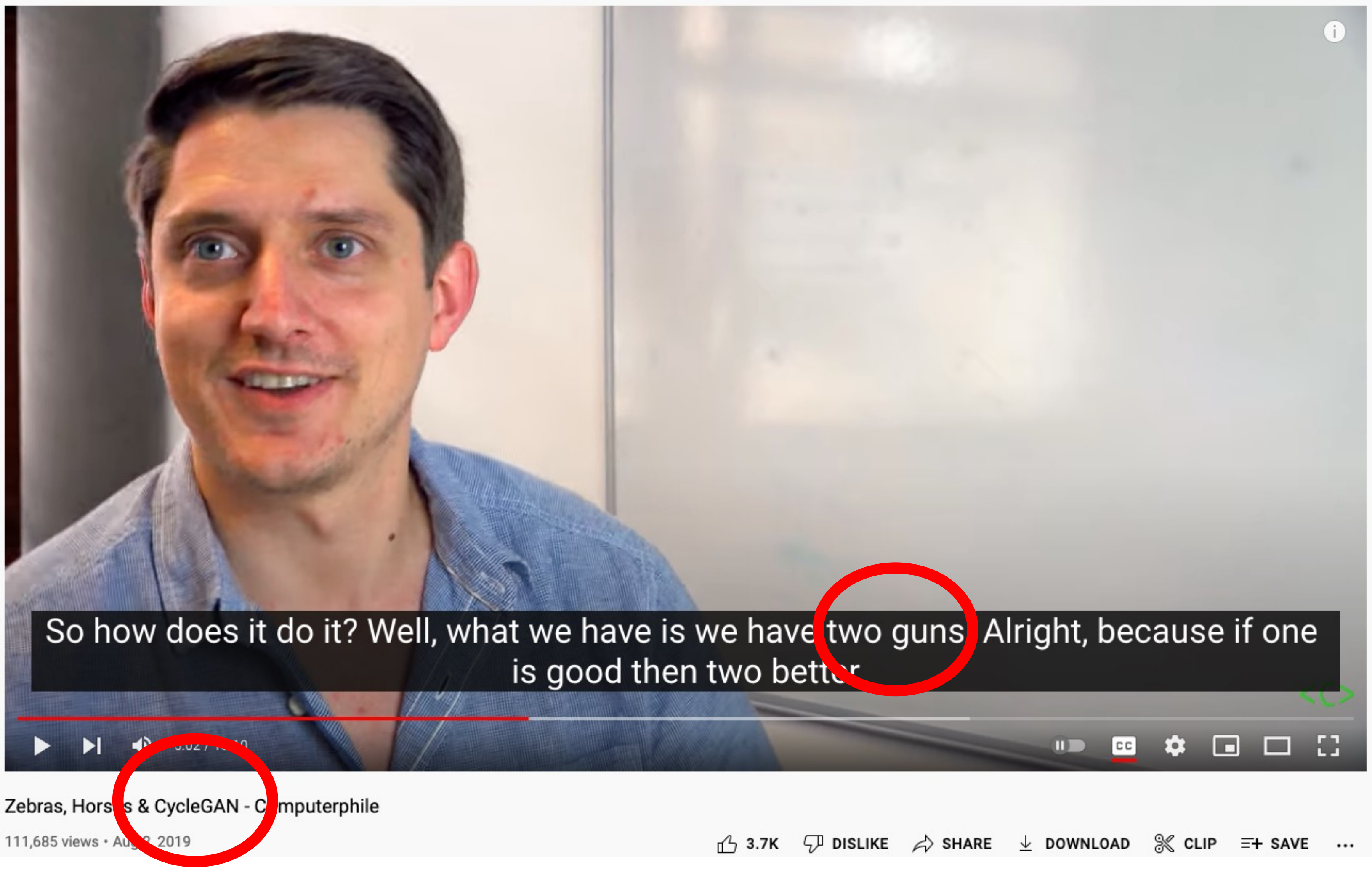
Common solutions

- Human Oracles
- “Explicit” Oracles
  - Crashes (e.g., SEGV, ABORT including sanitizer aborts)
  - Hangs
- Differential Testing [William M. McKeeman]
- Metamorphic Testing [T.Y. Chen et al.]

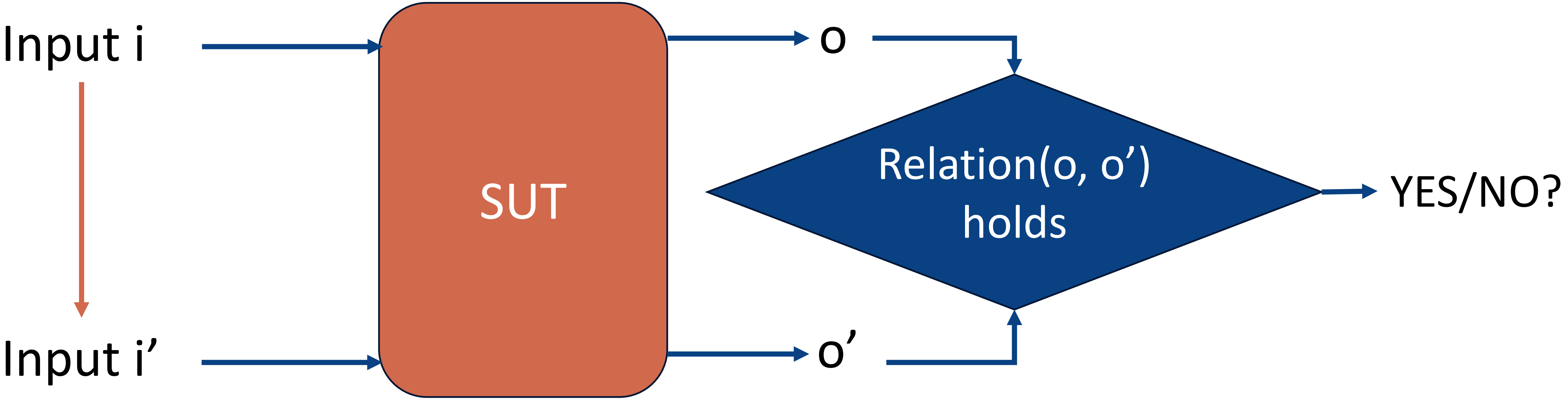
Differential testing



Do SUT-1 and SUT-2 need to solve the same problem?



Metamorphic testing



How would you identify metamorphic relations?

Q-1) What is the System/Component-Under Test?

Q-2) What transformations can be applied to the inputs? (e.g., deletion, semantic-preserving transformations, combination, addition)

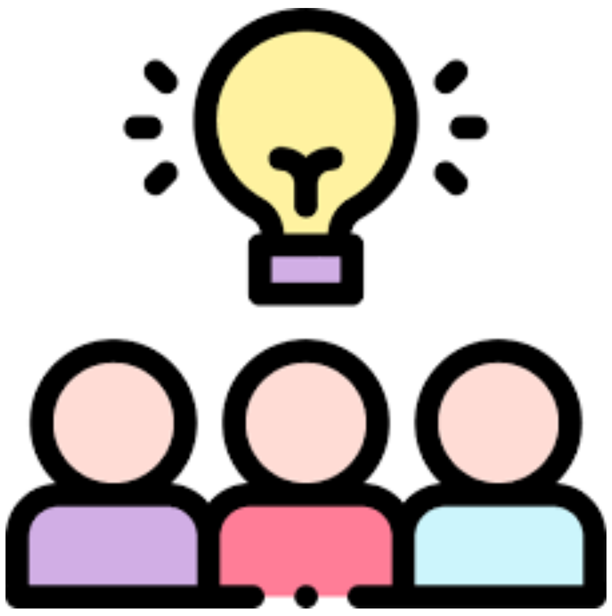
Q-3) Does it exist any relation between o and o' that should hold on all pairs (i, i') where i' = transformation(i)?

Activity: identify metamorphic relations

Task 1: to test a shortest path finding algorithm implementation (e.g., Dijkstra algorithm)?

Task 2: to test a C compiler (e.g., Clang/gcc)?

Task 3: to test an CNN-based image recognition system (e.g., used in autonomous cars)?



Group size: 4-5

15 minutes

Reading recommendation

Metamorphic Testing: A Review of Challenges and Opportunities

TSONG YUEH CHEN and FEI-CHING KUO,
HUAI LIU, Victoria University
PAK-LOK POON, RMIT University
DAVE TOWEY, University of Nottingham Ningbo C
T. H. TSE, The University of Hong Kong
ZHI QUAN ZHOU, University of Wollongong

DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars

Yuchi Tian
University of Virginia
yuchi@virginia.edu

Kexin Pei
Columbia University
kpei@cs.columbia.edu

Suman Jana
Columbia University
suman@cs.columbia.edu

Baishakhi Ray
University of Virginia
rayb@virginia.edu

Finding Bugs in Database Systems via Query Partitioning

MANUEL RIGGER, ETH Zurich, Switzerland
ZHENDONG SU, ETH Zurich, Switzerland

Automated Testing of Graphics Shader Compilers*

ALASTAIR F. DONALDSON, Imperial College London, UK
HUGUES EVRARD, Imperial College London, UK
ANDREI LASCU, Imperial College London, UK
PAUL THOMSON, Imperial College London, UK

EDEFuzz: A Web API Fuzzer for Excessive Data Exposures

Lianglu Pan, Shaanan Cohney, Toby Murray, Van-Thuan Pham
lianglup@student.unimelb.edu.au,shaanan@cohney.info,{toby.murray,thuan.pham}@unimelb.edu.au
The University of Melbourne, Melbourne, Australia

ABSTRACT

APIs often transmit far more data to client applications than they need, and in the context of web applications, often do so over public channels. This issue, termed *Excessive Data Exposure* (EDE), was OWASP's third most significant API vulnerability of 2019. However, there are few automated tools—either in research or industry—to effectively find and remediate such issues. This is unsurprising as the problem lacks an explicit test oracle: the vulnerability does not manifest through explicit abnormal behaviours (e.g., program crashes or memory access violations).

In this work, we develop a metamorphic relation to tackle that challenge and build the first fuzzing tool—that we call EDEFuzz—to systematically detect EDEs. EDEFuzz can significantly reduce false negatives that occur during manual inspection and ad-hoc text-matching techniques, the current most-used approaches.

We tested EDEFuzz against the sixty-nine applicable targets from the Alexa Top-200 and found 33,365 potential leaks—illustrating our tool's broad applicability and scalability. In a more-tightly controlled experiment of eight popular websites in Australia, EDEFuzz achieved a high true positive rate of 98.65% with minimal configuration, illustrating our tool's accuracy and efficiency.

> *"Automatic tools usually can't detect this type of vulnerability because it's hard to differentiate between legitimate data returned from the API, and sensitive data that should not be returned without a deep understanding of the application."*
>
> — The Open Web Application Security Project (OWASP)

> *"This vulnerability is so prevalent (place 3 in the top 10) because it's easy to miss. Automation is near useless here because robots can not tell what data should not be served to the user without telling them exactly how the application should work. This is bad because API's are often implemented in a generic way, returning all data and expecting the front-end to filter it out."*
>
> — Wallarm End-to-End API Security Solution

Figure 1: Industry views on the EDEs. These indicate the prevalence of EDEs and limitations of existing detection tools

We start with a definition: an API is vulnerable to EDE if it exposes meaningfully more data than what the client legitimately needs [1].

Consider a simple example of an online storefront. When a user views the page for a specific product, an API call may be made to

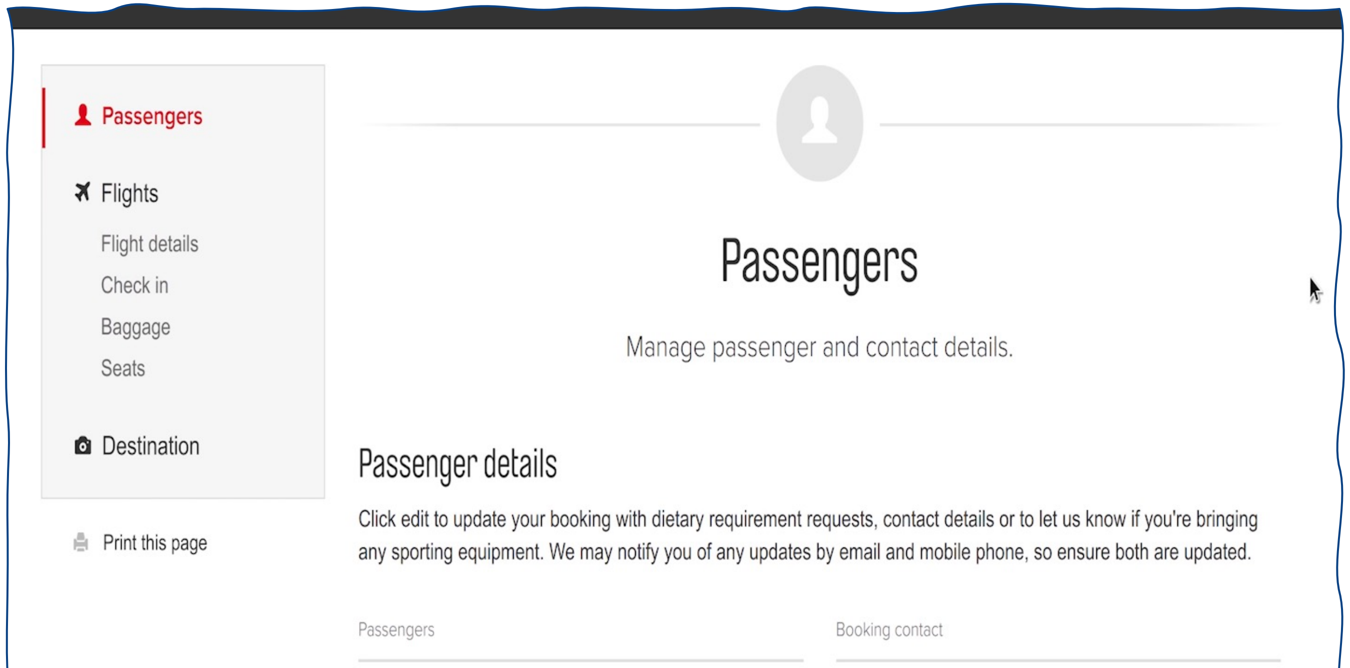
Detecting Excessive Data Exposures in Web APIs

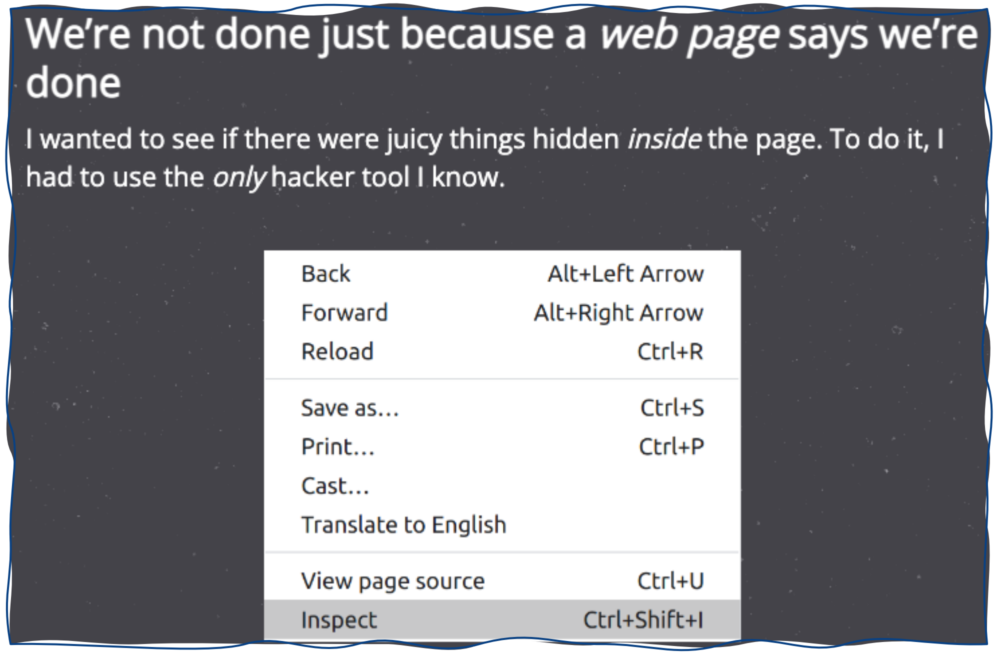
Why should we care about excessive data exposures/data leak?

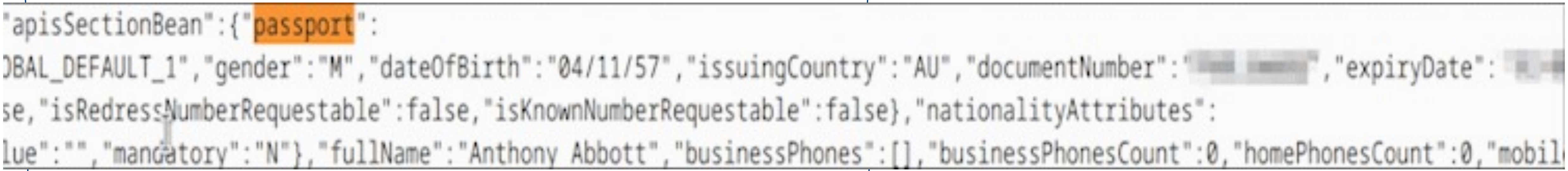

Tony Abbott hacked after posting boarding pass on Instagram

17 September 2020

What happened?







Details: https://mango.pdf.zone/finding-former-australian-prime-minister-tony-abbotts-passport-number-on-instagram

Excessive Data Exposures & Its prevalence

Ranked 3rd in OWASP 2019 TOP-10 critical vulnerabilities in APIs

- The API (e.g., Web API) returns full data objects as they are stored in the backend database.
- The client application filters the responses and only shows the data that the users really need to see.
- Attackers call the API directly and get also the sensitive data that the UI would filter out.

> *"Automatic tools usually can't detect this type of vulnerability because it's hard to differentiate between legitimate data returned from the API, and sensitive data that should not be returned without a deep understanding of the application."*
>
> — The Open Web Application Security Project (OWASP)

> *"This vulnerability is so prevalent (place 3 in the top 10) because it's easy to miss. Automation is near useless here because robots can not tell what data should not be served to the user without telling them exactly how the application should work. This is bad because API's are often implemented in a generic way, returning all data and expecting the front-end to filter it out."*
>
> — Wallarm End-to-End API Security Solution

Figure 1: Industry views on the EDEs. These indicate the prevalence of EDEs and limitations of existing detection tools

What are the consequences of exposing excessive data?

- (sensitive) data leakages
- Application performance
- Cost due to higher requirement for bandwidth
- ???

Manual Detection?

{"id":279980,"vin":"WVWZZZCDZNW001240","model_code":"CD13NS\/22","car_configurator_model_code":"CD13NS-GPJ3PJ3-GPLAPLA-GPRDPRD-GPZFPZF-GWA3WA3-GWCRWCR-
GWC4WC4","brand":"Volkswagen","sub_brand":"PV","model_year":"2022","is_freestock":**false**,"carline_model":"Golf","model_variant":"110TSI
Life","model_family":"Golf","body_type":"Hatch","body_type_sort_order":1,"common_color":"Yellow","vehicle_type":"New","transmission":"Auto","fuel":"Petrol","driven_wheels":"FWD","dea
ler_code":"30281","dealer_state":"VIC","dealer_postcode":"3875","mrdp":"46031","payload":{"vehicle_details":{"carline_model":"Golf","model_variant":"110TSI
Life","model_code":"CD13NS\/22","car_configurator_model_code":"CD13NS-GPJ3PJ3-GPLAPLA-GPRDPRD-GPZFPZF-GWA3WA3-GWCRWCR-
GWC4WC4","brochure_code":"GOLF","body_shape":"Hatch","model_year":"2022","engine_capacity":"1.4 TSI","transmission":"Auto","transmission_desc":"8-Speed
Automatic","fuel":"Petrol","driven_wheels":"FWD","model_family":"Golf","vin":"WVWZZZCDZNW001240","list_in_stock":**true**,"common_color":"Yellow","color_code":"C1C1","vehicle_type":"New"
,"vehicle_options":"MP,PM,PZC","customer_order_status":"5","is_freestock":**false**,"brand":"Volkswagen","sub_brand":"PV","demo_kilometers_driven":**null**,"demo_kilometer_updated":**null**},"de
fault_settings":{"default_finance_term":"48","default_finance_pa_comparison_rate":"8.82","default_finance_deposit":"10","default_finance_allowance":"60000","default_driveaway_disclai
mer":"a4r2e0000007IEWAA2","default_finance_disclaimer":"a4r2e0000007I9kAAE","default_driveaway_legend_icon":"~"},"images":{"new":{"hero_image_listing_page":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/right","hero_image_detail_page":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/front","gallery":{"front":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/front","back":"https:\/\/vga-images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/back","right":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/right","left":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/left"}},"demo":{"hero_image_listing_page":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/right","hero_image_detail_page":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/front","gallery":{"image0":"https:\/\/vga-
images.herokuapp.com\/VICCI\/stock\/image\/WVWZZZCDZNW001240\/front","image1":**null**,"image2":**null**,"image3":**null**,"image4":**null**,"image5":**null**}}},"dealer_details":{"dealer_code":"30281",
"dealer_name":"THE BIG GARAGE VOLKSWAGEN","dealer_state":"VIC","dealer_postcode":"3875","info":{"dealerName":"THE BIG GARAGE
VOLKSWAGEN","dealerCode":"30281","locality":"BAIRNSDALE","dealerAddress":"628 PRINCES HIGHWAY,
BAIRNSDALE","dealerState":"VIC","dealerPostcode":"3875","webKey":"","UsedCarDealer":"False","PartsAvailable":"True","PartsPostCode":"3875","SalesPostCode":"3875","ServicePostCode":"3
875","PartsStateCode":"VIC","SalesStateCode":"VIC","ServiceStateCode":"VIC","PartsSuberbCode":"BAIRNSDALE","SalesSuberbCode":"BAIRNSDALE","ServiceSuberbCode":"BAIRNSDALE","PartsEmail
Code":"","SalesEmailCode":"","ServiceEmailCode":"","PartsPhoneCode":"(03) 5152 4131","SalesPhoneCode":"(03) 5152 4131","ServicePhoneCode":"(03) 5152 4131","PartsStreetCode":"628
PRINCES HIGHWAY","SalesStreetCode":"628 PRINCES HIGHWAY","ServiceStreetCode":"628 PRINCES
HIGHWAY","PartsFaxCode":"","SalesFaxCode":"","ServiceFaxCode":"","FinanceEmail":"","ServiceDealer":"True","phone":"0351525313","tradingHours":[{"name":"Sunday","isclosed":**true**,"start
Time":"09:00 AM","endTime":"05:00 PM"},{"name":"Monday","isclosed":**false**,"startTime":"08:30 AM","endTime":"05:30 PM"},{"name":"Tuesday","isclosed":**false**,"startTime":"08:30
AM","endTime":"05:30 PM"},{"name":"Wednesday","isclosed":**false**,"startTime":"08:30 AM","endTime":"05:30 PM"},{"name":"Thursday","isclosed":**false**,"startTime":"08:30
AM","endTime":"05:30 PM"},{"name":"Friday","isclosed":**false**,"startTime":"08:30 AM","endTime":"05:30 PM"},{"name":"Saturday","isclosed":**false**,"startTime":"09:00 AM","endTime":"12:00
PM"}],"publicHolidays":["2018-11-06","2019-03-11","2019-01-28","2019-04-19","2019-04-21","2019-04-22"],"mapLocation":"","DealerOpTiming":[{"Type":"Parts","operatingTime":"Mon:0830-
1700,Tue:0830-1700,Wed:0830-1700,Thu:0830-1700,Fri:0830-1700"},{"Type":"Service","operatingTime":"Mon:0830-1700,Tue:0830-1700,Wed:0830-1700,Thu:0830-1700,Fri:0830-
1700"}]}}},"tile_info":{"background_color":"#e4d330","font_color":"#000"},"careplanData":{"model":"CD13NS\/22","careplan_model_code":"CD13NS\/22_CARE_PLAN_5","careplan_name":"5-Year
Care Plan","term":"60","price_inc_gst":"2400","price_exc_gst":"2181.82","Policy_Type":"Service Pack"},"optional_packages":[{"name":"Metallic
Paint","code":"GMPMP","price":"650"},{"name":"Premium Metallic Paint","code":"GPMPM","price":"900"},{"name":"Sound & Vision
Package","code":"GPZCPZC","price":"1500"}],"optional_packages_comb_string":"Metallic Paint, Premium Metallic Paint, Sound & Vision Package","banner":{"banner_heading":"MY22 Finance
Offer","sub_headings":["Free 3 Year\/45,000 KM Care Plan\u2020"],"application":"Offer;Stock","vehicle_type":"New","banner_end_date":"2022-06-30","banner_start_date":"2022-04-
01"},"pcsOfferIds":["a2E9q0000005AWvEAM"],"offers":{"Driveaway":{"offer_name":"Driveaway~","offer_heading_1":**null**,"offer_price":"0","applications":"Offer;Stock","start_date":"2022-
04-01","end_date":"2022-06-
30","legend_icon":**null**,"term":**null**,"pa_comparison_rate":**null**,"allowance":**null**,"deposit_required":**null**,"is_deposit_in_cash":**false**,"offer_type":"Driveaway","disclaimer":"a4r9q00000006n
8AAA","disclaimer1":**null**,"disclaimer2":**null**}},"discalimers":[{"id":"a4r9q00000006n8AAA","name":"D-0077","legend_icon":**null**,"description":"PV MY22 Golf Driveaway
Disclaimer","disclaimer":"~Manufacturer's recommended driveaway price (MRDP) for new MY22 vehicles in white sold and delivered on or after
01\/06\/2022."}],"dealer_comment":**null**},"last_modified_date":"2022-04-13 14:05:27","version":238,"status":1,"inserted_time":"2022-04-15 02:37:21","updated_time":"2022-04-15
02:37:21","did":"30281","ordering":23}

How can one automatically detect if a web API exposes more data than it should?

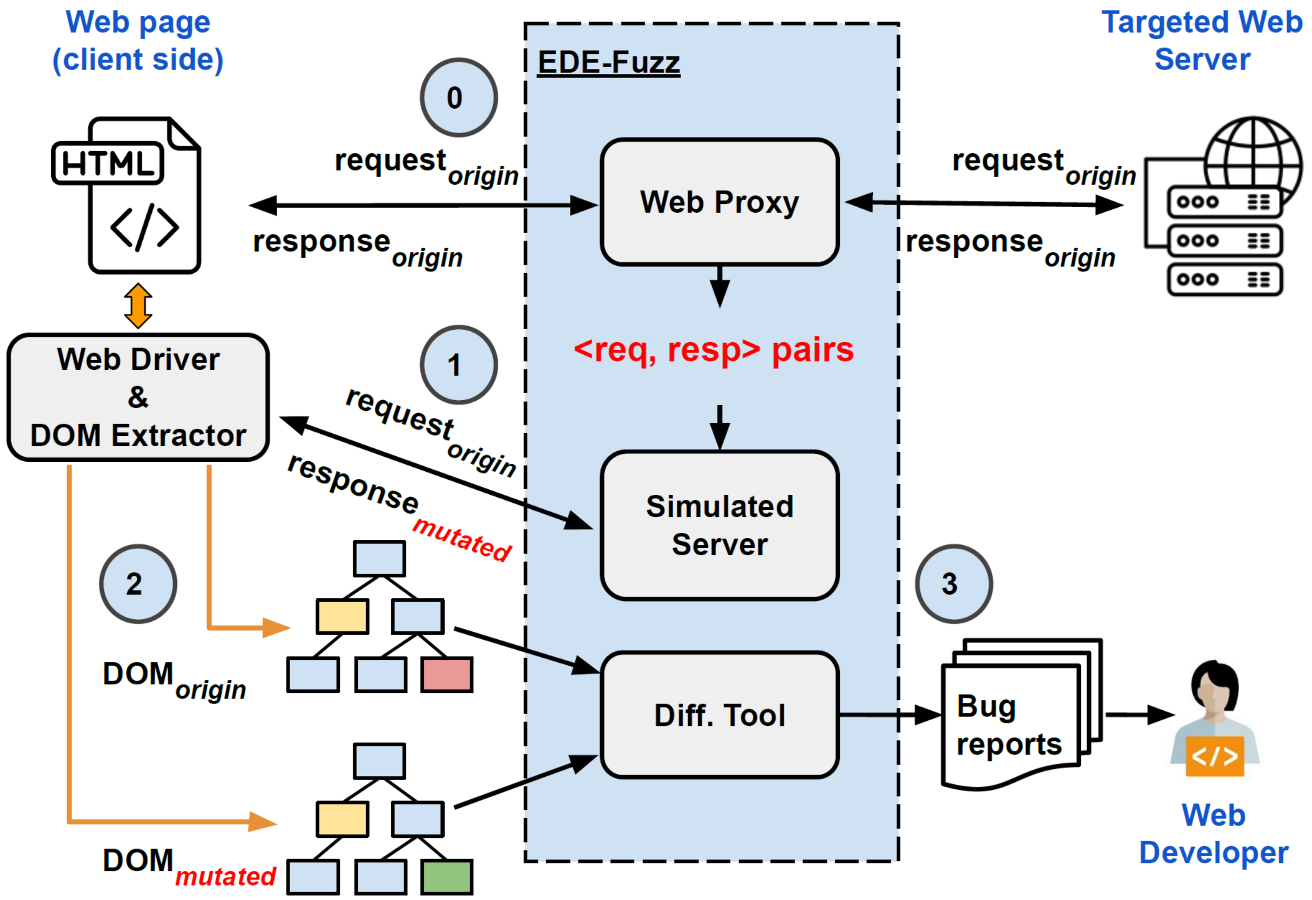
- The question is related to the famous ***test oracle problem***.
- We address this challenge with the following key insight:
  - ➢ Data returned from an API endpoint is more likely excessive if it has no impact on the content displayed to a user.
  - ➢ This is a metamorphic relation

if a field in a JSON response is not needed,
the rendered webpage should not change if the field is deleted.

Workflow of EDEFuzz

Three simple steps:

1) Save a copy of the rendered webpage after the API call
2) Try deleting *each of the fields* in the response
3) Check if the newly rendered page is the same as the saved copy. No change? The data was unnecessary



Results & Discussions

Target	Data fields	Reported	Confirmed	TP	Preparation (min)	Execution (min)	Classification (min)	Sensitive	Non-sensitive
Company-A	189	124	124	100.00%	10	11	5	0	124
Company-B	18	16	14	87.50%	20	2	2	2	12
Company-C	2600	2580	2504	97.05%	5	306	3	104	2400
Company-D	545	506	479	94.66%	15	43	10	9	470
Company-E	4249	4147	4127	99.52%	10	755	15	0	4127
Company-F	778	749	749	100.00%	15	103	5	0	749
Company-G	120	100	100	100.00%	5	12	3	0	100
Company-H	1465	1066	1066	100.00%	15	79	20	19	1047

Table 2: Summary statistics from the Australian sites. Data fields reports the total number of fields contained in the API response of each target, Reported is the number of fields flagged by EDEFuzz as excessive; Confirmed is the number of fields manually confirmed to be excessive, i.e. true positives, TP. The time taken to configure EDEFuzz for each target is reported in Preparation, as is the duration of test execution (Duration) and the human effort required to manually classify the flagged fields as sensitive or not (Classification), all measured in minutes. We also report (Sensitive) the number of fields we classified as containing sensitive data, after manual inspection.

EDEFuzz is now open source at

https://github.com/Broken-Assumptions/EDEFuzz

SCAN ME

Reading recommendation

Toss a Fault to Your Witcher: Applying Grey-box Coverage-Guided Mutational Fuzzing to Detect SQL and Command Injection Vulnerabilities

Erik Trick
Giovanni Vigna[†], Christopher Kru

Automated Black-box Testing of Mass Assignment Vulnerabilities in RESTful APIs

Davide Corradini[*], Michele Pasqua[†] and Mariano Ceccato[‡]
Department of Computer Science

FuzzOrigin: Detecting UXSS vulnerabilities in Browsers through Origin Fuzzing

Sunwoo Kim[*]
Samsung Research
sunwoo28.kim@samsung.com

Young Min Kim
Seoul National University
ym.kim@snu.ac.kr

Jaewon Hur
Seoul National University
hurjaewon@snu.ac.kr

Suhwan Song
Seoul National University
sshkeb96@snu.ac.kr

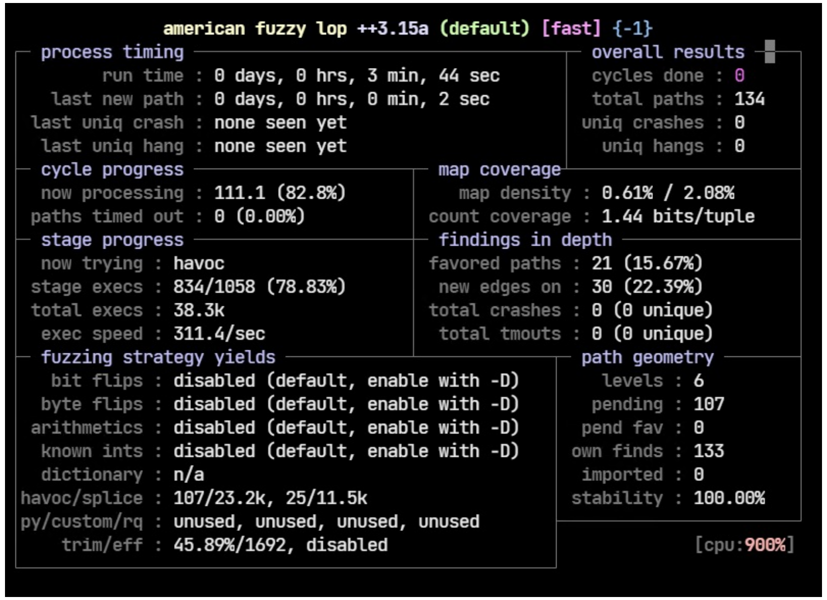
Gwangmu Lee[†]
EPFL
gwangmu.lee@epfl.ch

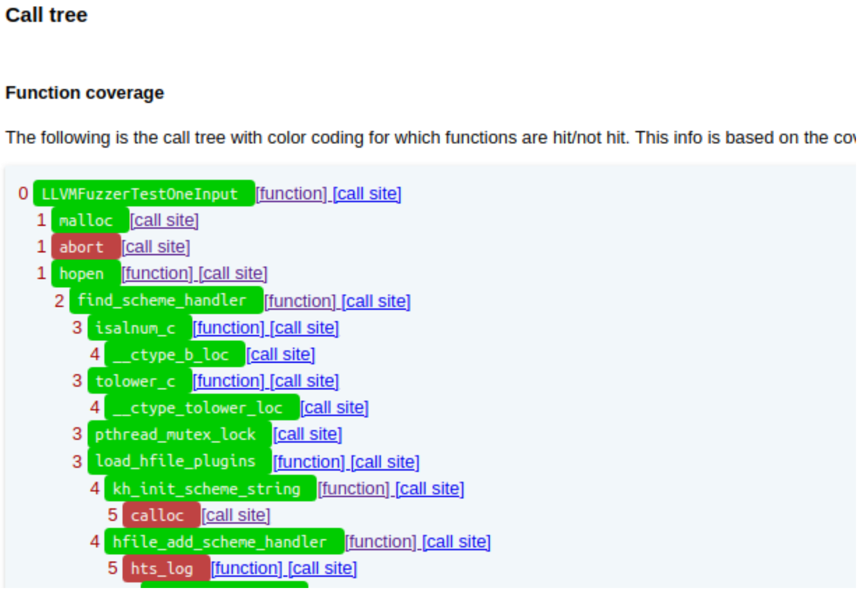
Byoungyoung Lee[‡]
Seoul National University
byoungyoung@snu.ac.kr

Beyond The Coverage Plateau

Human-In-The-Loop Fuzzing

1. How do humans and fuzzing tools communicate? *--- so they can "understand" each other*
2. When should they talk to each other? *--- so humans are not overwhelmed*
3. What humans can help? Can we reuse their previous suggestions?
4. How does fuzzing leverage humans' guidance?
5. How do we improve the Graphical User Interface (GUI) to support effective human-fuzzing interaction?





What can humans help?

Registered Report: Beyond The Coverage Plateau - A Comprehensive Study of Fuzz Blockers

Wentao Gao
wentaog1@student.unimelb.edu.au
The University of Melbourne

Van-Thuan Pham
thuan.pham@unimelb.edu.au
The University of Melbourne

Dongge Liu
donggeliu@google.com
Google

Oliver Chang
ochang@google.com
Google

Toby Murray
tobby.murray@unimelb.edu.au
The University of Melbourne

Benjamin I.P. Rubinstein
benjamin.rubinstein@unimelb.edu.au
The University of Melbourne

ABSTRACT

Fuzzing and particularly code coverage-guided greybox fuzzing, has proven highly successful in automated vulnerability discovery, as evidenced by the multitude of vulnerabilities uncovered in real-world software systems. However, results on large benchmarks such as Google FuzzBench indicate that the state-of-the-art fuzzers often reach a plateau after a certain period, typically around 12 hours. With the aid of the newly introduced Fuzz Introspector platform, this study aims to analyze and categorize the fuzz blockers that impede the progress of fuzzers. Such insights can shed the light for future research directions in fuzzing, suggesting areas that require further attention. Our preliminary findings reveal that the majority of top fuzz blockers are unrelated to the program input, emphasizing the need for enhanced techniques in automated fuzz driver generation and modification.

KEYWORDS

fuzzing, vulnerability detection, software security

ACM Reference Format:
Wentao Gao, Van-Thuan Pham, Dongge Liu, Oliver Chang, Toby Murray, and Benjamin I.P. Rubinstein. 2023. Registered Report: Beyond The Coverage Plateau - A Comprehensive Study of Fuzz Blockers. In *Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2023)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

published to improve the technique in various aspects [34]. These efforts have focused on enhancing fuzzing in areas such as feedback collections [16, 24, 26, 29], corpus management [28], seed selection algorithms [21, 22], input generation algorithms [15, 19, 30, 42, 45], and novel test oracle designs [37, 43]. Additionally, researchers have attempted to extend the applicability of fuzzing to challenging targets such as network protocols [17, 41], database systems [43, 48], SMT solvers [38], compilers [25], device drivers [39], and heterogeneous applications [46]. Another noteworthy research direction is parallel or distributed fuzzing [32, 36, 40], which aims to improve fuzzing efficiency by utilizing high-performance computing resources.

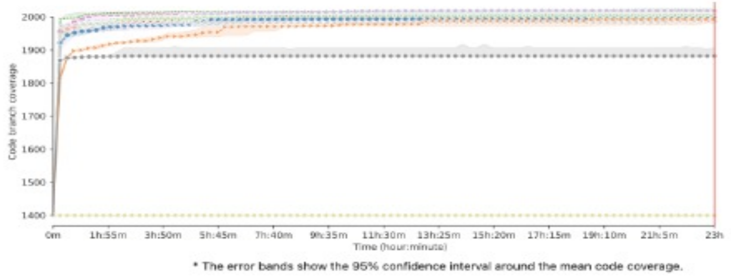


Figure 1: SBFT'23 Fuzzing Competition result of LibPNG. Mean branch coverage growth over time is reported. 17 trials/fuzzer, 23 hours per trial. Most fuzzers reach their plateau after 14 hours.

Fuzz blockers analysis

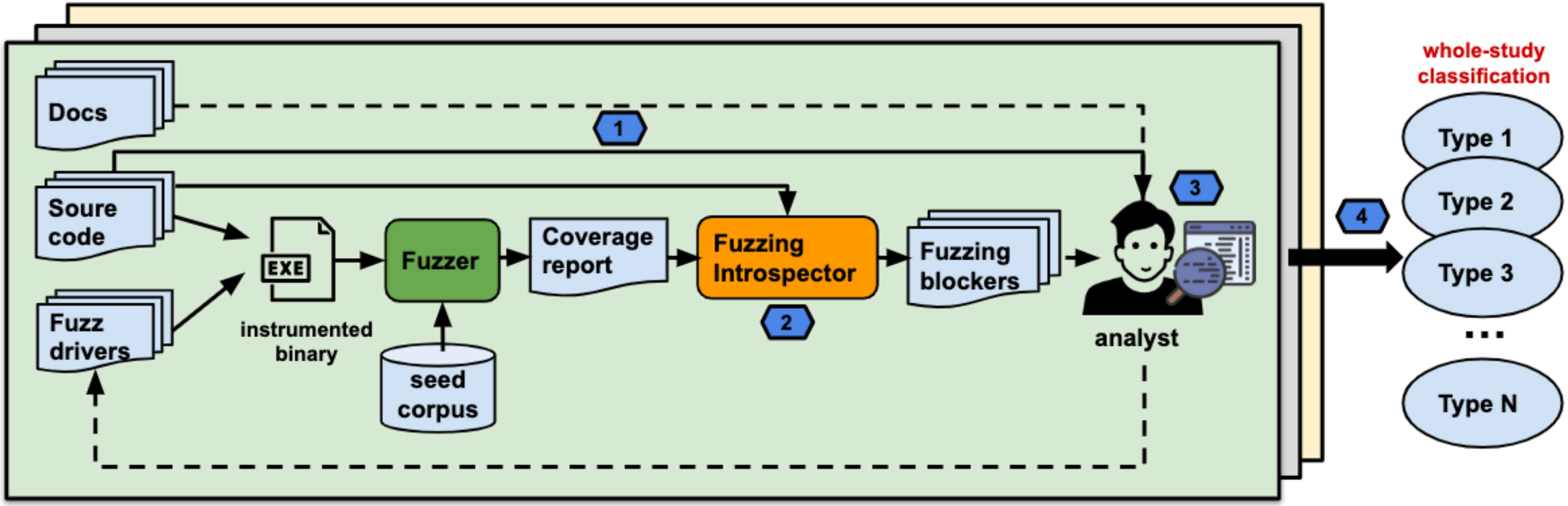


Figure 5: The 4-step workflow to conduct our study. (Step 1 - Manual) Understanding subject program; (Step 2 - Fully Automated with Fuzz Introspector [5]) Identifying fuzz blockers; (Step 3 - Manual) Analyzing fuzz blockers; (Step 4 - Semi-Automated using taint analysis) Classifying fuzz blockers. Dashed lines indicate that the steps/flows are optional.

Preliminary study & initial results

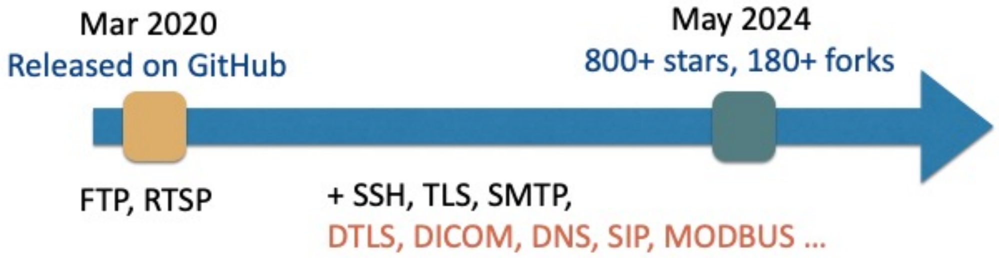
Libraries	Size	Functionalities	Function Reachability Result	#Fuzz drivers
LibPNG	105k	Image processing	52%	1
iGraph	520k	Graph analysis	25%	11
OpenSSL	1570k	Crytography	28%	13

Table 1: Three subject libraries of our preliminary study. We report the code size in Lines of Code (LoC) and the function reachability results are reported by Fuzz Introspector. Code size information is taken from Black Duck Open Hub [2], which is a website tracking and comparing open source projects.

- We have classified the fuzz blockers five types:
  - **Type-1:** due to wrong function arguments
  - **Type-2:** due to missing function call(s)
  - **Type-3:** due to missing different order(s) of function calls
  - **Type-4:** due to missing "extreme" inputs
- 12/12 top fuzz blockers of LibPNG are *input independent*

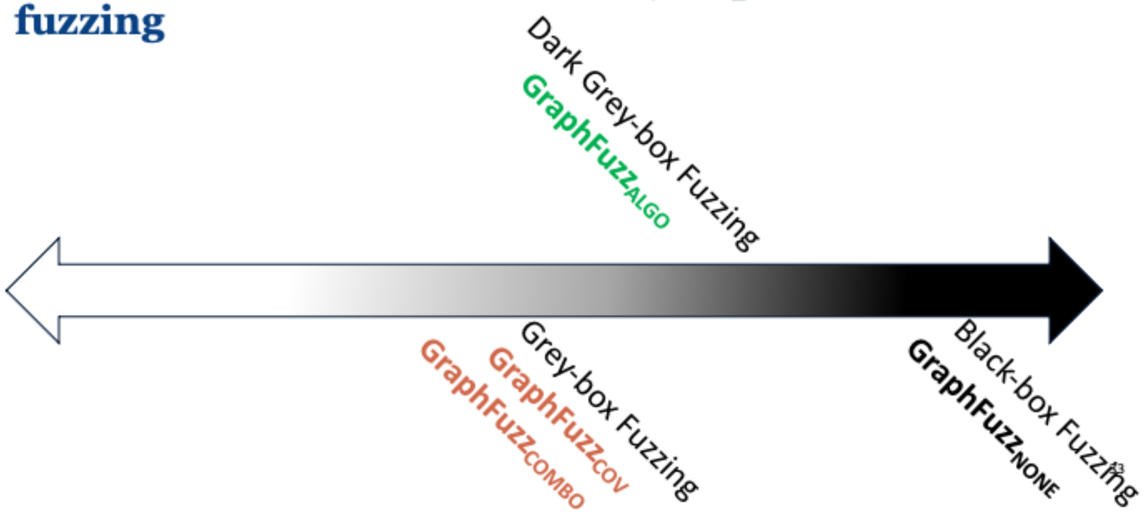
AFLNet keeps evolving

https://github.com/aflnet/aflnet



RQ-1. Effectiveness & Efficiency of the new signals

RQ-2. Effectiveness & Efficiency of pure black-box fuzzing



Workflow of EDEFuzz

Three simple steps:

1) Save a copy of the rendered webpage after the API call
2) Try deleting *each of the fields* in the response
3) Check if the newly rendered page is the same as the saved copy. No change? The data was unnecessary