Final Report
ECE1776
Amarpreet Singh, 1002513764

Introduction

The approach of the project comprises two objectives.

The first objective of the project is to determine the coverage of the AFL fuzzer when the input programs are the following GNU coreutils programs in the LAVA-M dataset: base64, md5sum, uniq and who [1].

The second objective of the project is to improve the coverage of the AFL fuzzer when the input programs are the following GNU coreutils programs in the LAVA-M dataset: base64, md5sum, uniq and who [1]

Associated work

The associated work that improves the coverage of the AFL fuzzer is in the paper named CollAFL: Path Sensitive Fuzzing, and the associated work comprises two studies [2].

The first study in the associated work is about the bitmap in the AFL fuzzer that includes the information about the hash of the edges in the control flow graph of the input program.

The second study in the associated work is about the runtime performance of the AFL fuzzer to complete the fuzzing after modifying the size of the bitmap that includes the information about the hash of the edges in the control flow graph of the input program.

The work to recreate and extend the associated work in the paper named CollAFL: Path Sensitive Fuzzing could include two studies.

The first study would recreate the associated work that is about the runtime performance of the LLVM mode of the AFL fuzzer to find the program paths in the input program after modifying the size of the bitmap in the AFL fuzzer.

The second study would extend the associated work after finding the size of the bitmap in the AFL fuzzer based on the input program that would increase the runtime performance of the LLVM mode of the AFL fuzzer to find the program paths in the input program.

Design

The experiments to recreate and extend the associated work in the paper named CollAFL: Path Sensitive Fuzzing could include two experiments.

The first experiment would recreate the study of the associated work that is about the runtime performance of the AFL fuzzer when the size of the bitmap in the AFL fuzzer is modified.

The second experiment would extend the study of the associated work after finding the size of the bitmap in the AFL fuzzer based on the input program that would increase the runtime performance of the AFL fuzzer to find the program paths in the input program.

The design of the code that could find the size of the bitmap in the AFL fuzzer in

order to improve the performance of the AFL fuzzer to find the program paths in the input program could be in the following diagram:

```
Input:
IP = INPUT_PROGRAM
DP = DIRECTORY_PATH
LM = LLVM_MODE
TS = TESTCASE_SIZE
Output:
BP = BITMAP_SIZE
  1:  bitmap_size = 16
  2:  good_result = 0
  3:  for bitmap_size = 10 to 20 do
  4:     obtain_the_AFL_fuzzer_results
         (IP, DP, LM, bitmap_size, TS)
  5:  end for
  6:  for result in the_AFL_fuzzer_results do
  7:    good_result = max(good_result, result)
  8:  end for
  9:  return good_result.bitmap_size
```

Diagram: The design of the code that could find the size of the bitmap in the AFL fuzzer in order to improve the performance of the AFL fuzzer to find the program paths in the input program.

The function named obtain_the_AFL_fuzzer_results could be in the Evaluation of the final report.

The function named max could obtain two AFL fuzzer results and provide the AFL fuzzer result that includes more total paths.

Experimental methodology

The first step of the experimental methodology of the project could be to start the Google Cloud Platform machine.

Table 1: The configuration of the Google Cloud Platform machine [3].

The name of the attribute in the configuration of the machine	The value of the attribute in the configuration of the machine
Region	Northamerica-northeast2, Toronto
Zone	northamerica-northeast2-a
Machine type	e2-standard-4
vCPU	4
Memory	16 GB
Disk size	100 GB
Image	Debian GNU/Linux 11, bullseye
Architecture	x86/64

The second step of the experimental methodology of the project could be to start the fuzzer on the Google Cloud Platform machine.

Table 2: The steps to start the fuzzer on the Google Cloud Platform machine.

Step number	Step	Definition of the step
1	sudo apt-get -y install wget	Install the program named wget on the Google Cloud Platform machine.
2	wget https://raw.githubusercontent.com/singh264/scripts/ece1776_final_report/ece1776_afl_tutorial.sh	Download the script that starts the AFL fuzzer on the Google Cloud Platform machine.
3	sudo apt-get -y install screen	Install the program named screen on the Google Cloud Platform machine.
4	screen -S <the_name_of_the_input_program>	Create the screen session named <the_name_of_the_input_program>.
5	mkdir <the_name_of_the_input_program>	Create the directory to save the results of the AFL fuzzer.
6	bash /home/user/ece1776_afl_tutorial.sh --input_program=<the_name_of_the_input_program> --directory_path=<the_name_of_the_input_program> --map_size_pow2=<the_size_of_the_bitmap> --llvm_mode --max_dict_file=<the_size_of_the_testcase>	Start the script that starts the AFL fuzzer and provide the name of the input program, the path of the directory that would include the results of the AFL fuzzer, the size of the bitmap, and the size of the test case of the AFL fuzzer.
7	Press control-a and d	Exit the screen session named <the_name_of_the_input_program>.
8	screen -r <the_name_of_the_input_program>	Enter the screen session named <the_name_of_the_input_program> after the AFL fuzzer completes the fuzzing.
9	Press control-c	Stop the AFL fuzzer and start to obtain the

		coverage results of the AFL fuzzer.

The script named ece1776_afl_tutorial.sh is located at https://github.com/singh264/scripts/blob/ece1776_final_report/ece1776_afl_tutorial.sh and the script named ece1776_afl_tutorial.sh was created after completing the steps that are detailed in the document named ECE1776 (2022F)-AFL Tutorial [4].

The script named ece1776_afl_tutorial.sh that is located at https://github.com/singh264/scripts/blob/ece1776_final_report/ece1776_afl_tutorial.sh includes the AFL fuzzer that was modified to create the file with the information about the statistic in the coverage of the AFL fuzzer that is located at https://github.com/singh264/AFL.

The file with the information about the statistic in the coverage of the AFL fuzzer includes the unix timestamp that is associated with the statistic in the coverage of the AFL fuzzer, and the new line is added to the file when the AFL fuzzer modifies the statistic in the coverage of the AFL fuzzer.

The third step of the experimental methodology of the project could be to obtain the results of the fuzzer in the Google Cloud Platform machine.

Table 3: The step to obtain the results of the fuzzer in the Google Cloud Platform machine.

Step number	Step	Definition of the step
1	scp -r -i <google_cloud_platform_private_key> user@<google_cloud_platform_machine_external_ip_address>:<directory_path> <directory_path>	Obtain the results of the fuzzer in the Google Cloud Platform machine and provide the ssh private key of the ssh public key that is configured in the Google Cloud Platform project, the name of the user of the Google Cloud Platform machine, the external IP address of the Google Cloud Platform machine, and the home directory path of the Google Cloud Platform machine that includes the

		script named ece1776_afl_tutorial.sh in Table 2.

The fourth step of the experimental methodology of the project could be to display the results of the fuzzer in the Google Cloud Platform machine.

Table 4: The steps to display the results of the fuzzer in the Google Cloud Platform machine.

Step number	Step	Definition of the step
1	wget https://raw.githubusercontent.com/singh264/AFL/ece1776_final_report/scripts/script.py	Download the script that displays the results of the AFL fuzzer after completing the fuzzing of the AFL fuzzer.
2	python script.py	Start the script that displays the results of the AFL fuzzer after completing the fuzzing of the AFL fuzzer.

The script that displays the data of the results of the AFL fuzzer utilizes the library named matplotlib in order to create the plots that display the information about the statistics in the coverage of the AFL fuzzer.

Results

The results of the project that were obtained after completing the first experiment of the project and that includes the LLVM mode AFL coverage of the GNU coreutils program when the size of the bitmap is the default 64 kilobytes is in the following table:

Table 5: The LLVM mode AFL coverage of the GNU coreutils programs [5].

	GNU coreutils programs			
	base64	md5sum	uniq	who
Lines	0.2% (48 of 29630 lines)		0.4% (120 of 29670 lines)	11.4% (3823 of 33415 lines)
Functions	0.3% (4 of 1530 functions)		0.4% (6 of 1530 functions)	0.7% (11 of 1530 functions)
Paths total	137		63	5
Cycles done	2		0	28585
Runtime	28.0 hours		23.9 hours	24.2 hours

The results of the project that were obtained after completing the first

experiment of the project throughout the duration of approximately 24 hours is displayed in the following diagrams:

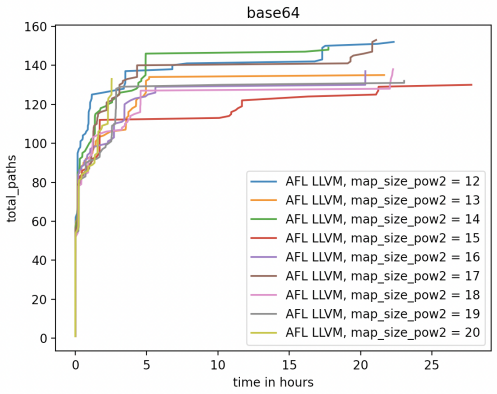

Diagram 1: The total paths in the input program named base64 [5].

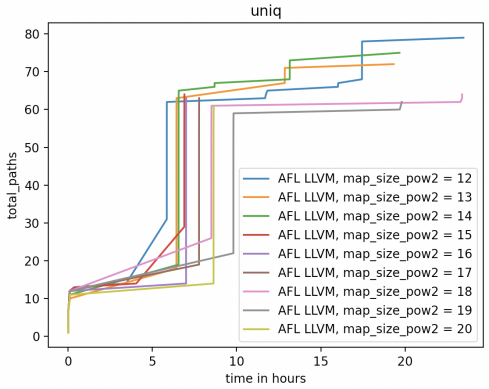

Diagram 2: The total paths in the input program named uniq [5].

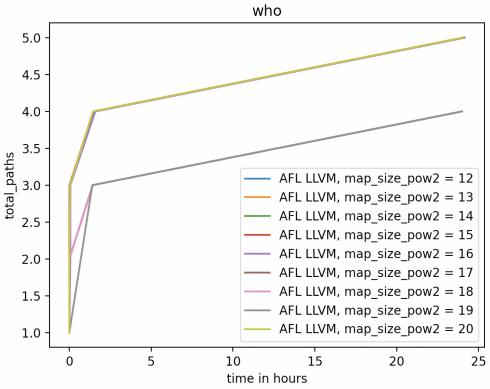

Diagram 3: The total paths in the input program named who [5].

The results of the project that were obtained after completing the second experiment of the project throughout the duration of approximately 24 hours is displayed in the following tables:

Table 6: The results of the project that were obtained after completing the second experiment of the project throughout the duration of approximately 24 hours [5].

	GNU coreutils programs			
	base64	md5sum	uniq	who
Good map_size_pow2	17		12	12

Table 7: Hash collisions of the edges in the control flow graph of the GNU Coreutils programs with the LLVM mode of the AFL fuzzer [5].

map_size_pow_2	Hash collisions of the edges in the control flow graph of the GNU Coreutils programs with the LLVM mode of the AFL fuzzer in percent			
	base64	md5sum	uniq	who
12	89.1		89.0	90.5
13	78.4		78.3	81.0
14	60.8		60.7	64.6
15	40.4		40.1	44.3
16	40.4		23.7	26.6
17	13.2		12.7	14.9
18	6.7		6.8	7.9
19	3.4		3.6	4.1
20	1.8		1.9	2.0

Lessons learned

The expectation after completing the first experiment of the project is that the runtime performance of the LLVM mode of the AFL fuzzer would decrease when the size of the bitmap in the AFL fuzzer increases, and the runtime performance of the LLVM mode of the AFL fuzzer would increase when the size of the bitmap in the AFL fuzzer decreases.

The intuition of the first expectation could be found in the paper named CollAFL: Path Sensitive Fuzzing that indicates that increasing the size of the bitmap in the AFL fuzzer could decrease the memory that is available to efficiently complete the fuzzing, and the intuition of the first expectation could be found in source code of the AFL fuzzer that is located at https://github.com/singh264/AFL/blob/ece1776_final_report/config.h#L322.

The results of the project that were obtained after completing the second experiment of the project throughout the duration of approximately 24 hours in Table 6 indicates that the good size of the bitmap in the AFL fuzzer was less than the default size of the bitmap in the AFL fuzzer when the input program was the program named uniq and the program named who.

The expectation after completing the second experiment of the project is that the AFL fuzzer would find more program paths in the input program when the size of the bitmap in the AFL fuzzer increases, and the AFL fuzzer would find less program paths in the input program when the size of the bitmap in the AFL fuzzer decreases.

The intuition of the second expectation could be found in the paper named CollAFL: Path Sensitive Fuzzing that indicates that increasing the size of the bitmap in the AFL fuzzer decreases the hash collisions of the edges in the control flow graph of the input program, and decreasing the size of the bitmap in the AFL fuzzer increases the hash collisions of the edges in the control flow graph of the input program.

The results of the project that were obtained after completing the second experiment of the project throughout the duration of 24 hours in Table 7 indicates

that increasing the size of the bitmap in the AFL fuzzer decreases the hash collisions of the edges in the control flow graph of the input program, and decreasing the size of the bitmap in the AFL fuzzer increases the hash collisions of the edges in the control flow graph of the input program.

The results of the project that were obtained after completing the second experiment of the project throughout the duration of approximately 24 hours in Table 6 indicates that the good size of the bitmap in the AFL fuzzer was more than the default size of the bitmap in the AFL fuzzer when the input program was the program named base64.

Future work

The future work could include the task to obtain the results of the LLVM mode of the AFL
fuzzer with the input program named md5sum, and the information about the task to obtain the results of the LLVM mode of the AFL fuzzer with the program named md5sum could be at https://drive.google.com/drive/folders/1rdywKjZ5DR-DGG7utyNDFaW-Dlb4Wj6e?usp=sharing.

References

[1] GNU coreutils 9.1. [Online]. Available: https://www.gnu.org/software/coreutils/manual/coreutils.html. [Accessed: 08-Oct-2022].

[2] Shuitao Gan, Chao Zhang, Xiaojun Qin, Xuwen Tu, Kang Li, Zhongyu Pei, and Zuoning Chen. Collafl: Path sensitive fuzzing. In 2018 IEEE Symposium on Security and Privacy (SP).

[3] “Free trial and free tier | google cloud,” Google. [Online]. Available: https://cloud.google.com/free/. [Accessed: 14-Oct-2022].

[4] “ECE1776 (2022F)- AFL Tutorial,” Google Drive. [Online]. Available: https://drive.google.com/file/d/1KQVwOageERsLpcOWtLg52g3hpy3X6kKC/view?usp=sharing. [Accessed: 04-Dec-2022].

[5] “Results,” Google Drive. [Online]. Available: https://drive.google.com/drive/folders/1_pl5RbHQOWKVKWMnQuix3dcUmklrETEj?usp=sharing. [Accessed: 10-Dec-2022].