

Практикум: фаззинг C++ программы с AFL++ на Windows 10

Цель занятия: показать полный цикл фаззинг-тестирования: подготовка среды, написание простой уязвимой программы, инструментированная сборка, запуск AFL++, анализ найденного падения и минимальное исправление.

Границы применения. Материал предназначен для учебного стенда и собственного кода. Не запускайте фаззинг на чужих продуктах, сетевых сервисах или рабочих системах без разрешения владельца и изоляции среды.

Проверено на стенде. Все команды методички прогнаны на WSL2 с Ubuntu 24.04.4 LTS (afl-fuzz++4.09c, clang 18.1.3, g++ 13.3.0). Учебный проект `my-first-python-project` в архиве уже содержит `src/main.cpp`, seed-файлы в `in/` и примеры каталогов результатов `out*/`. В публичный архив включены исходники; бинарники необходимо собрать командами из разделов 6 и 7.

1. Что понадобится на ПК слушателя

- Windows 10 версии 2004 или новее, сборка 19041 или новее.
- Права администратора для первичной установки WSL.
- Включенная виртуализация в BIOS/UEFI, если WSL2 не стартует.
- Visual Studio Code для Windows.
- Интернет для загрузки Ubuntu, пакетов и расширений VS Code.

AFL++ ориентирован на Unix/Linux-среду. На Windows 10 самый простой учебный вариант - использовать WSL2 с Ubuntu, а VS Code подключить к проекту внутри Linux через расширение WSL.

2. Установка WSL2 и Ubuntu

2.1. Установка из PowerShell

Откройте **PowerShell** от имени администратора и выполните:

```
wsl --install -d Ubuntu
```

Перезагрузите компьютер, если установщик попросит это сделать. После первого запуска Ubuntu создайте Linux-пользователя и пароль.

Проверьте версию WSL:

```
wsl -l -v
```

В строке Ubuntu желательно видеть `VERSION 2`. Если указана версия 1:

```
wsl --set-version Ubuntu 2
```

2.2. Базовое обновление Ubuntu

Откройте приложение Ubuntu из меню Пуск и выполните:

```
sudo apt update
sudo apt upgrade -y
```

3. Настройка VS Code для WSL

1. Установите Visual Studio Code для Windows.
2. В VS Code откройте Extensions и установите расширение **WSL** или пакет **Remote Development**.
3. В терминале Ubuntu создайте папку проекта:

```
mkdir -p ~/fuzzing-lab/aflpp-demo
cd ~/fuzzing-lab/aflpp-demo
code .
```

После команды `code .` VS Code откроет папку уже внутри WSL. В левом нижнем углу окна должен быть индикатор вида `WSL: Ubuntu`.

Храните учебный проект в файловой системе WSL, например в `~/fuzzing-lab`, а не в `/mnt/c/...`. Так сборка и фаззинг обычно работают заметно быстрее.

4. Установка AFL++

Вариант А: быстрый вариант для занятия

Для демонстрации обычно достаточно пакета из репозитория Ubuntu:

```
sudo apt update
sudo apt install -y afl++ clang make g++ build-essential unzip
```

Проверьте установку:

```
afl-fuzz --version
which afl-clang-fast++
```

Используйте именно `--version`. Короткий ключ `-V` в AFL++ означает «фаззить заданное число секунд» и требует аргумента: команда `afl-fuzz -V` без числа выдаст `option requires an argument -- 'V'` и напечатает справку — это не ошибка установки.

Пакет `unzip` нужен, чтобы распаковать архив с примером. Если архив распаковывается на стороне Windows (проводник, 7-Zip, WinRAR), этот пакет не требуется.

Вариант В: сборка AFL++ из исходников

Этот способ ближе к официальной документации AFL++ и полезен, если пакет Ubuntu отсутствует или сильно устарел:

```
sudo apt update
sudo apt install -y build-essential python3-dev automake cmake git flex bison \
  libglib2.0-dev libpixmap-1-dev python3-setuptools cargo libgtk-3-dev \
  lld llvm llvm-dev clang ninja-build cpio libcapstone-dev wget curl python3-pip

git clone https://github.com/AFLplusplus/AFLplusplus.git
cd AFLplusplus
make source-only
sudo make install
```

Проверка:

```
afl-fuzz -V
afl-cc -hh
```

5. Создание простой C++ программы

В VS Code создайте папку и файл `src/main.cpp`:

```
mkdir -p src in out build
code src/main.cpp
```

Вставьте код:

```

#include <cstdint>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

static std::vector<uint8_t> read_file(const std::string& path) {
    std::ifstream file(path, std::ios::binary);
    return std::vector<uint8_t>(
        std::istreambuf_iterator<char>(file),
        std::istreambuf_iterator<char>()
    );
}

static int parse_demo_format(const std::vector<uint8_t>& data) {
    if (data.size() < 5) {
        return 0;
    }

    if (data[0] != 'F' || data[1] != 'U' || data[2] != 'Z' || data[3] != '1') {
        return 0;
    }

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

    // Учебная ошибка: declared_size не проверяется относительно размера name.
    for (uint8_t i = 0; i < declared_size && 5 + i < data.size(); ++i) {
        name[i] = static_cast<char>(data[5 + i]);
    }

    if (name[0] == 'A' && name[1] == 'F' && name[2] == 'L') {
        std::cout << "interesting input\n";
    }

    return 0;
}

int main(int argc, char** argv) {
    if (argc != 2) {
        std::cerr << "usage: " << argv[0] << " <file>\n";
        return 1;
    }

    auto data = read_file(argv[1]);
    return parse_demo_format(data);
}

```

Программа читает файл, проверяет простую сигнатуру `FUZ1`, затем копирует указанное в файле количество байт в буфер на 16 байт. Ошибка сделана специально: если поле длины больше 16, возможна запись за границы массива.

6. Первичная сборка и ручная проверка

Соберите обычным компилятором, чтобы убедиться, что код компилируется:

```

g++ -std=c++17 -g -O1 src/main.cpp -o build/demo
printf 'FUZ1\x03ABC' > in/seed1
./build/demo in/seed1

```

Ожидаемый результат: программа завершается без ошибки. Сообщение `interesting input` может появиться, если первые символы имени равны `AFL`.

7. Инструментированная сборка для AFL++

Для надежной демонстрации ошибки используем AddressSanitizer вместе с инструментированным компилятором AFL++:

```
AFL_USE_ASAN=1 afl-clang-fast++ -std=c++17 -g -O1 -fno-omit-frame-pointer \
src/main.cpp -o build/fuzz_target
```

Ручная проверка аварийного входа:

```
printf 'FUZ1\x20AAAAAAAAAAAAAAAAAAAAAAAAAAAA' > /tmp/crash.bin
./build/fuzz_target /tmp/crash.bin
```

С большой вероятностью AddressSanitizer покажет ошибку записи за границы стека. Это подтверждает, что дефект достижим.

8. Запуск AFL++

Создайте начальный корпус. Для AFL++ достаточно одного-двух маленьких валидных примеров:

```
printf 'FUZ1\x01A' > in/seed1
printf 'hello' > in/seed2
```

Запустите фаззинг:

```
ASAN_OPTIONS=abort_on_error=1:symbolize=0 AFL_SKIP_CPUFREQ=1 \
AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1 \
afl-fuzz -i in -o out -m none -- ./build/fuzz_target @@
```

Про WSL2 и `core_pattern`. Проверьте значение:

```
cat /proc/sys/kernel/core_pattern
```

В части сборок WSL там стоит `|/wsl-capture-crash %t %E %p %s` — ядро направляет core dump во внешнюю утилиту. Увидев символ `|` в начале шаблона, AFL++ прерывает работу: `[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'`. В этом случае добавьте в команду запуска `AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1` (в примере выше она уже есть). Если в файле стоит `core`, переменная не требуется — но и не мешает: AFL++ лишь отметит, что она включена.

Альтернатива (нужны права root, сбрасывается после перезапуска WSL):

```
sudo sh -c 'echo core > /proc/sys/kernel/core_pattern'
```

Параметр	Назначение
<code>-i in</code>	Каталог с начальными входными файлами.
<code>-o out</code>	Каталог, куда AFL++ пишет очередь, статистику и найденные падения.

<code>-m none</code>	Отключает лимит памяти AFL++; удобно при запуске с ASAN.
<code>@@</code>	Место, куда AFL++ подставляет путь к текущему тестовому файлу.
<code>AFL_SKIP_CPUFREQ=1</code>	Для учебного стенда пропускает предупреждение о governor CPU.
<code>AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1</code>	Нужна, если <code>/proc/sys/kernel/core_pattern</code> в WSL начинается с <code> </code> : обходит проверку, из-за которой AFL++ прерывается с сообщением <code>Pipe at the beginning of 'core_pattern'</code> .

Остановить фаззинг можно сочетанием `Ctrl+C`. Для демонстрационной программы падение обычно находится быстро.

9. Где смотреть результат

Основные артефакты AFL++ находятся в каталоге `out/default`:

```
ls out/default
ls out/default/crashes
cat out/default/fuzzer_stats
```

Если найдено падение, имя файла будет начинаться с `id:`. Воспроизведите его:

```
./build/fuzz_target out/default/crashes/id:000000*
```

Если shell не подставляет имя из-за специальных символов, скопируйте точное имя файла из вывода `ls out/default/crashes` и передайте его программе в кавычках.

10. Исправление ошибки

В учебной программе ошибка находится в цикле копирования. Минимальное исправление - ограничить копирование размером буфера:

```
#include <algorithm> // для std::min

uint8_t declared_size = data[4];
char name[16] = {};
size_t copy_size = std::min<size_t>(declared_size, sizeof(name));

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

Директиву `#include <algorithm>` следует добавить в начало файла: `std::min` объявлена именно там. На практике код собирается и без неё за счёт транзитивных включений, но полагаться на это не нужно.

После исправления пересоберите цель и снова запустите найденный crash-файл. Падение должно исчезнуть:

```
AFL_USE_ASAN=1 afl-clang-fast++ -std=c++17 -g -O1 -fno-omit-frame-pointer \
src/main.cpp -o build/fuzz_target

./build/fuzz_target out/default/crashes/id:000000*
```

11. Мини-сценарий для преподавателя

1. Показать, что программа принимает нормальный seed-файл.
2. Пояснить идею покрытия: AFL++ сохраняет мутации, которые открывают новые ветки исполнения.
3. Собрать цель через `afl-clang-fast++`, чтобы вставить instrumentation.
4. Запустить `afl-fuzz` и обратить внимание на поля `cycles done`, `total paths`, `unique crashes`.
5. Воспроизвести crash-файл как обычный входной файл программы.
6. Исправить проверку границ и показать, что crash больше не воспроизводится.

12. Типовые проблемы

Симптом	Что проверить
<code>ws1 --install</code> не работает	Версию Windows 10: нужна 2004 / build 19041 или новее. На старых версиях используйте ручную установку WSL из документации Microsoft.
Команда <code>code .</code> не найдена	Переустановить VS Code с опцией добавления в PATH или запустить VS Code и открыть папку через Remote - WSL.
<code>afl-clang-fast++</code> не найден	Проверить установку <code>afl++</code> или выполнить сборку из исходников. Команда <code>which afl-clang-fast++</code> должна вернуть путь.
AFL++ ругается на лимит памяти при ASAN	Запускать с <code>-m none</code> .
Фаззинг идет очень медленно	Проект должен лежать внутри WSL, например в <code>~/fuzzing-lab</code> , а не на Windows-диске <code>/mnt/c</code> .
Нет падений	Проверить, что цель собрана с <code>AFL_USE_ASAN=1</code> , seed начинается с <code>FUZ1</code> , а запуск использует <code>@@</code> .
PROGRAM ABORT: Pipe at the beginning of 'core_pattern'	Проверьте <code>cat /proc/sys/kernel/core_pattern</code> : в WSL шаблон может начинаться с <code> </code> (передача core dump внешней утилите). Добавьте <code>AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1</code> в команду запуска (см. раздел 8).
<code>unzip: command not found</code>	Установите пакет: <code>sudo apt install -y unzip</code> . Либо распакуйте архив средствами Windows.
<code>option requires an argument -- 'V'</code> при проверке установки	Так ведёт себя <code>afl-fuzz -V</code> без аргумента. Для проверки версии используйте <code>afl-fuzz --version</code> .

Permission denied при запуске
`./build/fuzz_target`

Файл потерял бит исполнения (например, при распаковке архива сторонним распаковщиком). Выполните `chmod +x build/fuzz_target` или соберите цель заново — бинарники в архиве собраны на другой машине и всё равно подлежат пересборке.

13. Контрольные вопросы

1. Зачем AFL++ нужна инструментированная сборка?
2. Чем seed corpus отличается от найденной очереди `queue` ?
3. Почему `@@` нужен программам, которые читают вход из файла?
4. Почему ASAN полезен при демонстрации ошибок памяти?
5. Почему crash-файл нужно воспроизводить отдельно от AFL++?

14. Источники

- AFL++ Install: <https://aflplus.plus/docs/install/>
- AFL++ Fuzzing in Depth: https://aflplus.plus/docs/fuzzing_in_depth/
- AFL++ Quick Start Guide: <https://aflplus.plus/docs/quickstartguide/>
- Microsoft Learn, Install WSL: <https://learn.microsoft.com/windows/wsl/install>
- Microsoft Learn, VS Code with WSL: <https://learn.microsoft.com/windows/wsl/tutorials/wsl-vscode>