p-fuzz: an efficient fuzzing tool with parallel computing mechanism

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Nowadays, software is applied widely in our life, which accompanied with the occurrence of security vulnerabilities. Attackers always utilize these bugs and errors in codes to make target crash or grab some sensitive data. Therefore, it is urgently needed for us to pay close attention to find an effective approach to test software.

Software testing has two major technologies: symbolic execution and fuzzing. The symbolic execution abstracts the input values as symbols, which could lead symbolic engine to explore as many as possible execution paths at the same time. Then getting the results by solving constraints. However, the whole process of symbolic execution results in state space explosion which is still a bottleneck. As opposed to symbolic execution, fuzzing provides invalid, unexpected or random data to the inputs of a program which significantly enhances the performance of software testing. Fuzzing tools can be classified into three types based on the knowledge and information acquired from the source code of target programs, they are white-box, black-box, and grey-box fuzzer. The white-box fuzzer has full knowledge of the source code (eg. internal logic and structure) and uses the control structure of the procedural design to derive test cases. In contrast, The black-box fuzzer doesn't have any knowledge of source code but it generates test cases randomly and swiftly. The grey-box fuzzer try to combine the efficiency and effectiveness of black-box fuzzers and white-box fuzzers, which masters limited knowledge of the internal working of target programs.

Furthermore, existing fuzzers have been effective mainly in discovering superficial vulnerabilities and fail to uncover the vulnerabilities in deep paths without valid guidance. Through collecting the feedback information of target programs, grey-box fuzzers show the competitiveness of mutating test cases with valid guidance. It is implemented by lightweight instrumentation or other mechanisms to get program execution feedback, such as code coverage for the fuzzing process. American Fuzz Lop(AFL) is a state-of-the-art grey-box fuzzer whose principles are speed, reliability, and ease of use. AFL instruments the compiled program to get the edge coverage information. It adopts a deterministic strategy and non-deterministic strategy to generate test cases by mutating input seeds. The "interesting" change will be recorded for further detecting. Several works achieved significant results based on making an extension of the AFL. Böhme et al. designed AFL-fast which assigned more mutation energy to interesting paths. Gan et al introduced CollAFL which mitigated the path collisions by providing more accurate coverage information. Böhme also implemented a directed grey-box fuzzing tool AFLGO towards the dangerous locations which tend to produce vulnerabilities. All of these extensions gained higher coverage and found more bugs than AFL. Zhang et al. leverage hardware mechanism (Intel Processor Trace) to collect branch information, and feed this information back to the fuzzing process.

Nevertheless, there are some common challenges in single-computer structure fuzzing.

- The fuzzing process contains too many tedious mutations.

Identify applicable funding agency here. If none, delete this.

- The fuzzing efficiency and effectiveness are relatively low because of executing too much-repeated work.

In order to address the challenges, many researchers leverage parallel computing technology to speed up the fuzzing process. Parallel computing indicates a type of computation where many calculations are executed simultaneously. It collects a group of computing resources to decompose the heavy fuzzing task. Test cases produce by mutation are allocated to each computer, which balances the system workload. Some researchers have proceeded works about parallel or distributed fuzzing. Xie using grid computing for large scale fuzzing in 2010, which reduce almost two-thirds of fuzzing time. It was implemented by dividing fuzzing jobs into tasks, storing them in a server and scheduling remote clients to download them. Lian et al. proposed a dynamic resource-aware approach for parallel fuzzing. Some distributed fuzzing tools based on the parallel function of AFL are implemented in the client-server model. The clients synchronize their queue to server continuously which benefit from each other's work.

Despite the parallel computing technology accelerate the fuzzing process, it faces several challenges and difficulties:

- The client computers are always fuzzing same test cases caused the concurrency and race.
- The transferring speed is limited while the quantity of common resources is large enough.
- Fixed synchronizing time entails the response latency to a new test case.
- All clients share the same seeds get from the server. The approach tires clients and entails the low resources utilization of test cases.

By summarizing these research above, we propose a parallel fuzzing platform —p-fuzz, which not only alleviates the problems but also speeds up the fuzzing process by leveraging abundant parallel resources.

II. BACKGROUND

A. Parallel computing technology

Parallel computing specifies a type of computation where many calculations are executed simultaneously. According to different granularities, parallel computing can be classified into bit-level, instruction-level, data-level, and task-level parallelism. The task-level parallelism means a large job can be divided into several small tasks, and each node of parallel computers gets a piece of the task and execute it. In this paper, we talk about task-level parallelism.

The origin of parallel computing technology dates back to 1950s. John Cocke and Daniel Slotnick discuss the use of parallelism in numerical calculations. In the 1970s, The e-mail was invented and became the earliest and most successful example of a large-scale distributed application in ARPANET. By demanding and increase exponentially, distributed and parallel computing became its own branch of computer science in the 1980s. In the 1990s, client-server architecture appeared and became popular. After 2000, With the big data era coming, grid computing and cloud computing provided various, massive and prompt services by their extraordinary computing ability. Nowadays, supercomputer plays an important role in computing. It handles a wide range of computationally intensive tasks in various fields.

To be specific. a distributed parallel application always falls into one of several basic architectures: client-server, three-tier, n-tier, or peer-to-peer and database-centric Architecture. Database-centric architecture specifies the software architecture in which databases is a core of the whole system. The architecture provides reliability, performance, and capacity, and scalability. We implement the p-fuzz platform with the help of database-centric architecture, and details of implementation are discussed in section III.

B. The details about AFL

In this section, we will discuss fuzzing techniques and AFL in detail. These limitations motivate us to propose our p-fuzz. American Fuzzy Lop(AFL) is an instrumentation-guided grey-box fuzzer. The fuzzer stays brute-force that makes the fuzzing process keep speed. AFL is designed for such goals, i.e,

- Speed: adopting an appropriate instrument approach not only gives guidance to fuzzing but also keeps its native speed.
- Rock-solid reliability, it adapts to real-world targets.
- Simplicity. AFL is simple and user-friendly.

There are two things we need to care about: the **seed** and **bitmap**.

1) Seed: Seed indicates the test cases which trigger the fuzzer to traverse new interesting paths. A queue is maintained to store the seeds. The high-quality corpus of candidate files will be selected as interesting seeds for further rounds.

2) Bitmap: Before talking about bitmap, we need to define the "new branch". If there are three basic blocks A, B, C in a program, the tuple (AB, BC) describes a branch transition. A new branch indicates a branch transition which doesn't appear before. In AFL, if a new branch is triggered by a test case, it is considered as a new path. All of the branches information is recorded in the bitmap. Bitmap describes the coverage of fuzzing, which stored in shared memory. the index of a bitmap is produced by previous basic block and current basic block. How to index a branch transition is shown as following. *A* and *B* represent previous basic block and current basic block respectively.

$$(A \oplus B)\%BITMAP\_SIZE \tag{1}$$

However, AFL is inaccurate because of the path collision caused by an infinite space of bitmap. It prevents AFL from discovering potential paths that lead to new crashes.

C. The discussion of parallel mechanism in fuzzing

In fact, if you only put a single job on a multi-core system or a multi-machine computing group, you will be underutilizing the hardware. At this time, parallel the computing resources can make full use of hardware and bring profit to low-efficiency fuzzing process. We will discuss how to make parallelization in this section.

Algorithm 1 123

```
inti = 0
scanf("%d", &i)
if i > 0 then
    if i < 100000&&i >= 1000 then
        if i >= 5000&&i < 10000 then
            if i >= 7000&&i < 8000 then
                if i >= 7500&&i <= 7599 then
                    if i >= 7545&&i < 7571 then
                        printf("win")
                    end if
                end if
            end if
        end if
    end if
end if
```

1) A Naïve approach: A naïve approach to improve the efficiency of fuzzing is starting a group of fuzzers. However, the approach fails to produce a satisfying result and occupy resources, we take a simple experiment to show this case. As is shown in algorithm.1 the simple program contains 6 levels branches. If we only run a single AFL engine, it reaches the "win" location in 2 minutes and 34s. Compared with the single engine, we run two engines in two cores simultaneously, the time of reaching "win" location keeps at the same level.

Also, we respectively run an AFL engine and two AFL engines in two cores on a target program "uniq" in lava-m, and compare their bitmap density. In one hour, a single engine and two engines get same results. Both of them hit 214 new branches, which are shown in bitmaps.

2) Previous parallel fuzzing application: There are two fuzzing tools extent the parallel function in AFL. One is Roving, which is implemented by running multiple copies of AFL on multiple machines in a cluster, all of them fuzzing the same target. It benefits from the client-server structure which shares crashes, hangs, and queues of each client. Every 300 seconds, the client update the fuzzing environment by uploading and downloading changes. The whole framework is scheduled by the central server.

The other is distributed fuzzing. The main work of distributed fuzzing is similar to roving, and the difference of them is implementation. the sharing data is handled by PHP scripts in fuzzing server. All of the target projects are stored in a server. Clients download 1 or more projects(according to CPU cores) and AFL fuzzes program from the server. New queue and hangs and crashes produced by a client will be synchronized to the server and downloaded by other clients in fixed time gap.

Although the two frameworks utilize the computing resources and parallel the fuzzing progresses, which makes each client benefits from each other's work, they have drawbacks as below.

- As time goes by, the seeds, queues, crashes and hangs entail the synchronizing speed slower and slower.
- This kind of sharing mechanism makes all of the clients always fuzzing the same seeds.
- The server accepts all data from clients updated, which will result in security problems.

D. Concurrency and data race

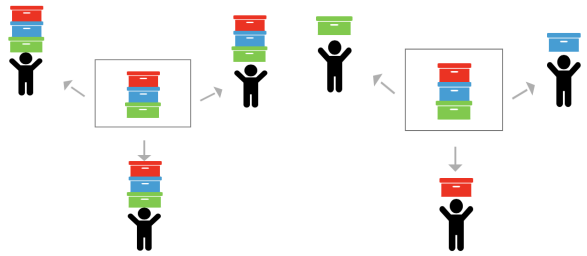
In parallelization computing, some uncontrolled accesses to shared data happen simultaneously, which results in race conditions. Data races are race conditions occur at memory access level, which are the most common causes of the concurrency errors and bugs.

A data race occurs when two actions are accessing the same memory, and at least one of the two accesses is a write, and the sequence of accesses is not assured by synchronization primitives. To prevent memory access from data race, some methods such as lock, semaphore, and mutex are adopted. The database always adopts transactions to solve some level of data races after recovery from a crash to maintain the atomicity, consistency, isolation, and durability.

III. METHODOLOGY

To improve the fuzzing speed and make full use of computing resources, we design p-fuzz, which is a parallel fuzzing framework.

A. The algorithm of balancing workloads



(a) The example of distributing workloads by previous works (b) The example of distributing workloads by p-fuzz

Fig. 1. Example of distributing workloads

Hardware resources and fuzzing tasks are two entities of parallel fuzzing. And the most important work is to distribute fuzzing tasks to hardware resources appropriately. previous studies show us two drawbacks in tackling this work:

- Underutilizing the hardware resources, which burdens the single core with many fuzzing tasks.
- Sharing all information including(seeds, queues, crashes and hangs) with each of client, which may results all computing cores do repeated work and doesn't fully reflect the advantages of parallel. This case is depicted in Fig.1(a).

To make full use of hardware resources and enhance fuzzing efficiency, we schedule the fuzzing tasks to balance workload with the help of Database-centric architecture as Fig.1(b). Database-centric architecture put a database as a core of the whole system, and other hardware resources act as clients

to communicate with the database. The p-fuzz framework is designed based on the database-centric architecture as shown in Fig.2. We deploy a server with a database to communicate with other hardware resources. Also, we mark the sharing records with flags and time stamps in the database, to differentiate whether this record is occupied by a client. Furthermore, we start services to monitor the server which can not only schedule the fuzzing tasks, but also solve the race problem from parallelization. In this way, all of the hardware resources get different seeds and do different tasks in the scheduling of p-fuzz mechanism.

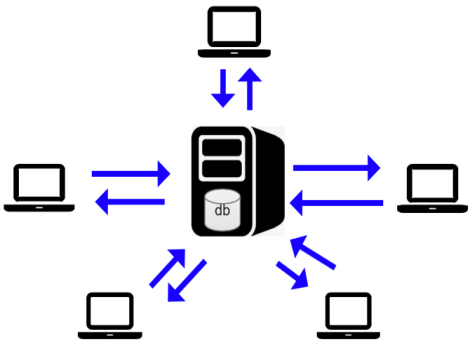


Fig. 2. the framework overview of p-fuzz

B. Immediate response to update

Different from roving and distributed fuzzing which synchronizes the sharing data in a fixed time gap, p-fuzz updates the new seeds and bitmap data to the database when AFL produces them.

When a test case triggers a new interesting path in the fuzzing process, the test case will be uploaded as a record in the database. Also, the bitmap stored in the database will be updated in time when a client find of a new path.

It's a prompt action to make all clients in the system get the information immediately.

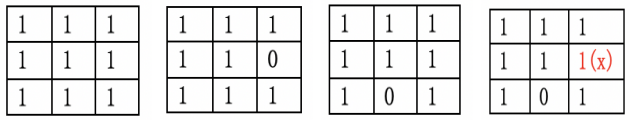
C. Data race handling

1) Flag:two clients want to access the same seed simultaneously.: As above mentioned, we sharing seeds each AFL produced by storing them into a database. Different clients access to different seeds to enhance the fuzzing efficiency. However, when two clients access to a seed simultaneously, a data race happen. The same seed fuzzed repeatedly will produce a similar result.

To alleviate this case, we set a flag with seeds, which marks whether this seed is fuzzing by a client. If the flag is "1", the client will choose other seeds to fuzz.

2) Service:some clients update bitmap in the database simultaneously: We store the bitmap as a record in the database for sharing. A data race happen as shown in Fig.3. some bitmaps with "1" or "0" represent the path uncovered or covered. There are two clients updating their new bitmaps together(Fig.3(b)(c)). If we do not control the updating process, information will get lost which is shown in Fig.3(d).

To alleviate this case, we start a service in the server to maintain the sequence of updating. The service builds a queue to store the bitmap temporarily. When bitmaps come up together, they are enqueued according to the time order. The database merges these bitmaps in the queue one by one so that the bitmap maintains all necessary information.



(a) the origin bitmap (b) the bitmap from updated client1 (c) the bitmap from updated client2 (d) the final bitmap

Fig. 3. race of updating bitmap from different clients

3) Time stamp:a client quits fuzzing accidently but doesn't finish a complete fuzzing round: as above mentioned, we set a flag to mark whether the seed is occupied by a client. However, in parallel computing, a client sometimes quit with errors or other accidents. At this time, the flag is "1" but the fuzzing process of the corresponding seed isn't finished.

To solve this problem, we put a time stamp when the flag is set to "1". We also monitor if the fuzzing is overtime by the current time minus the time stamp. This mechanism assures exceptions won't disturb the parallel fuzzing.

IV. IMPLEMENTATION

A. Workflow

The workflow of p-fuzz is shown below:

- Setting up the database in the server machine
- Configuring the services
- Starting afl in each clients
- Getting the results

B. Server

The server machine is the core of the whole system. We deploy a MongoDB database on the server to store the sharing data.

Hash value(key)	Seed content	flag	Time stamp
028412495	seed	1	201901011245
042315831	seed1	0	201901011257

(a) the seed collection in the database

Key	Bitmap
1	111111111

(b) the bitmap collection in the database

Fig. 4. the two collections of database

1) MongoDB: MongoDB is an open-source document database, which with high performance, high availability and automatic scaling.

As shown in Fig.4(a)(b), We set two collections in the database. One is "seed", the first row is the key to the collection which is a hash value of file name , the second row records the content of seed, the third row is flag to mark whether the seed is being fuzzed, and the last row is timestamp, which is used to mark when the fuzzing start.

The other is "bitmap", the first row is the key of the collection is a hash value of file name, the second row records the sharing bitmap.

2) Service: As shown in Fig.5, we start a service in the server to maintain the sequence of updating. The service binding with server keeps running and records the time of each bitmap coming. When bitmaps from several clients are sent to the server together, the service put them into a queue according to the time order. The queue provides bitmaps to the database to merge them into latest state continuously.

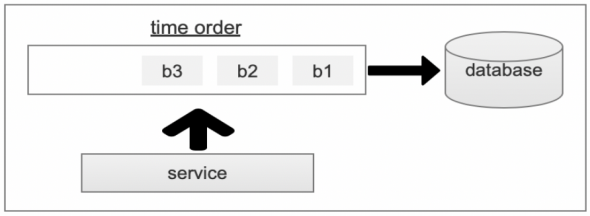


Fig. 5. the workflow of service

C. Client

We choose several computers in a local area network as clients. We deploy p-fuzz client, which is a parallel fuzzing AFL version on each client, and put the same program to being fuzzed. At the start of fuzzing, each client downloads a seed from the central database. When the fuzzing engine finds some interesting paths, it updates these new seeds to the central database. Furthermore, the write and read in fuzzing is conducted by updating or downloading records to or from the central database. We achieve sharing data in parallel fuzzing by this mechanism

V. EXPERIMENT

A. experiment setup

B. comparison in crashes and branches

C. comparison in speed

VI. DISCUSSION

A. limitation

B. future work

VII. CONCLUSION

$$a + b = \gamma \tag{2}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(2)", not "Eq. (2)" or "equation (2)", except at the beginning of a sentence: "Equation (2) is . . ."

A. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

B. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).

- Do not use the word “essentially” to mean “approximately” or “effectively”.
- In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
- Do not confuse “imply” and “infer”.
- The prefix “non” is not a word; it should be joined to the word it mo
- There is no “et al.”.
- The abbrev ation “e.g.’

An excellent sty

C. Authors and

The class fi authors. A min ence articles. A to right and the author sequence indexing service group by affiliat possible (for ex of the same org

D. Identify the

Headings, or reader through heads and text

Component your paper and Examples inclu these, the corre caption” for yo table title. Run- to apply a style provided by the the text.

Text heads o basis. For exan because all subs one topic. If t level head (upp conversely, if t subheads shoul

E. Figures and

a) Positio tables at the to in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert

TABLE I
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

Fig. 6. Example of a figure caption.

figures and tables after they are cited in the text. Use the abbreviation "Fig. 6", even at the beginning of a sentence.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.