Демонстрация: ошибка, найденная фаззером AFL++

Цель: показать студенту, как фаззер находит ошибку записи за пределы буфера в простой C++ программе и как программист исправляет такую ошибку.

> Формат входного файла в примере: первые 4 байта должны быть `FUZ1`, пятый байт задает длину имени, дальше идут байты имени.

1. В чем ошибка

Программа читает размер имени из файла:

```
uint8_t declared_size = data[4];
```

Но затем использует это значение как лимит записи в массив фиксированного размера:

```
char name[16] = {};
```

> **Ошибка:** файл может указать длину больше 16, например 90. Тогда цикл начинает писать в `name[16]`, `name[17]` и дальше, хотя допустимые индексы массива только `0..15`.

2. Исходный и исправленный код

Исходный вариант с ошибкой

```
uint8_t declared_size = data[4];
char name[16] = {};

// ОШИБКА: declared_size взят из
файла
// и не ограничен размером name.
for (uint8_t i = 0;
     i < declared_size && 5 + i <
data.size();
     ++i) {
    name[i] = static_cast<char>
(data[5 + i]);
}
```

Исправленный вариант

```
uint8_t declared_size = data[4];
char name[16] = {};

size_t available_size = data.size()
- 5;
size_t copy_size = std::min<size_t>
({
    declared_size,
    sizeof(name),
    available_size
});

for (size_t i = 0; i < copy_size;
++i) {
    name[i] = static_cast<char>
(data[5 + i]);
}
```

3. Почему проверка была неполной

В исходном коде условие цикла проверяло только чтение из входного файла:

```
5 + i < data.size()
```

Эта проверка отвечает на вопрос: "Есть ли еще байт во входных данных?"

Но она не отвечает на другой важный вопрос: "Есть ли еще место в массиве `name`?"

> Поэтому программа могла безопасно читать байты из файла, но небезопасно записывать их в маленький стековый массив.

4. Что делает исправление

Исправленный код считает безопасный размер копирования:

```
size_t copy_size = std::min<size_t>({
    declared_size,
    sizeof(name),
    available_size
});
```

Берется минимум из трех значений:

Значение	Что означает
`declared_size`	Сколько байт просит сам входной файл.

Значение	Что означает
`sizeof(name)`	Сколько байт реально помещается в буфер назначения.
`available_size`	Сколько байт реально есть во входных данных после заголовка.

> После исправления файл все еще может указать длину 90 или 255, но программа скопирует максимум 16 байт, потому что размер записи ограничен размером `name`.

5. Как фаззер показывает проблему

Сборка с AFL++ и AddressSanitizer:

```
AFL_USE_ASAN=1 afl-clang-fast++ -g -O1 src/main.cpp -o build/demo_afl_asan
```

Запуск фаззера:

```
AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 AFL_SKIP_CPUFREQ=1 \
afl-fuzz -i in -o out_afl_asan_demo -- build/demo_afl_asan @@
```

Воспроизведение найденного падения:

```
build/demo_afl_asan out_afl_asan_demo/default/crashes/id*
```

Типичный результат:

```
ERROR: AddressSanitizer: stack-buffer-overflow
WRITE of size 1
src/main.cpp:29:17 in parse_demo_format
```

6. Главный вывод

> Данные из файла, сети или пользователя нельзя использовать как размер записи без проверки. При копировании всегда надо учитывать оба ограничения: сколько данных есть в источнике и сколько места есть в буфере назначения.

Учебный пример предназначен для демонстрации фаззинга собственного кода в изолированной среде.