**ТЗ на оформление модельного кода для развития Structure-Aware Fuzzing**

**Цель:** получить компонент для экосистемы Crusher, позволяющий автоматизировать подготовку и выполнение процедуры Structure-Aware Fuzzing, при условии наличия исходных кодов.

**Задачи:**

- автоматизация порождения оберток для функций, принимающих на вход сложные наборы параметров;

- возможность структурно-корректного мутирования сэмплов, позволивших открыть новый путь;

- развитие метрик покрытия в направлении учета логических состояний тестируемого кода в дополнение к числу и порядку пройденных базовых блоков.

**Требования по оформлению кейса:**

10. должен быть представлен исходный код анализируемой функции и иных объектов, от которых данная функция зависит, а также ссылка на оригинальную программа полностью, с указанием модуля и строки откуда взят код для минимального теста;

15. выполнение исходного кода не должно зависеть от внешних медленных ресурсов;

20. исходный код должен быть написан на ЯП С/С++;

30. исходный код должен компилироваться компилятором clang(++)-12 и выполняться в usermode-режиме в ОС Ubuntu Linux 20+;

35. у кого есть хороший пример для Windows – пожалуйста напишите в чат по динамике в адрес Шамиля (@kursh), он подскажет нюансы по компилятору и т.п.

40. анализируемая функция либо должна вызываться без создания какого-либо дополнительного контекста, либо создание контекста должно быть:

- описано в виде конкретных переменных/команд, подлежащих созданию;

- независимо от внешних «медленных» ресурсов.

50. должен быть приведён make-файл, позволяющий скомпилировать приведённый код в статическую библиотеку;

60. в свободной форме должно быть дано описание того, что именно выполняет указанный участок кода (назначение кода), а также известные ограничения диапазонов входных переменных (контракт анализируемой функции);

70. в свободной форме должно быть дано описание того, что именно разработчик считает одной из координат вектора пространства состояний, описывающего состояние (state) в момент окончания выполнения анализируемой функции. К координатам вектора пространства состояний могут относиться:

- значения конкретных переменных (как локальных, так и глобальных по отношению к анализируемой функции);

- общий объем занятой памяти;

- число открытых дескрипторов, число открытых дескрипторов на запись;

- и т.п. (см. пример ниже)

**Пример (*абстрактный,* *с допущениями*):**

Назначение кода: в коде реализован диспетчер памяти, позволяющий выделять, очищать и перевыделять объем памяти.

Сборка кода: clang –static libmemory.c –o libmemory.a

Анализируемая функция:

*// commands – батч команд диспетчеру памяти*

*// zeroValue – символ которым следует занулять очищаемую память (в т.ч. при реаллоцировании сегмента)*

chunks \* **do\_mem**(sbatch \* commands, const char zeroValue)

*// структура описывает последовательность команд, которые будут вызваны*

struct **sbatch**

{

int count,

scommand \* commands

}

*//Структура описывает единичную команду на выделение памяти*

*//code: код команды диспетчерупамяти: 0 – выделение, 1 – очистка, 2 – перевыделение большего объема*

*//memchunk: при коде 0 будет создан новый указатель, при коде 1 – указатель будет очищен, а память возвращена в банк, при коде 2 – по существующему указателю будет выделен новый объем памяти*

*//len - при коде 1 значение может быть произвольным – должен очиститься ровно тот объем, которое ранее был выделен по данному указателю*

struct **scommand**

{

short code,

char \* memchunk,

int len

}

*//Структура содержит число всех имеющихся чанков памяти и их длины*

struct **chunks**

{

int \* chunk\_begin\_point,

int \* chunk\_length,

int count

}

Координаты вектора состояния (state):

* **число и порядок покрытых базовых блоков**;
* число чанков chunks.count;
* сумма длин всех чанков chunks.chunk\_length;
* отсутствие перекрытия всех чанков (проанализировать отсутствие перекрытия можно отдельной функцией);
* общий объем выделенной памяти;
* отсутствие в диапазонах памяти между чанками байт, отличающихся от байт со значением zeroValue;
* и т.п.

**Важно!** Аналитик самостоятельно определяет и фиксирует в описании, какие именно эвристики считать значимыми для вектора состояний, и предоставляет функции расчёта указанных эвристик. Функции могут как вычисляться после окончания анализируемой функции, так и подлежать встраиванию в анализируемую функцию.