УДК 62 **Технические науки**

Тронов Кирилл Александрович, *студент-бакалавр, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*
Белов Юрий Сергеевич, *к.ф. -м.н., доцент, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

ОПТИМИЗАЦИЯ ИНСТРУМЕНТАРИЯ AFL ДЛЯ ЛУЧШЕГО ПОКРЫТИЯ КОДА ПРИ РАБОТЕ СО СПЕЦИФИЧНЫМИ ДАННЫМИ

Аннотация: Фаззинг – это автоматическое тестирование, которое достигается путем выполнения множества программ со случайно сгенерированными входными данными с последующей проверкой состояния программы. Фаззинг считается успешным, если программа потерпела крах или была в другом некорректном состоянии. AFL (American fuzzy lop) – это современный фаззер, использующий новый тип инструментария времени компиляции и эволюционные алгоритмы. В данной статье рассматривается доработка инструментария afl для программы со специфическими входными данными, на которых фаззер показывает неудовлетворительное покрытие кода.

Ключевые слова: Фаззинг, afl, american fuzzy lop, анализ покрытия кода, автоматическое тестирование.

Annotation: Fuzzing is automatic testing that is accomplished by executing multiple programs with randomly generated input data and then checking the state of the program. Fuzzing is considered successful if the program crashed or was in another invalid state. AFL (American fuzzy lop) is a modern fuzzer using a new type of compile-time tooling and evolutionary algorithms. This article discusses the

refinement of the afl toolkit for a program with specific input data, on which the fuzzer shows unsatisfactory code coverage.

Keywords: Fuzzing, afl, american fuzzy lop, code coverage analysis, automated testing.

Введение. Идея фаззинга проста, но при этом очень эффективна. Ученые из Университета Висконсин-Мэдисон в 90-х годах протестировали 88 инструментов Unix, таких как grep, make и vi, с помощью инструмента, генерирующего случайные входные данные. В результате в более чем 25% протестированных программах были обнаружены ошибки. В 2013 году Михаил Залевски создал AFL(American fuzzy lop). American fuzzy lop - это фаззер , ориентированный на безопасность, который использует новый тип инструментария времени компиляции и эволюционные алгоритмы для автоматического обнаружения чистых, интересных тестовых случаев, которые запускают новые внутренние состояния в целевом двоичном файле [1]. Это существенно улучшает функциональное покрытие нечеткого кода. Тесты, полученные с помощью фаззинга, классифицируются как отрицательные [4]. Ошибки, которые можно найти с помощью отрицательных тестов (и, следовательно, фаззинга):

- Неправильное управление памятью (особенно в C / C ++) [3]
- Неправильная обработка нулей
- Плохая обработка исключений
- Бесконечные петли
- Неопределенное поведение
- Неправильное управление другими ресурсами

Исходные данные. В данном исследовании фаззинг использовался для тестирования программы, которая выполняет набор действий с образами, созданными приложениями резервного копирования (такими как EMC NetWorker или Symantec NetBackup). В программе реализовано 8 различных алгоритмов, и каждый алгоритм нужно было фаззить отдельно. Фаззинг

проводился после выполнения стандартной процедуры тестирования. Образы резервных копий, как правило, представляют собой большие файлы (более 100 МБ), в то время как AFL работает лучше всего, если входной файл короче нескольких килобайт. С другой стороны, приложение очень быстро работает для небольших вводов (более 500 запусков в секунду с использованием одного ядра).

Сначала каждый алгоритм был фаззирован с AFL с использованием отдельного ядра в течение 90 часов. Ошибок обнаружено не было, поэтому было решено измерить эффективность фаззинга с помощью afl-cov - инструмента, основанного на gcov, который вычисляет покрытие кода, сгенерированное во время фаззинга [2]. Результаты сильно различались для каждого алгоритма. На диаграмме ниже показано покрытие кода (рис. 1), сгенерированного, когда входной файл содержал только строку «базового ввода» (синий), и покрытие, сгенерированное всеми входными файлами, созданными AFL (красный):

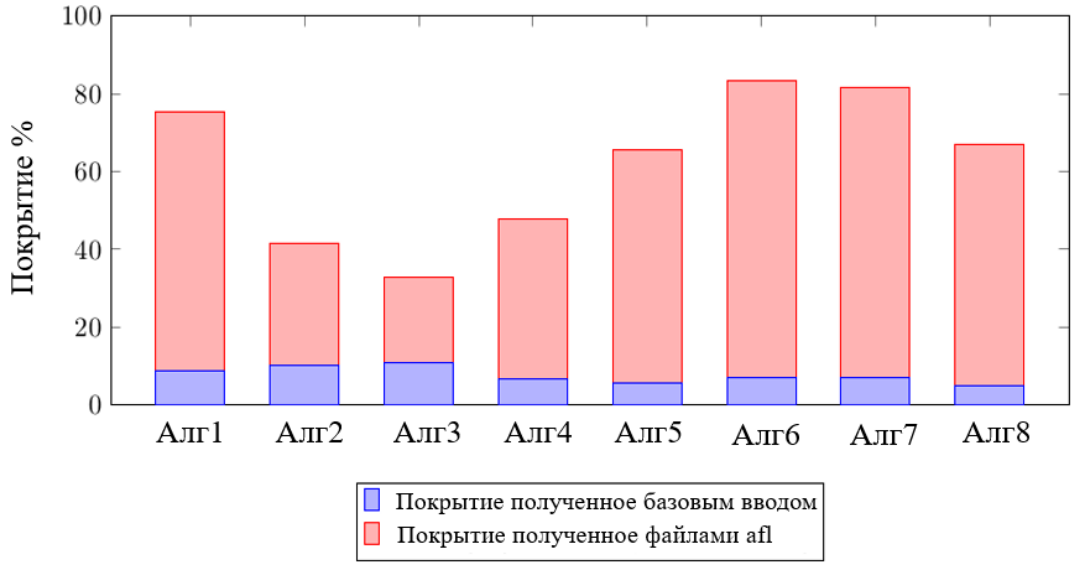

Рис. 1. Результаты покрытия кода для каждого алгоритма без модификаций afl

Покрытие кода, превышающее 80% для двух алгоритмов является хорошим результатом. Однако ошибок обнаружено не было, поэтому было решено проанализировать процесс фаззинга, чтобы понять, как этот процесс

можно улучшить. Проводить фаззинг дольше 90 часов не является целесообразным, т.к. afl перестает генерировать новые входные файлы (рис. 2).

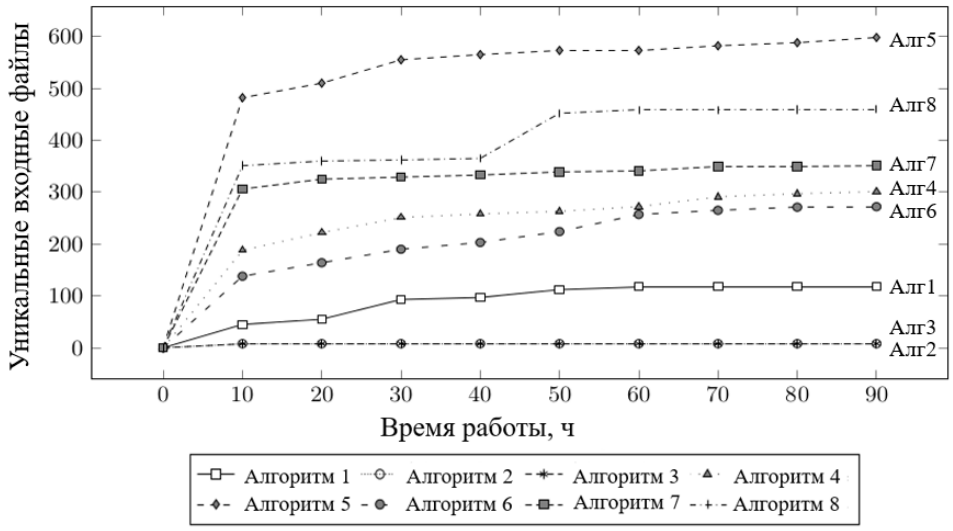

Рис. 2. График зависимости количества уникальных входных файлов от времени работы

Наихудший результат показали алгоритмы 2 и 3 (с наименьшим покрытием кода) в течение первых нескольких минут фаззинга было сгенерировано только несколько новых входных файлов.

Оптимизация инструментария. Чтобы покрыть незатронутый afl код, была изучена документация AFL, в частности особенности процесса фаззинга (среди прочего, как изменяются байты, как инструментарий обнаруживает новое поведение программы и как работает синхронизация во время параллельного фаззинга [5]). На основе полученных знаний после нескольких итераций фаззинга были внесены следующие улучшения:

- Созданы словари констант. У каждого алгоритма есть отдельный словарь.
- В AFL добавлена эвристика, которая пропускает детерминированный фаззинг для некоторых файлов размером более 50 КБ (до 8-кратного ускорения).
- Изменен механизм синхронизации параллельного фаззинга, чтобы избежать проблемы, которая приводит к неоптимальному детерминированному

фаззингу. Данное решение работает эффективно только тогда, когда фаззинг выполняется на одной машине.

- Увеличена степень детализации задач в параллельном фаззинге.
- Внесены изменения в тестируемую программу: отключена проверка входной контрольной суммы.

Окончательный фаззинг занял одну неделю (рис. 3). Фаззинг каждого алгоритма проводился на отдельной многоядерной системе. Покрытие кода для каждого алгоритма превысило 80% (рис. 4), что можно считать отличным результатом:

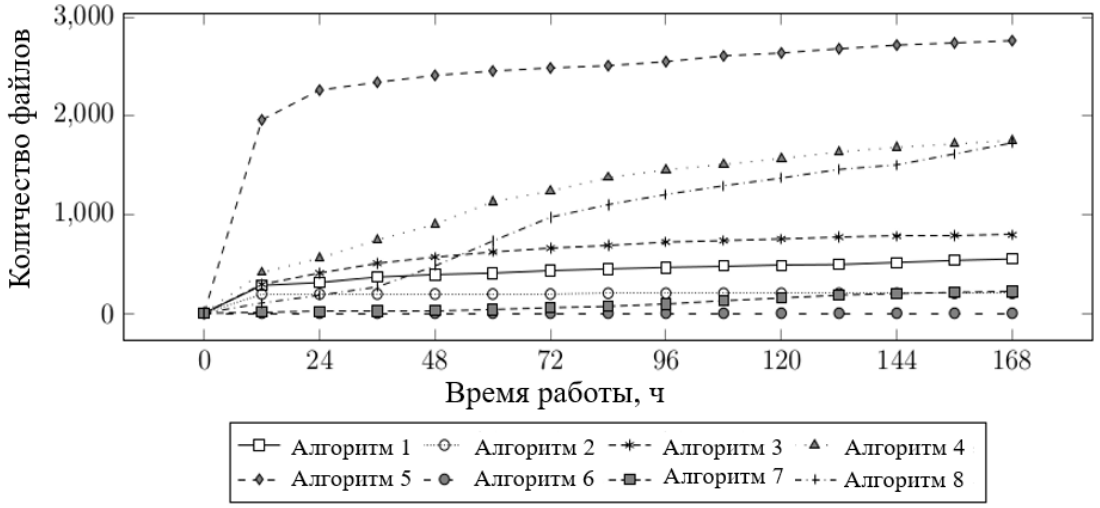

Рис. 3. Зависимость количества сгенерированных файлов модифицированным afl от времени работы

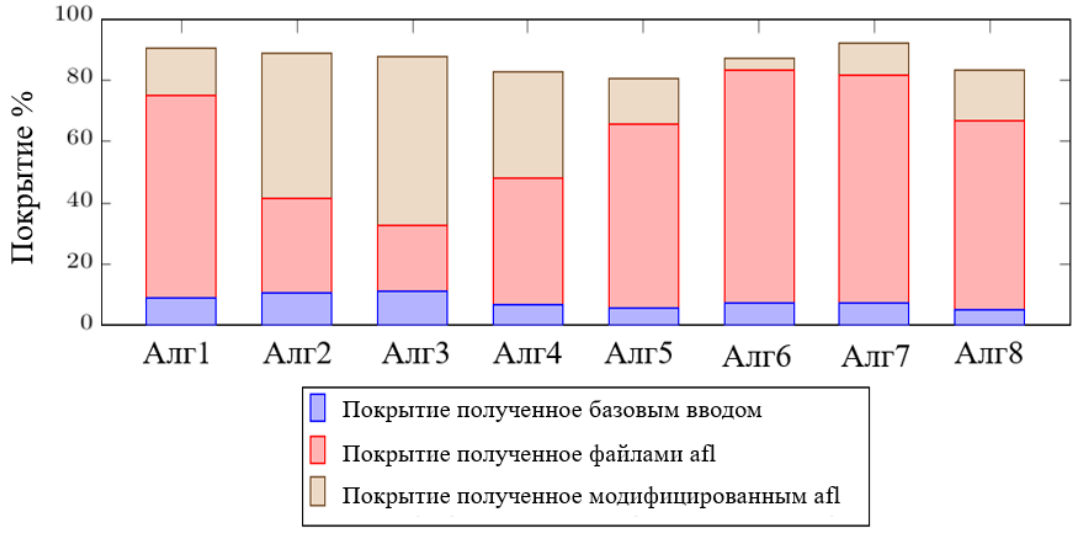

Рис. 4. Полученное покрытие кода модифицированным afl

Выводы: В этой статье были рассмотрены способы оптимизации инструментария afl для увеличения покрытия кода при работе с нестандартными входными данными, как широко используемые (словари), так и специфичные для конкретного случая (пропуск фазы детерминированного фаззинга для больших файлов).

Библиографический список:

1. M. Zalewski, AFL—American Fuzzy Lop (дата обращения: 01.05.2021) https://lcamtuf.coredump.cx/afl/.
2. N. Alshahwan, M. Harman, Coverage and Fault Detection of the Output-Uniqueness Test Selection Criteria // Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014), Jul 2014, P. 181-192.
3. P. Godefroid Fuzzing: Hack, Art and Science // Communications of the ACM, Feb 2020, vol. 63, no. 2, P. 70-76.
4. M. Polo, P. Reales, M. Piattini, C. Ebert Test automation // IEEE Softw., 2013, vol. 30, no. 1, P. 84–89.
5. SK. Cha, M. Woo, D. Brumley. Program-Adaptive Mutational Fuzzing // IEEE Symposium on Security and Privacy (S&P), 2015, P. 725-741.