Сборка производилась командой: AFL\_USE\_ASAN=1 CC=afl-clang-lto CXX=afl-clang-lto++ AR=llvm-ar RANLIB=llvm-ranlib AS=llvm-as CFLAGS="-O0 -g -fprofile-instr-generate -fcoverage-mapping" make fuzz\_harness

Фаззинг-тестирование запускалось командой: AFL\_USE\_ASAN=1 afl-fuzz -i in -o /home/ivan/afl\_outputs/out -m none -- ./fuzz\_harness



Рисунок 1 – Завершение фаззинг-тестирования

Далее с помощью скрипта был составлен txt-файл для удобства анализа ошибок – crash\_log.txt

Уязвимость типа «stack-buffer-overflow» была обнаружена при исполнении крашей с id 17, 22, 39, 44 и 50. Уязвимости были обнаружены в файле clever\_williams.c, первые четыре обнаружены на строке 278, последняя обнаружена на строке 284, на рисунке 2 отображён данный участок кода.

После проведения углублённого анализа стало ясно, что функция check\_bracket\_balance реализована некорректно: для валидации вложенности и сбалансированности скобок используется фиксированный массив, что приводит к уязвимости stack over flow при введении сообщения с большим числом скобок. Отсутствие динамической проверки баланса символов через стек также не позволяет гарантировать корректную парность символов.

Пример корректной реализации check\_bracket\_balance <https://www.geeksforgeeks.org/cpp-program-to-check-for-balanced-brackets-in-an-expression-well-formedness-using-stack/>

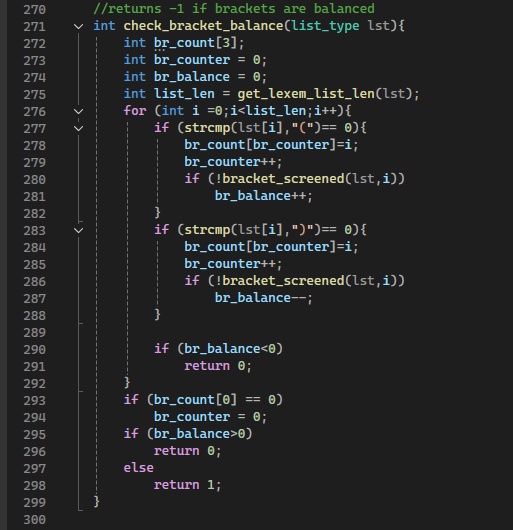


Рисунок 2 – Функция check\_bracket\_balance

Также уязвимость stack-buffer-overflow была обнаружена в файле kirch.c в строке 292, которая была вызвана бесконечной рекурсией (рисунок 3).

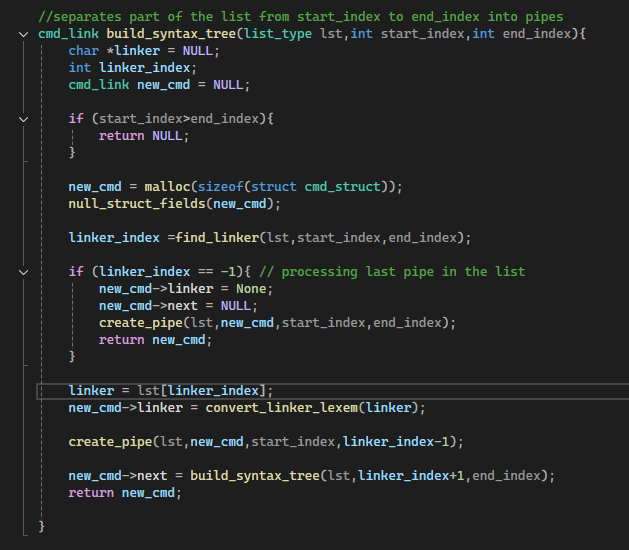


Рисунок 3 – Уязвимая функция к Buffer over flow в kirch.c

Отсутствие проверки границ между start\_index и linker\_index приводило к бесконечному вызову build\_syntax\_tree. Данное событие, в свою очередь, приводит к переполнению стека и аварийному завершению. Необходимо ввести проверку перед рекурсивным вызовом, например:

===========================================================

if (linker\_index + 1 <= end\_index) {

new\_cmd->next = build\_syntax\_tree(lst, linker\_index + 1, end\_index);

} else {

new\_cmd->next = NULL;

}

===========================================================

Это гарантирует, что рекурсивная функция не будет вызываться, если следующий фрагмент отсутствует, и тем самым предотвращает зацикливание и вылет при переполнении.

Значительное количество ошибок (более 20) было обнаружено в файле bhaskara.c в строке 143 (рисунок 4).

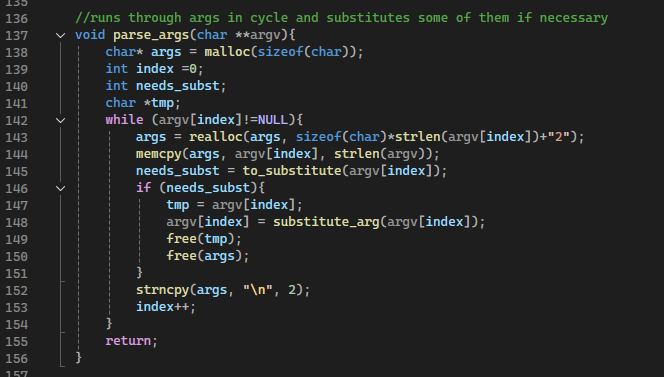


Рисунок 4 – Некорректная функция в файле bhaskara.c

Реализация указанной строки содержит логическую ошибку «2» - это строка, а не число. Для исправления ошибки, как было указано в промежуточном отчёте, необходимо указать, что добавляется число, а не строка: args = realloc(args, sizeof(char) \* (strlen(argv[index]) + 2)

Описание ошибки: некорректное использование функции realloc, приводящее к попытке выделения чрезмерно большого объёма памяти.

При более углублённом анализе было выявлено, что функция работает не совсем корректно. В строке 144 «memcpy(args, argv[index], strlen(argv));» argv представляет собой массив указателей на строки (массив аргументов командной строки). Однако функция strlen ожидает указатель на строку, а не на массив указателей. В результате фактически измерялась длина не строки-аргумента, а первого указателя в массиве, что приводило к некорректному поведению. Соответственно необходимо изменить строку на memcpy(args, argv[index], strlen(argv[index]));

Также было обнаружено множество ошибок, приводящих к уязвимости NULL Pointer Dereference, в файле bhaskara.c в строках 94 и 96 (рисунок 5).

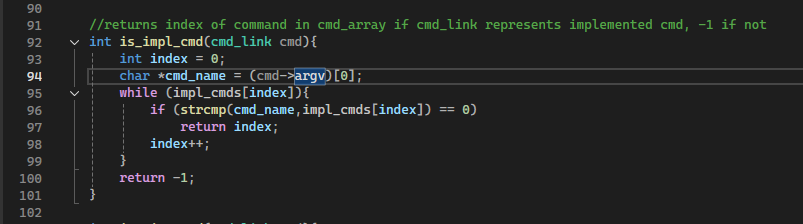


Рисунок 5 – Уязвимая функция в bhaskara.c

Ошибки в строках 94 и 96 файла bhaskara.c связаны с тем, что в функцию strcmp() передаётся нулевой указатель, что вызывает критическое завершение программы. Это происходит из-за отсутствия проверки валидности указателя cmd и его поля argv. Для исправления необходимо добавить предварительную проверку, чтобы убедиться в корректности данных перед вызовом strcmp() и избежать обращения к нулевому адресу.

Также подобная уязвимость была обнаружена в kirch.c в строке 40 (рисунок 6).

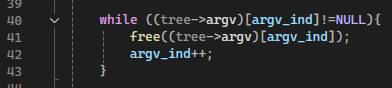


Рисунок 6 – Уязвимая функция в файле kirch.c

Для устранения уязвимости необходимо ввести проверку tree и tree->argv, чтобы они не были равны нулевому значению.