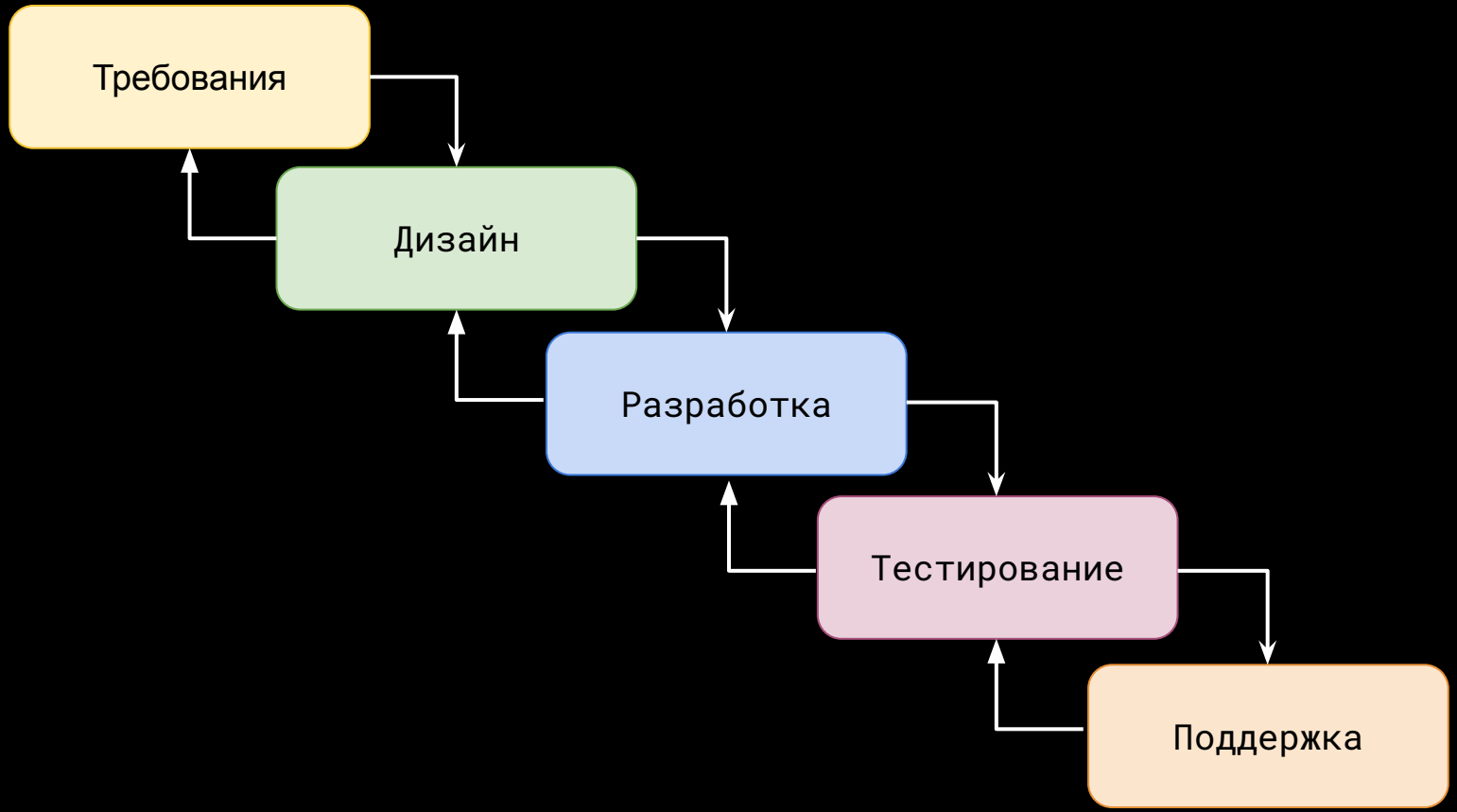
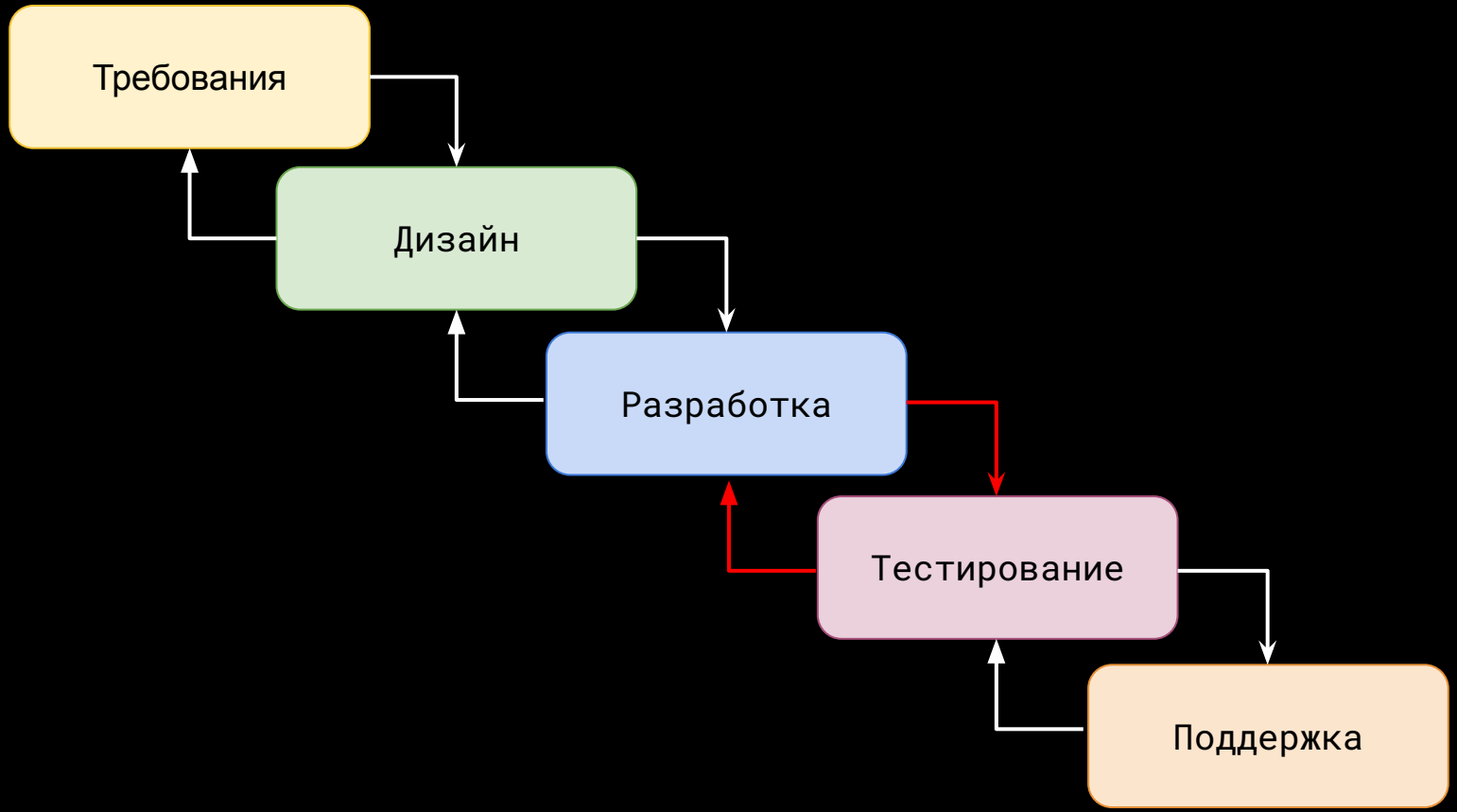
Фаззинг - основы

Дополнительные главы практической безопасности

30.03.2021


Требования
Дизайн
Разработка
Тестирование
Поддержка

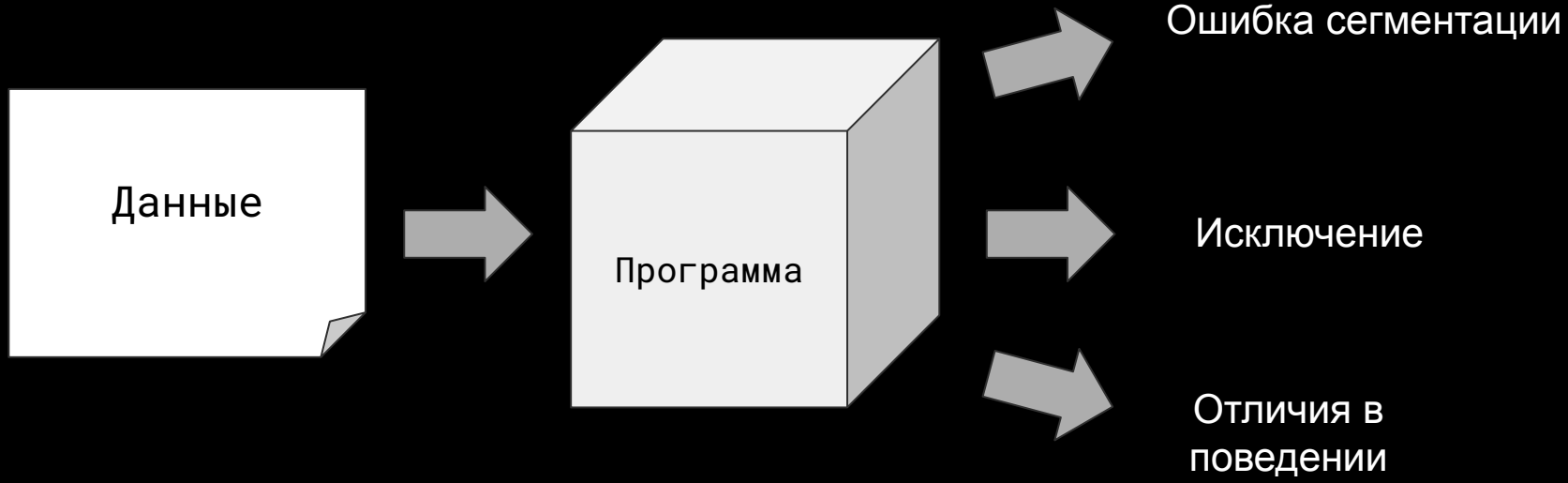

Требования
Дизайн
Разработка
Тестирование
Поддержка

Загадка! Сколько “ошибок” в коде?

```
int fd;
char *buffer;
int flags = O_WRONLY|O_APPEND|O_CREAT;

if ((fd = open(“data.log”, flags, 0666)) == 0) exit(1);

buffer = malloc(u32size + 1);
read(0, buffer, u32size);

write(fd, buffer, u32size);
...
```

На размышление даётся 30 секунд 👴

Цель: обнаружение нестандартного поведения



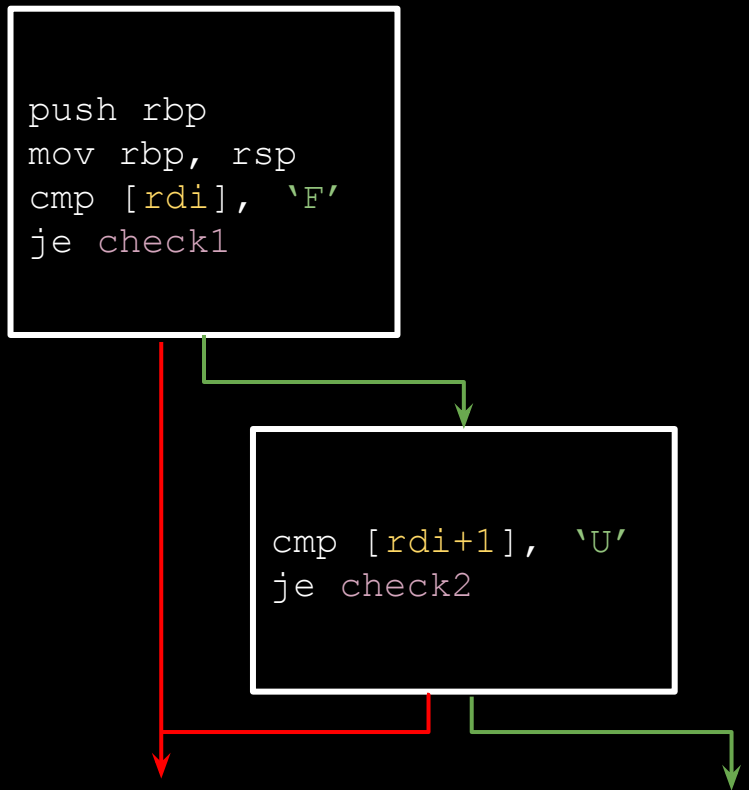
Очень простой фаззер

```
dd if=/dev/urandom of=input.sample bs=1 count=1024
cat input.sample | /usr/bin/prog

/usr/bin/prog -f input.sample
```

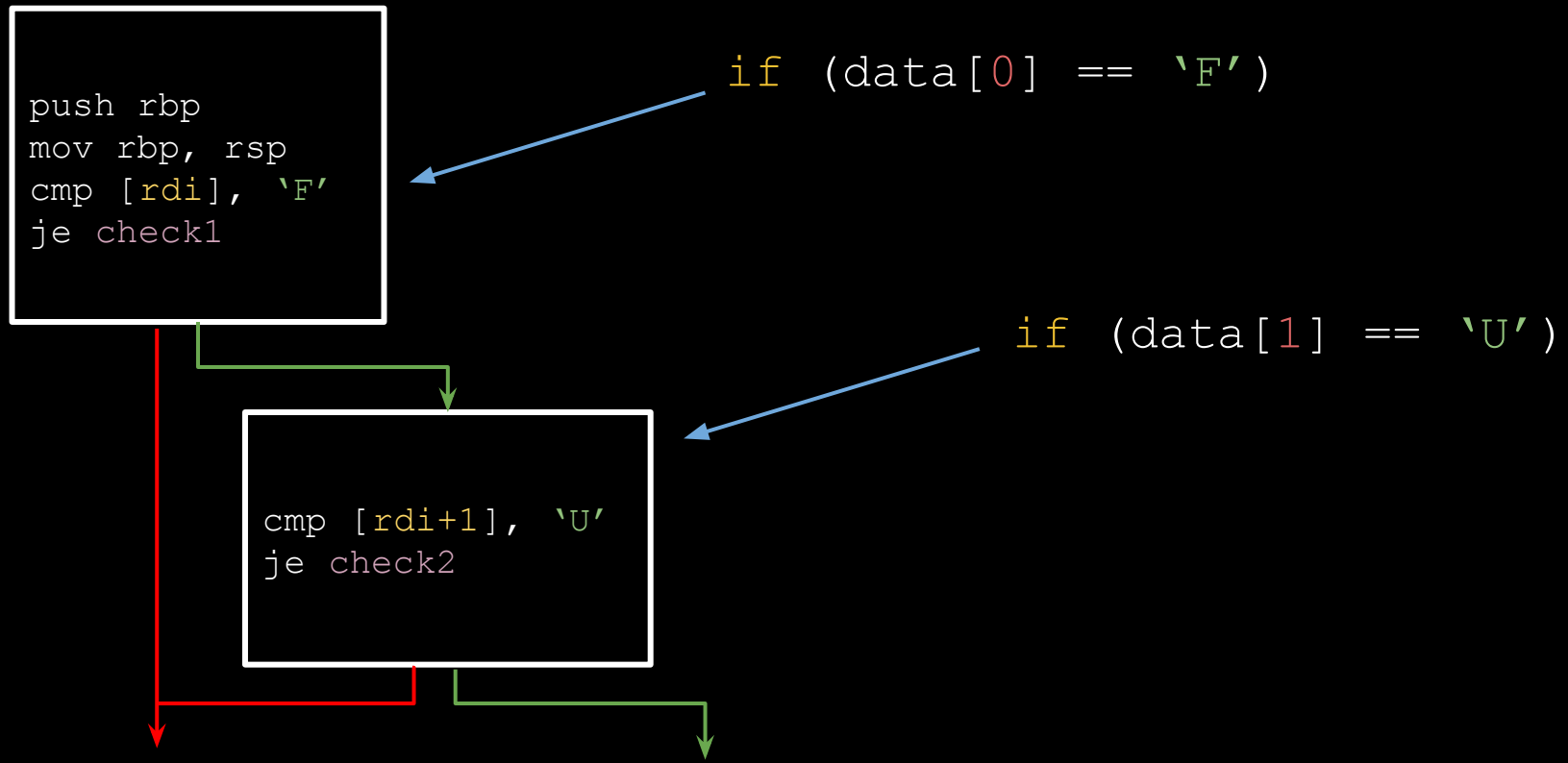
Стратегии создания данных

- генерация
- мутация

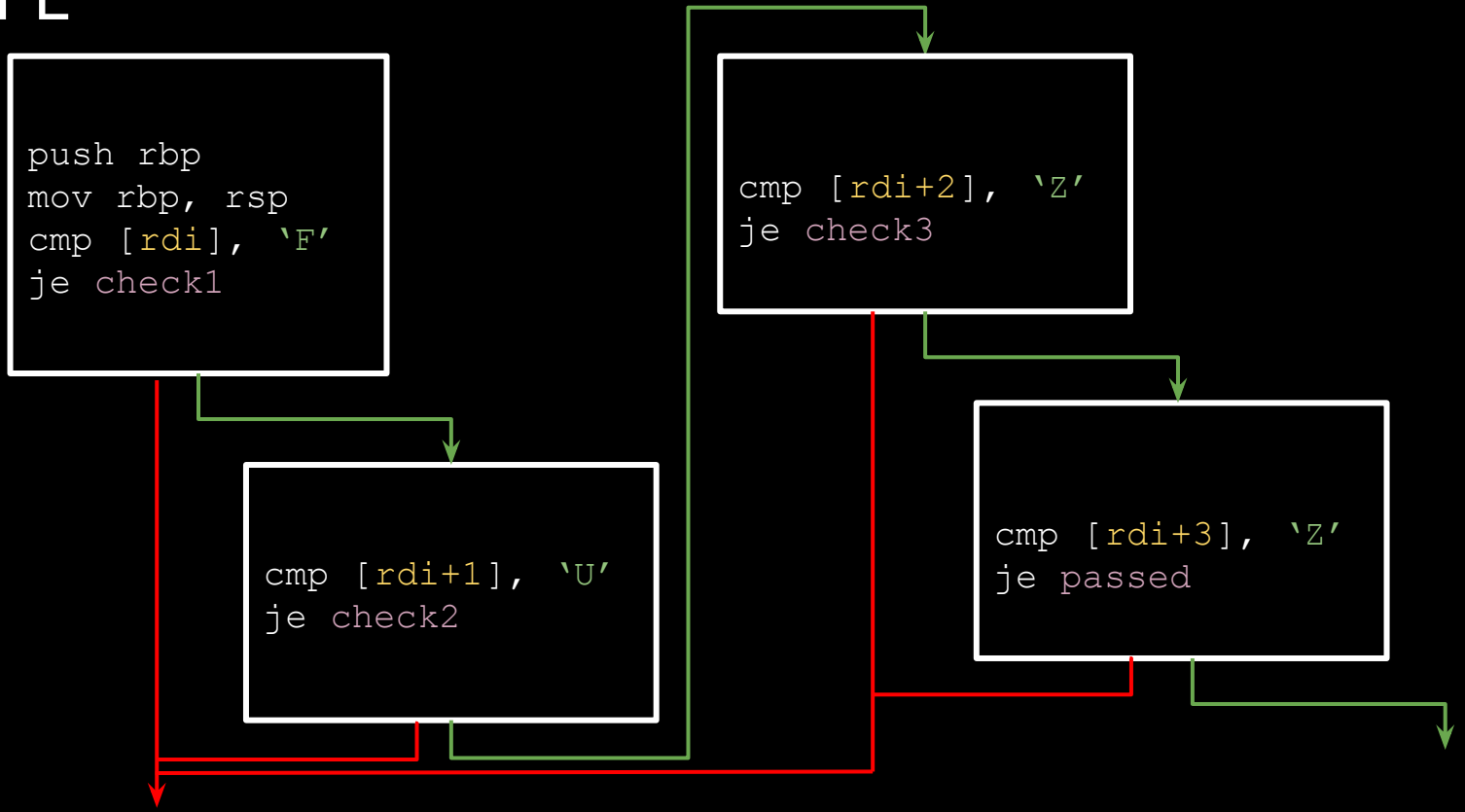
```
void check_fuzz(char *data, size_t size)
{
    if (data[0] == 'F')
        if (data[1] == 'U')
            if (data[2] == 'Z')
                if (data[3] == 'Z')
                    __builtin_trap();
}
```


push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
cmp [rdi+1], 'U'
je check2

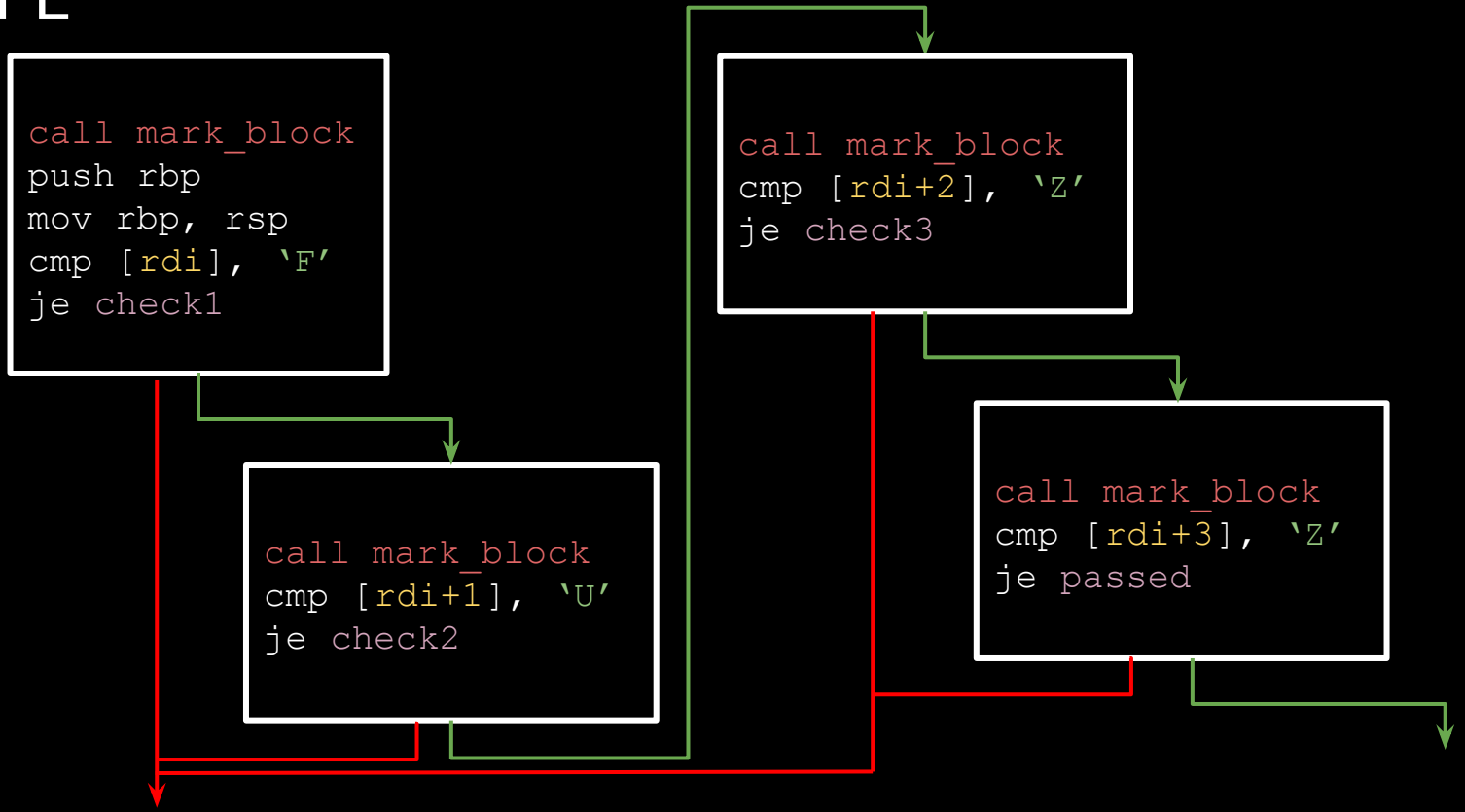
AFL

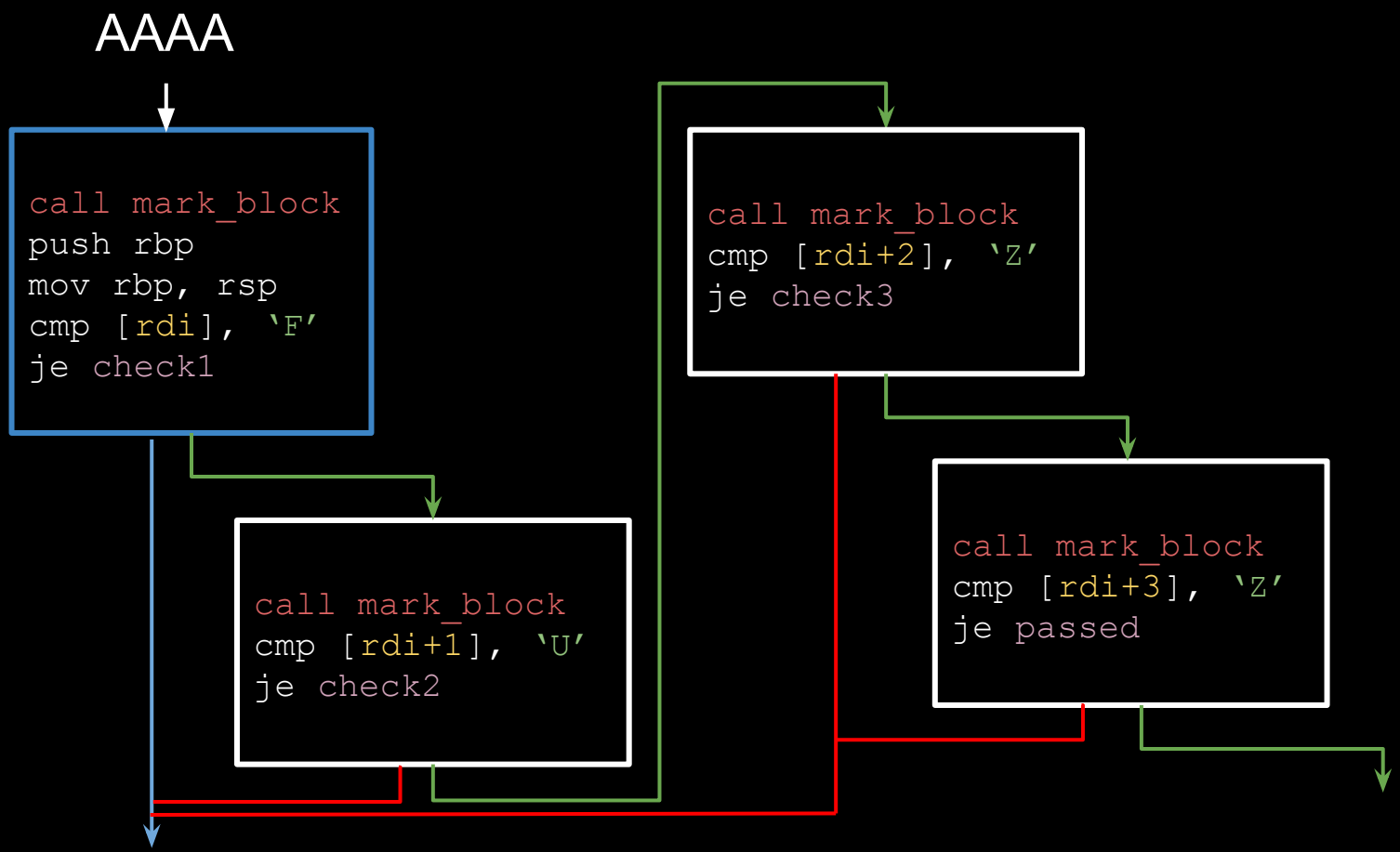

push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
if (data[0] == 'F')
if (data[1] == 'U')
cmp [rdi+1], 'U'
je check2

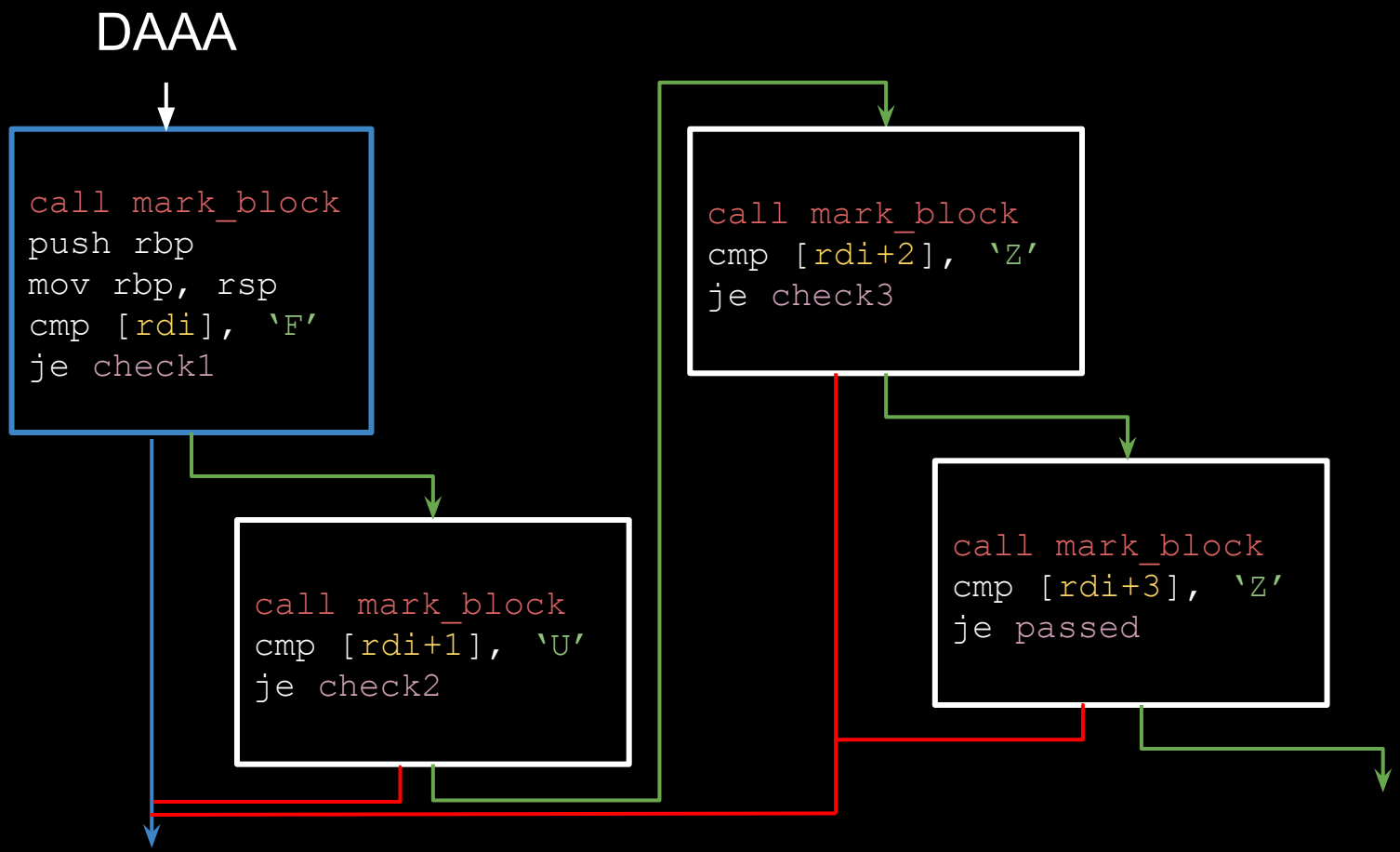
AFL

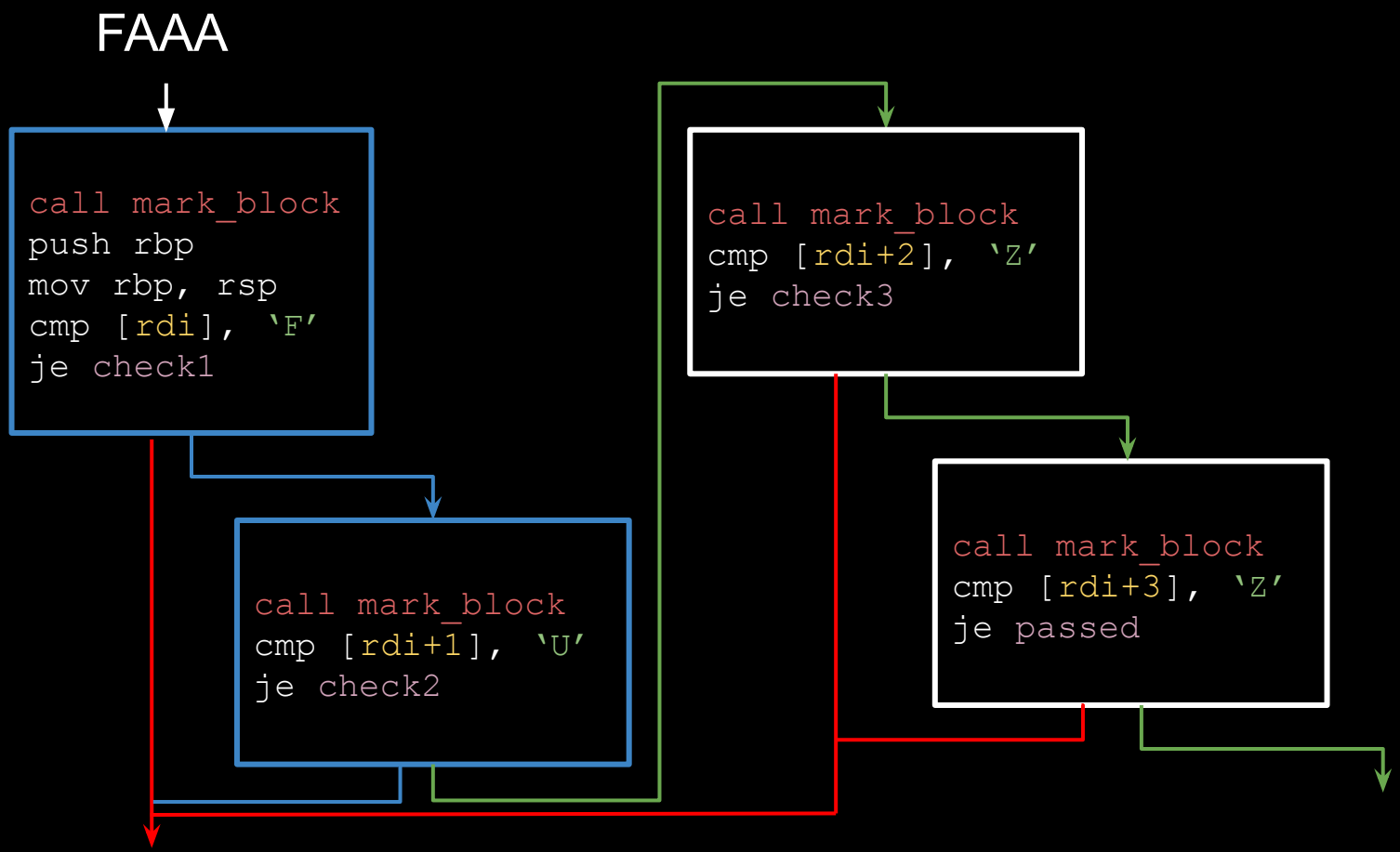

push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
cmp [rdi+1], 'U'
je check2
cmp [rdi+2], 'Z'
je check3
cmp [rdi+3], 'Z'
je passed

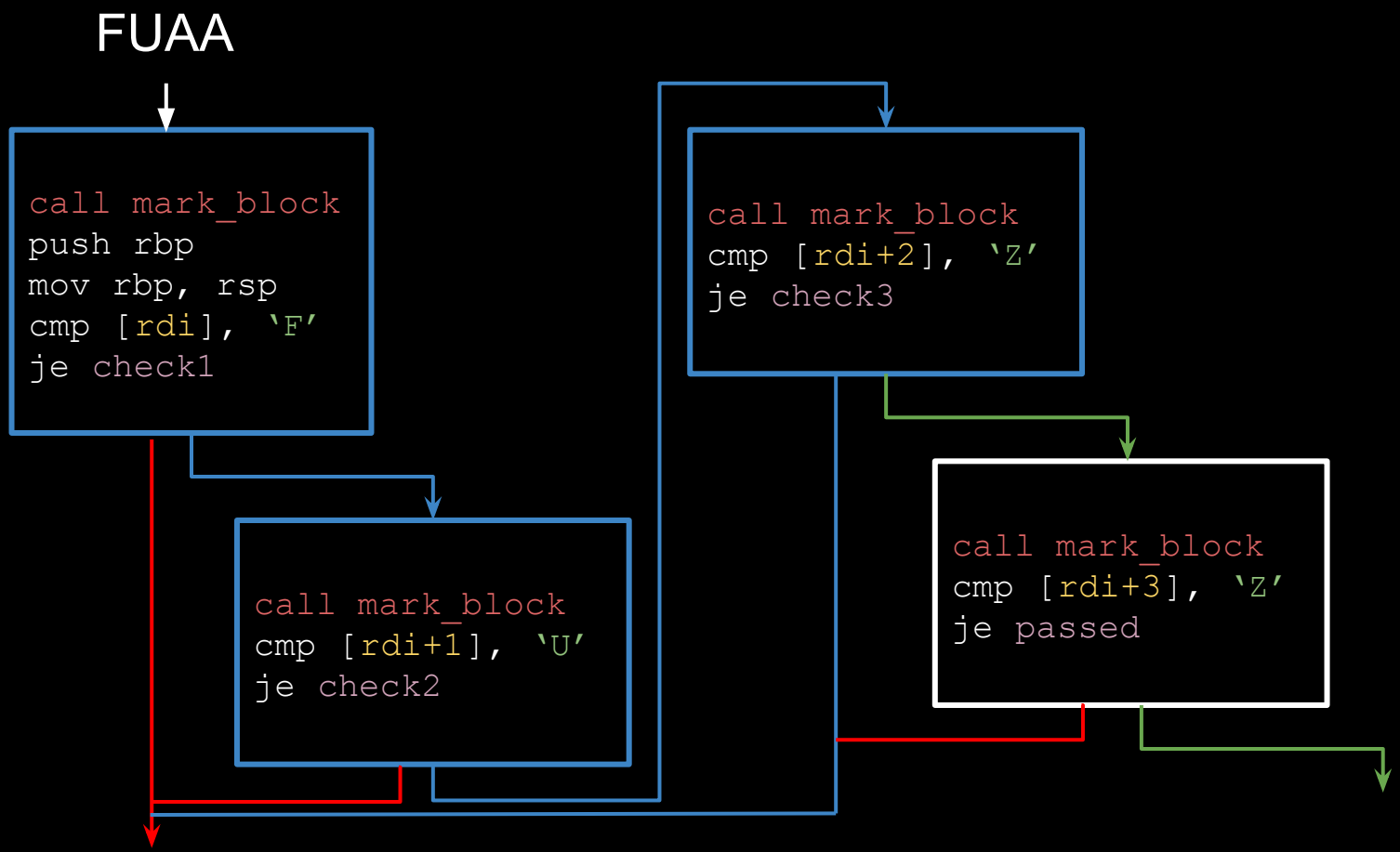
AFL

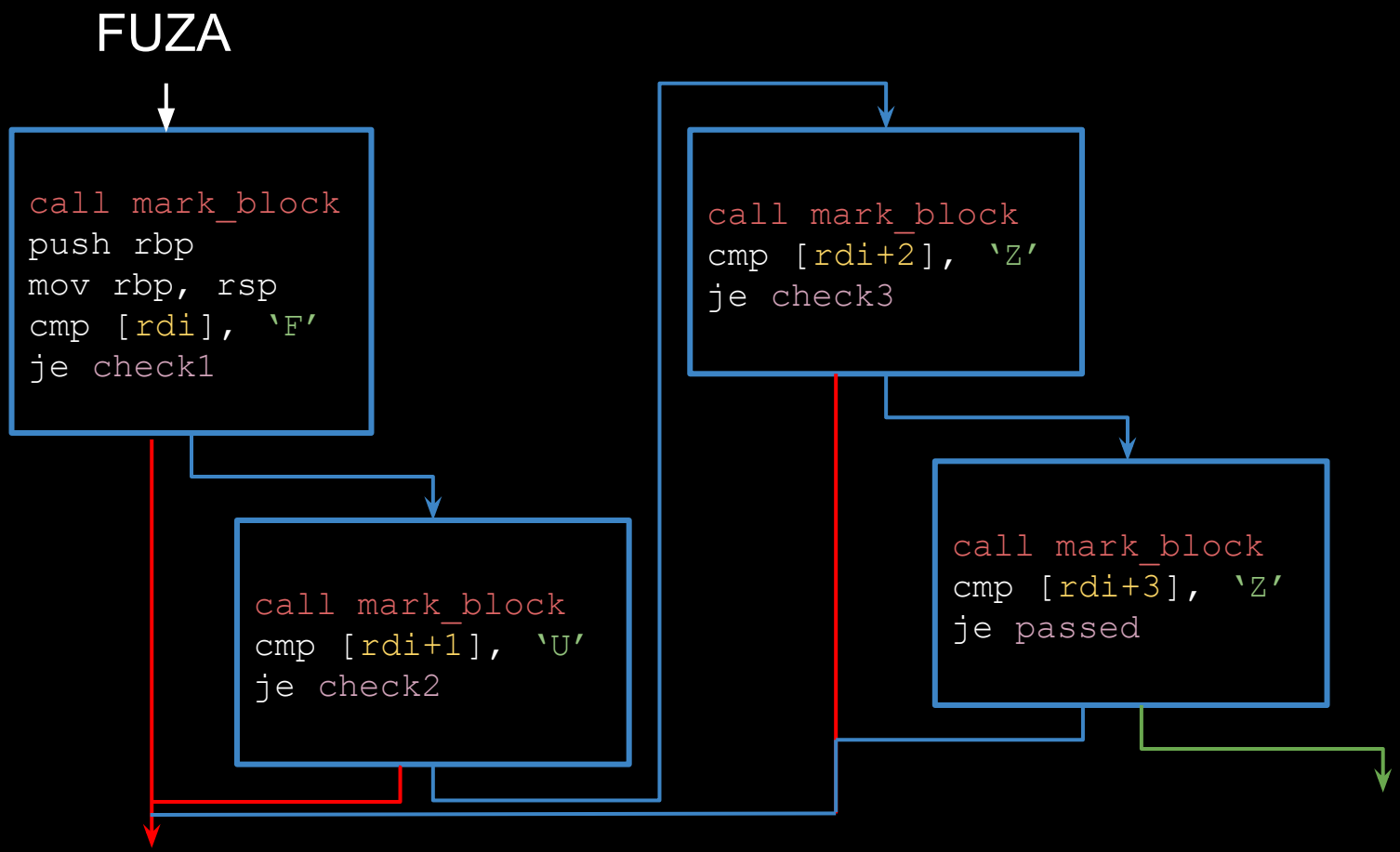

call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed

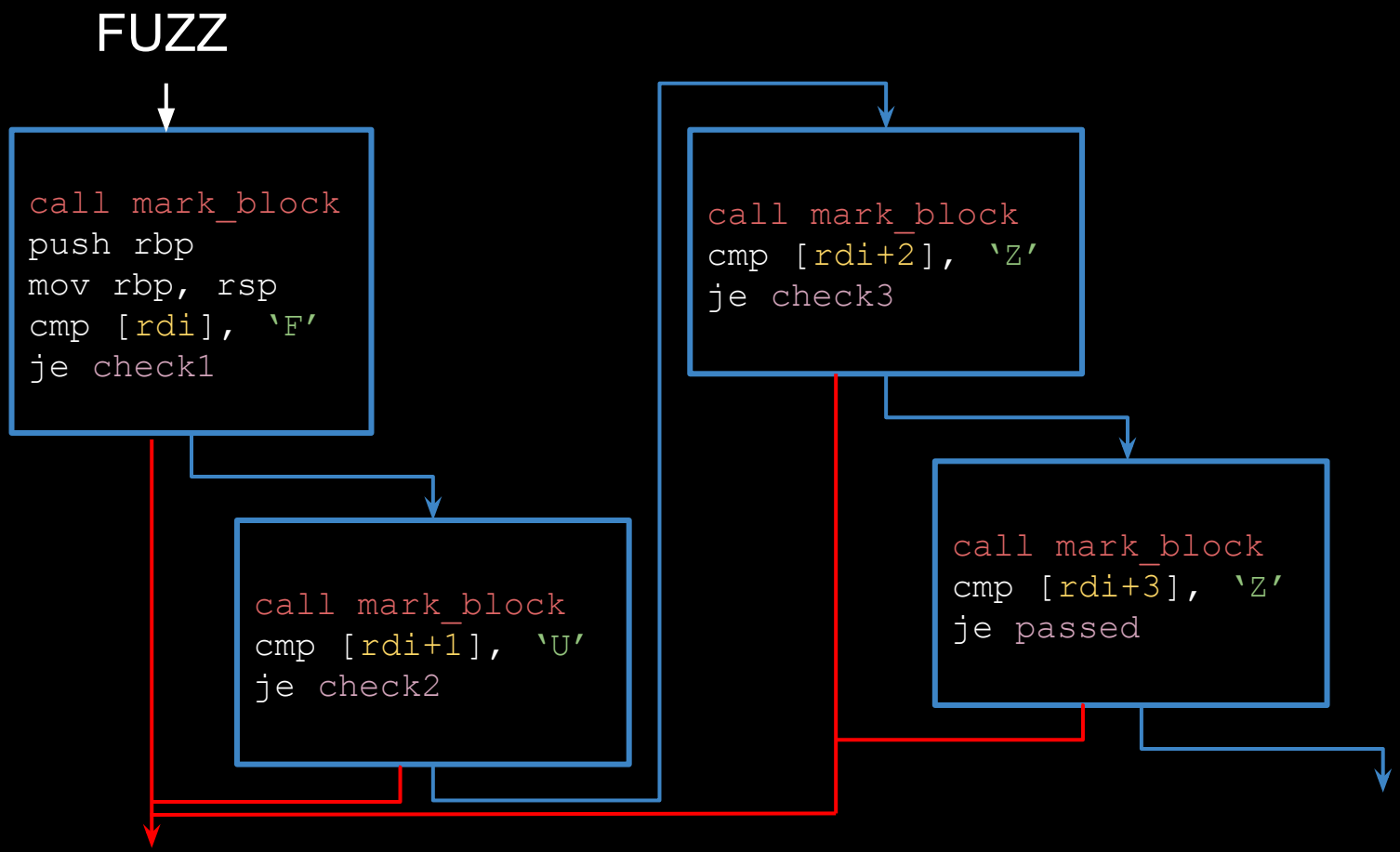

AAAA
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed


DAAA
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed

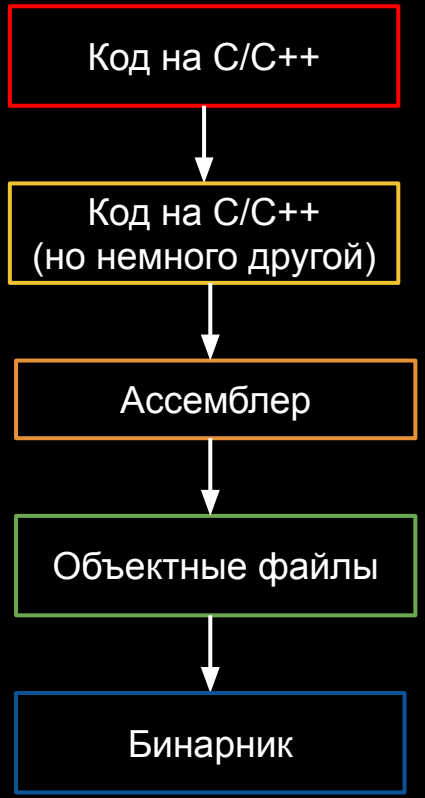

FAAA
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed


FUAA
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed


FUZA
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed

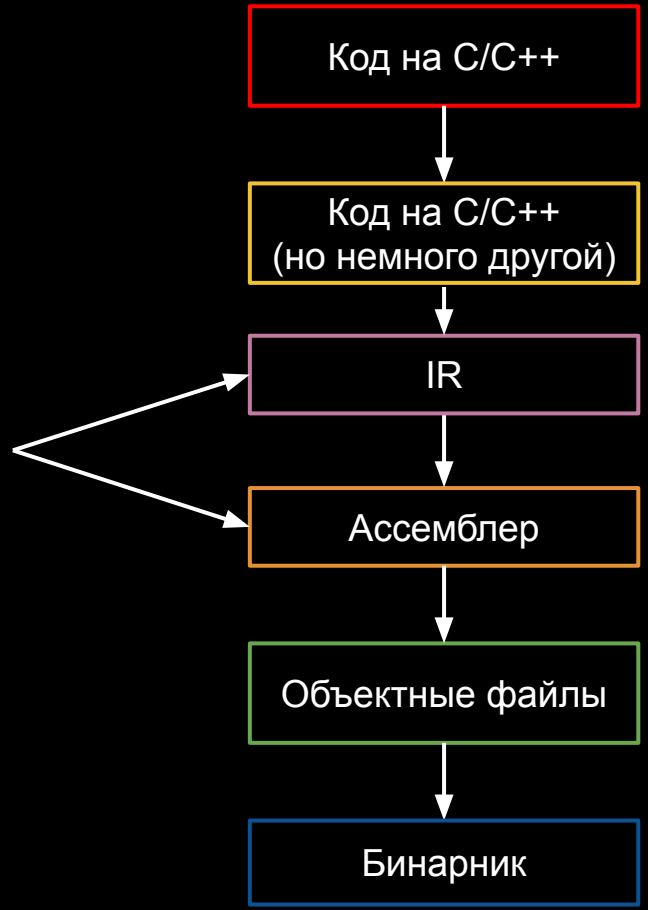

FUZZ
call mark_block
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
call mark_block
cmp [rdi+1], 'U'
je check2
call mark_block
cmp [rdi+2], 'Z'
je check3
call mark_block
cmp [rdi+3], 'Z'
je passed

Стадии компиляции прака, который ты списал



Стадии компиляции прака, который ты списал

Стадии компиляции прака, который ты списал



Размечены на базовые блоки

Добавим “свои” инструкции перед каждой меткой

Больше информации:

`AFL/afl-as.c`

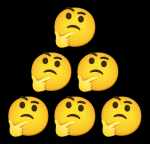
void check_fuzz()

```
push rbp
mov rbp, rsp
cmp [rdi], 'F'
je check1
```

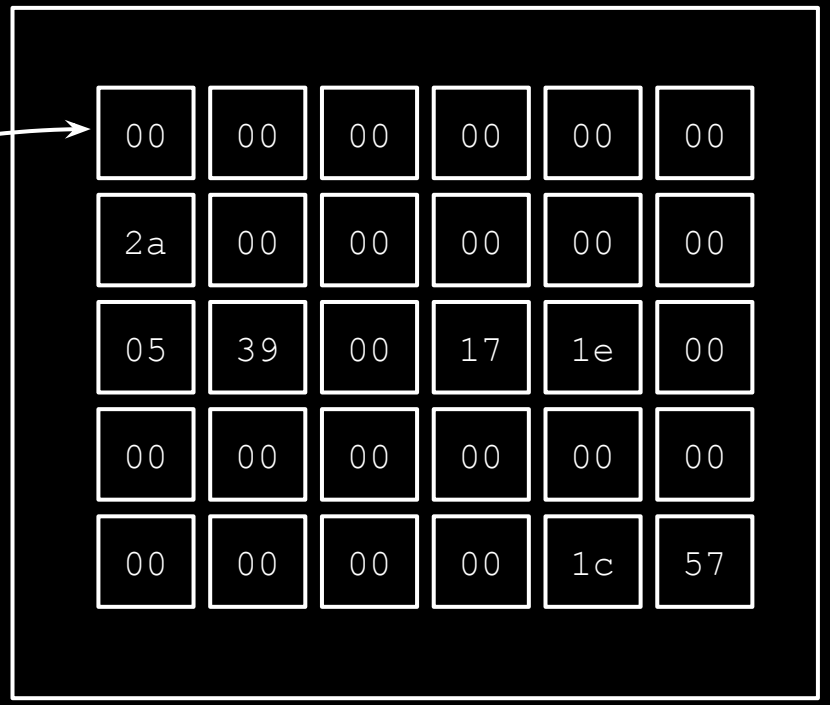
void check_buzz()

```
push rbp
mov rbp, rsp
cmp [rdi], 'B'
je check1
```

```
cmp [rdi+1], 'U'
je check2
```



MEM_SIZE = (1 << 16)

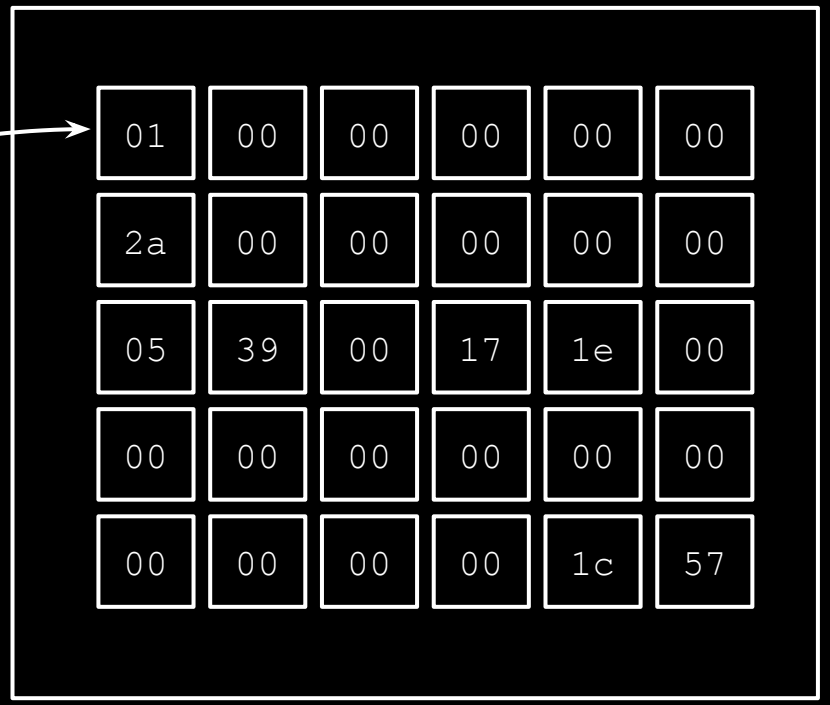



```
shared_mem[c1 ^ prev_location]++;
prev_location = c1;
```

A → B → C → D → E === AB BC CD DE
A → B → D → C → E === AB BD DC CE

Больше: AFL/docs/technical_details.txt

MEM_SIZE = (1 << 16)

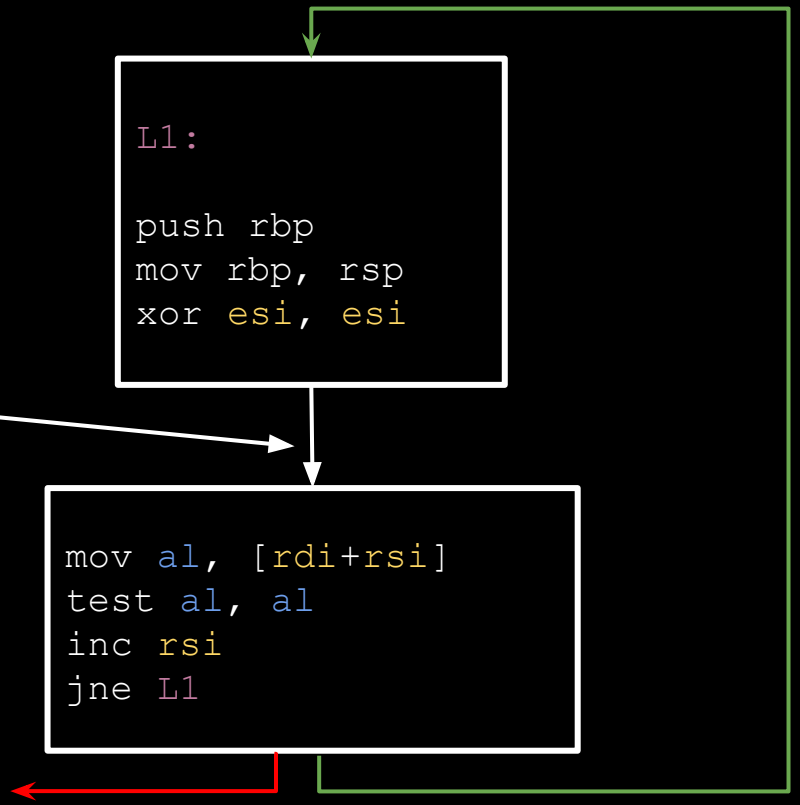



```
shared_mem[c1 ^ prev_location]++;
prev_location = c1;
```

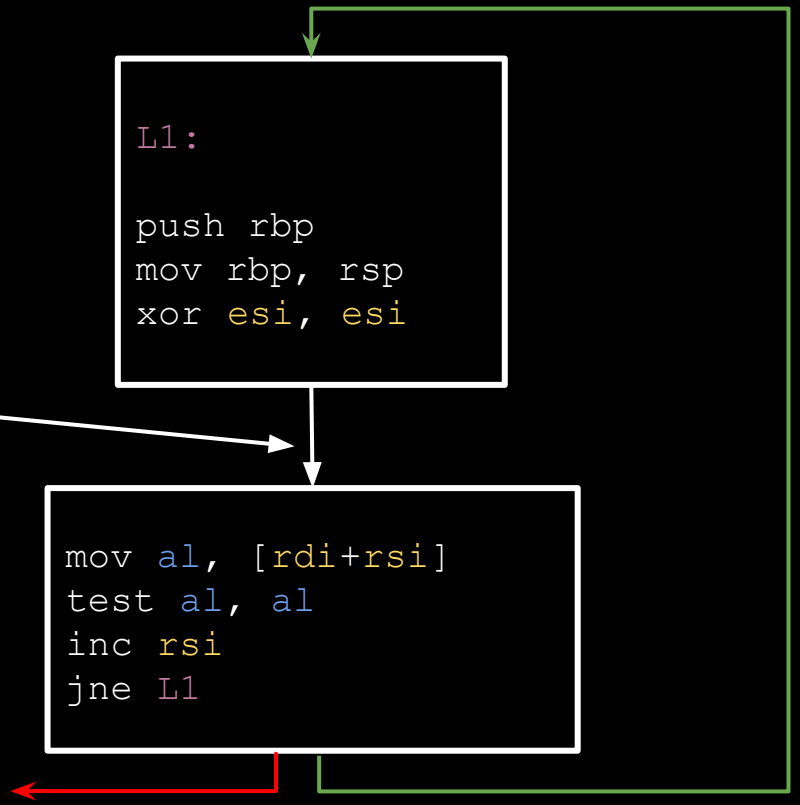
A → B → C → D → E === AB BC CD DE
A → B → D → C → E === AB BD DC CE

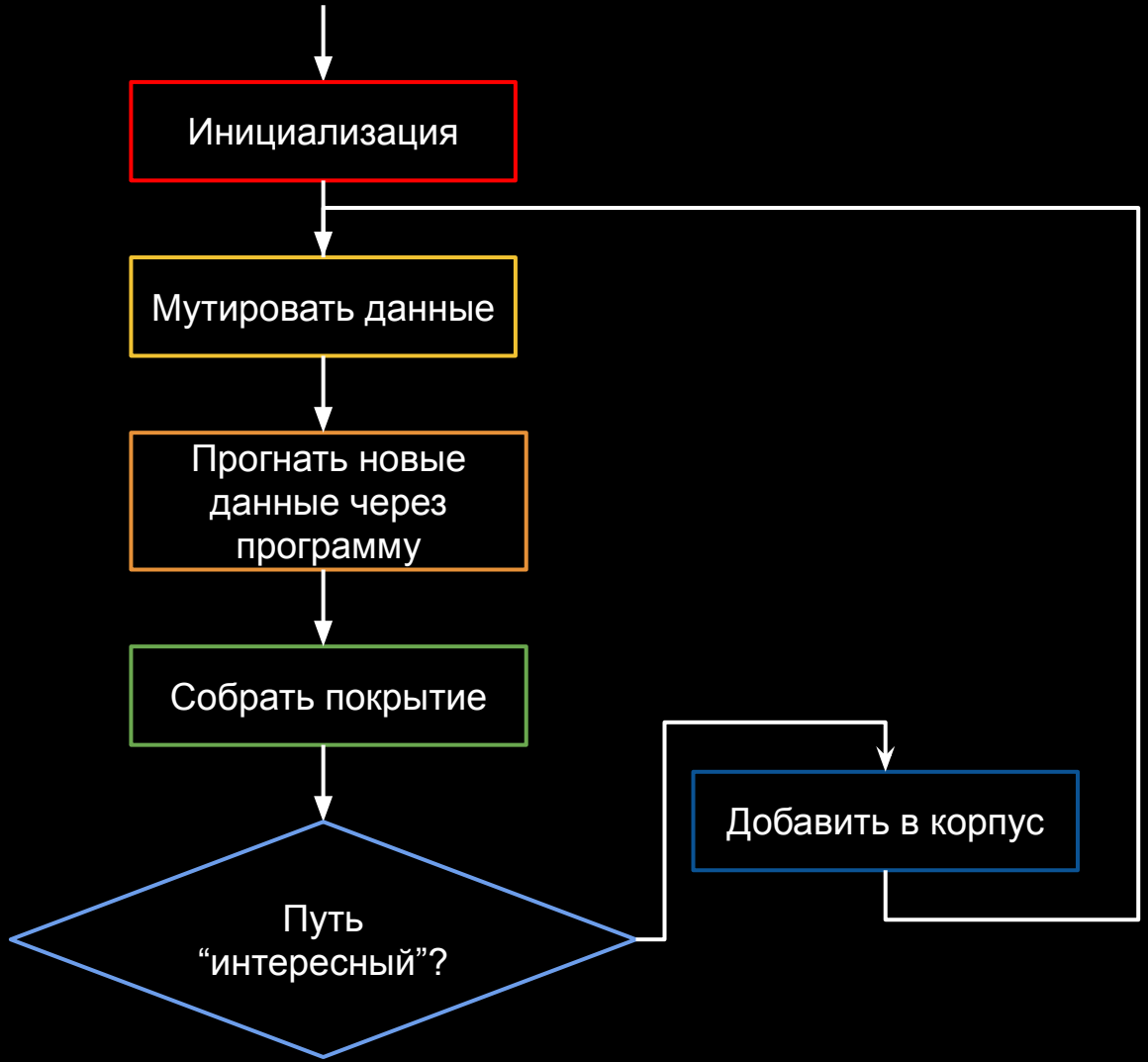
Больше: AFL/docs/technical_details.txt

```
shared_mem[c1 ^ prev_location]++;
prev_location = c1;
```



```
shared_mem[c1 ^ prev_location]++;
prev_location = c1 >> 1;
```




Инициализация
Мутировать данные
Прогнать новые данные через программу
Собрать покрытие
Путь “интересный”?
Добавить в корпус


Инициализация
Мутировать данные
Прогнать новые данные через программу
Собрать покрытие
Путь “интересный”?
Добавить в корпус
А как… мутировать?
А как определить интересный ли путь?
Пересоздавать ли процесс заново каждый цикл?

The tool can be thought of as a collection of hacks that have been tested in practice,found to be surprisingly effective...

— Michal Zalewski (lcamtuf)

Если это работает, то это работает

Типичные мутации:

- bit flip
- byte flip
- known integer
- arithmetic operations
- test case splicing
- erase bytes
- dictionary mutations
- . . .

https://lcamtuf.blogspot.com/2014/08/binary-fuzzing-strategies-what-works.html

Если это работает, то это работает

Интересный ли путь:

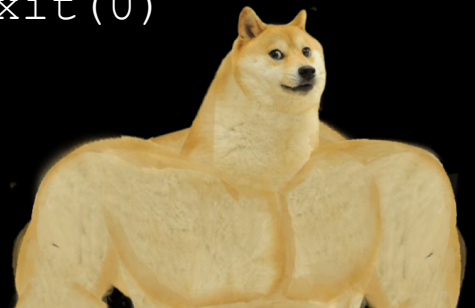
- новое ребро покрыто — интересно!
- отслеживаем циклы
- эвристики
- имитация отжига

AFL

Использует новый процесс на каждый запуск

Не крэшится, если есть утечки памяти

Нормально работает, если программы выходит по `exit(0)`



libFuzzer

In-memory фаззер

Ускорение за счёт одного процесса

Трассировка констант

Постоянно улучшается

Улучшения

```
void check_fuzzy(char *data, size_t size)
{
    if (strstr(data, “FUZZY_DOESNT_KNOW”))
            __builtin_trap();
}
```

Улучшения

```
void check_fuzzy(char *data, size_t size)
{
    if (strstr(data, “FUZZY_DOESNT_KNOW”))
        __builtin_trap();
}

char *strstr(const char *h, const char *n)
{
    add_to_dict(n);
    return orig_strstr(h, n);
}
```

Улучшения

```
void check_fuzzy(char *data, size_t size, uint32_t y)
{
    uint32_t x = *(uint32_t *)data;

    if (x == y) {
        __builtin_trap();
    }
}
```

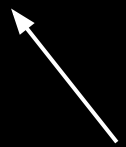
Улучшения

```
void check_fuzzy(char *data, size_t size, uint32_t y)
{
    uint32_t x = *(uint32_t *)data;

    __trace_cmp(x);
    __trace_cmp(y);
    if (x == y) {
        __builtin_trap();
    }
}
```

https://github.com/llvm-mirror/compiler-rt/blob/master/test/fuzzer/AbsNegAndConstant64Test.cpp

Тулзы!

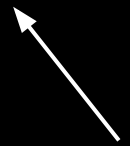
```
int fd;
char *buffer;
int flags = O_WRONLY|O_APPEND|O_CREAT;

if ((fd = open(“data.log”, flags, 0666)) == 0) exit(1);

buffer = malloc(u32size + 1);
...
```



Как обнаружить такую ошибку?

Тулзы!

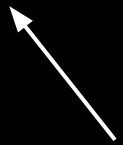
```
int fd;
char *buffer;
int flags = O_WRONLY|O_APPEND|O_CREAT;

if ((fd = open(“data.log”, flags, 0666)) == 0) exit(1);

check_int_overflow(u32size, 1);
buffer = malloc(u32size + 1);
...
```



Вкомпилим проверку! (UBSAN)

Тулзы!

```
char *buffer = malloc(strlen(s));

strcpy(buffer, s);
```



Как обнаружить переполнение?

Тулзы!

```
char *buffer = malloc(strlen(s));

strcpy(buffer, s);
```

Тулзы!

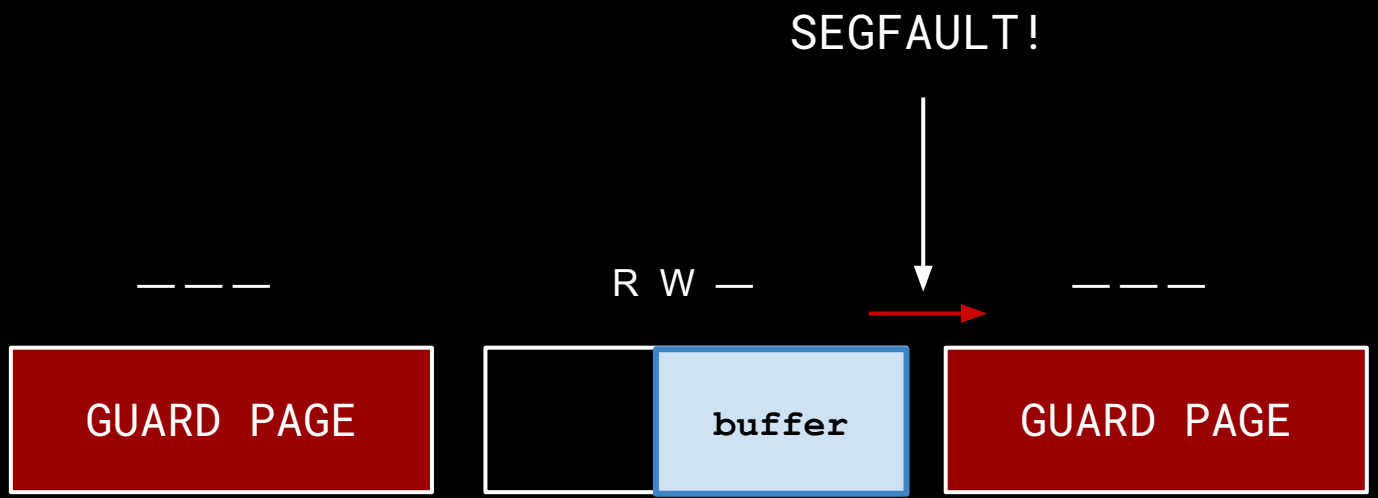
```
char *buffer = malloc(strlen(s));

strcpy(buffer, s);
```

Тулзы!

```
char *buffer = malloc(strlen(s));

strcpy(buffer, s);
```

Тулзы*!

```
char *buffer = malloc(strlen(s));

strcpy(buffer, s);
```



* Это для бедняков,
Address Sanitizer -- для мастеров.

Давайте пофаззим уже что-нибудь!

Про что мы сегодня не поговорили

- Фаззинг структурированных данных
- Фаззинг при помощи снэпшотов памяти
- Фаззинг сетевых приложений
- Как параллелить фаззинг
- Дедупликация крэшей
- Бинарная инструментация и фаззинг без исх. кода
- Динамическая инструментация
- Сбор покрытия в программе с тредами

Инфобез это не (только) кавычки

Для того, чтобы лучше фаззить нужно разбираться в:

- компиляторах
- операционных системах
- распределенных системах
- параллелизме
- алгоритмах (и уметь их хорошо программировать)
- низкоуровневые технологии

Настрой на фаззинг 99% 🦆

- Исходники фаззеров — лучший источник знаний о них
- Документация AFL — круто
- Фаззер хорош, если он нашёл баг
- “Гонка вооружений”
- Стримы gamozolabs