

Метод совмещения результатов статического и динамического анализа для цикла разработки безопасного программного обеспечения

Мишечкин Максим Владимирович

mishmax@ispras.ru

Курмангалеев Шамиль Фаимович

kursh@ispras.ru

Сложности для статического анализа

Ошибки первого и второго рода из-за:

- использование виртуальных методов;
- использование вызовов функций по указателям на вычисляемый адрес;
- недостаточной точности построения карты памяти;
- ...

Необходимость ручной разметки предупреждений:

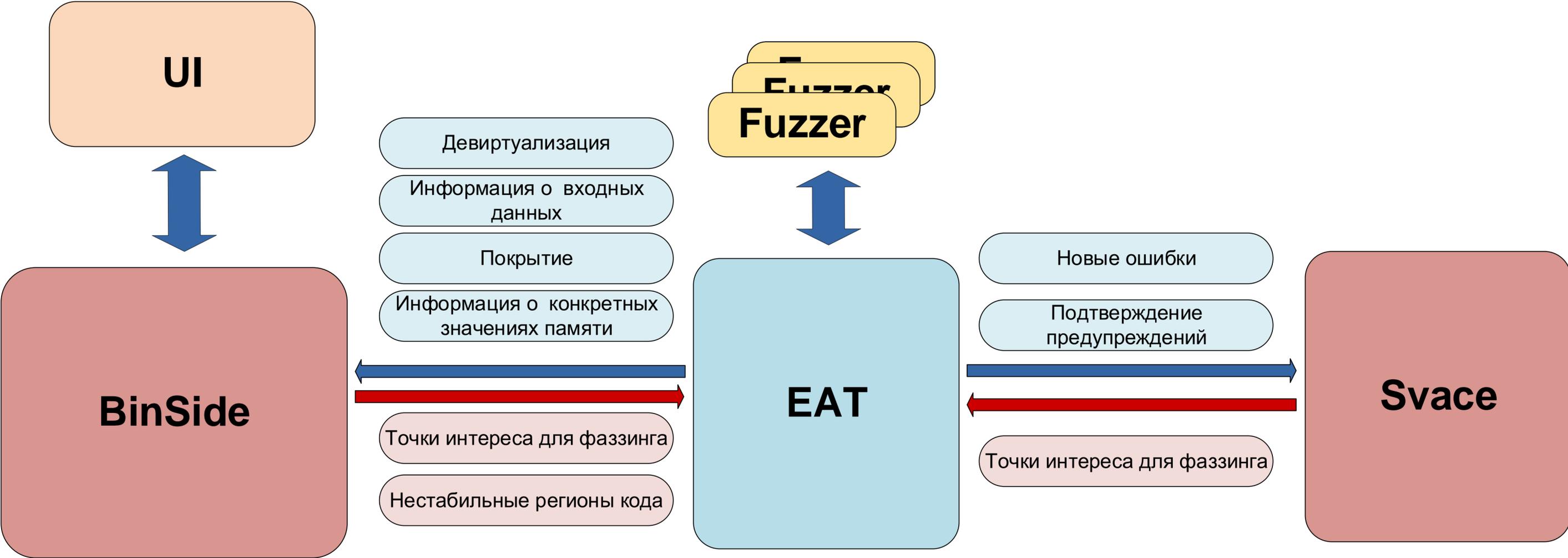
- разметка каждого предупреждения — это временные затраты;
- не полное понимание приоритета исправления найденных ошибок;
- нет гарантии, что человек правильно классифицирует ошибки;
- нет контекста на котором проявляется ошибка.

Сложности для динамического анализа

- ограниченная эффективность выбора входных данных для мутации;
- нестабильные регионы кода, которые влияют на качество проводимых мутаций;
- ограниченное понимание влияния байт на покрытие;

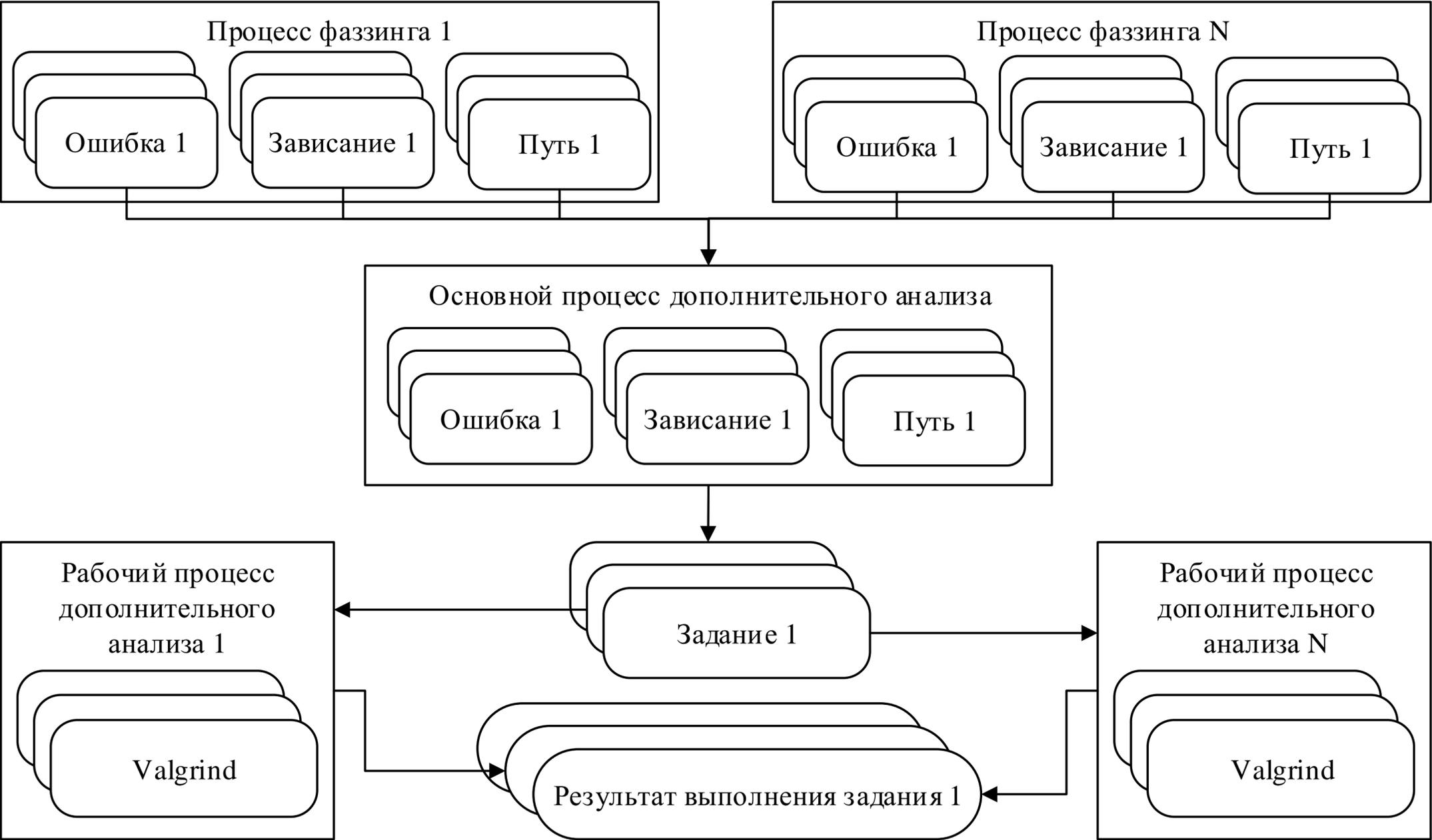
Crusher

Объединение статического и динамического анализа



Crusher

Модуль дополнительного анализа



Crusher

Интегрированные средства динамического анализа:

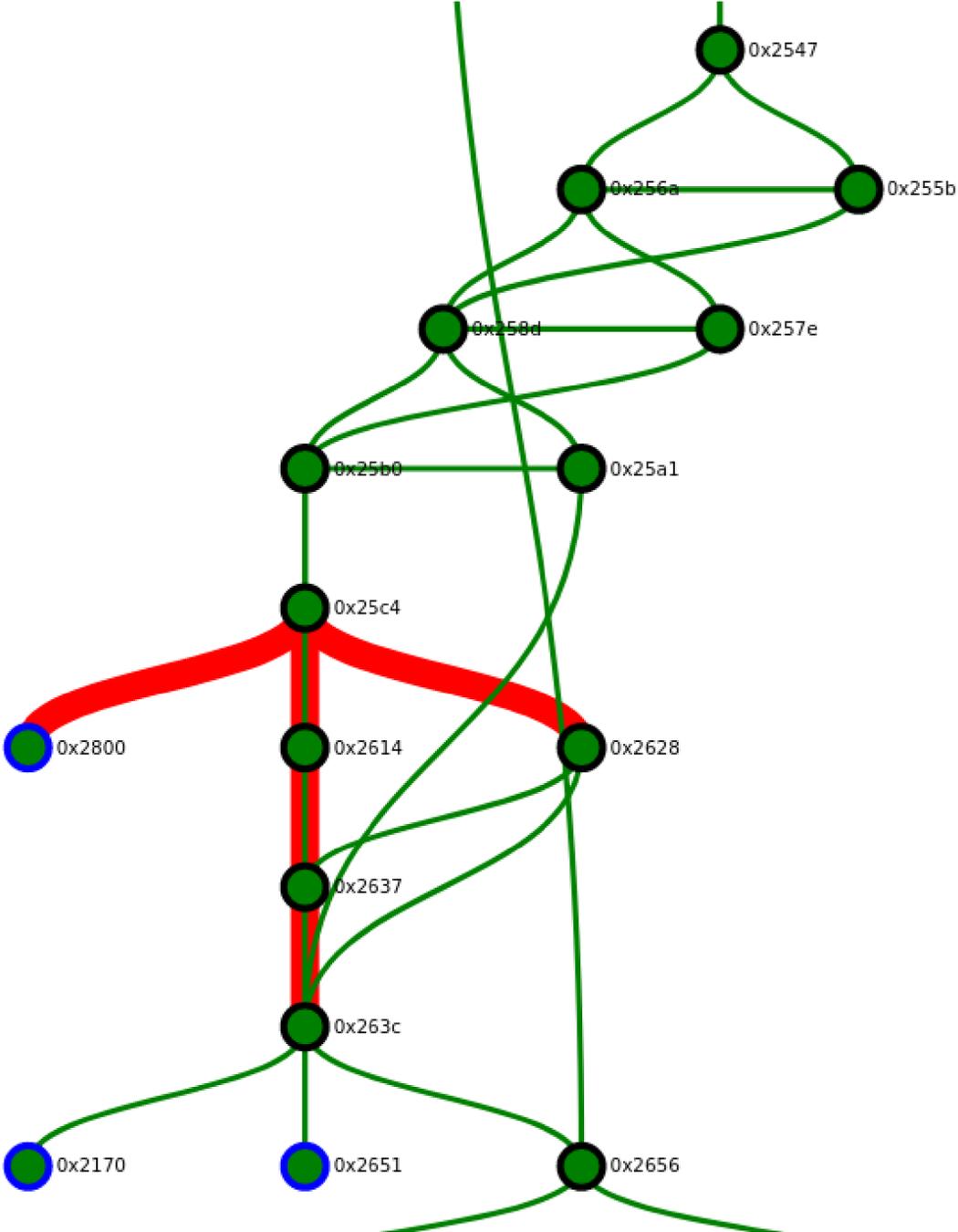
- Valgrind;
- Средства оценки критичности (exploitable, casr);
- DrMemory;
- QASan.

Результаты расширенного анализа:

- Отчеты аварийных завершений для Svace;
- Подтверждение Svace Warnings;
- Lighthouse покрытие;
- Трассы для инструмента BinSide.

Crusher

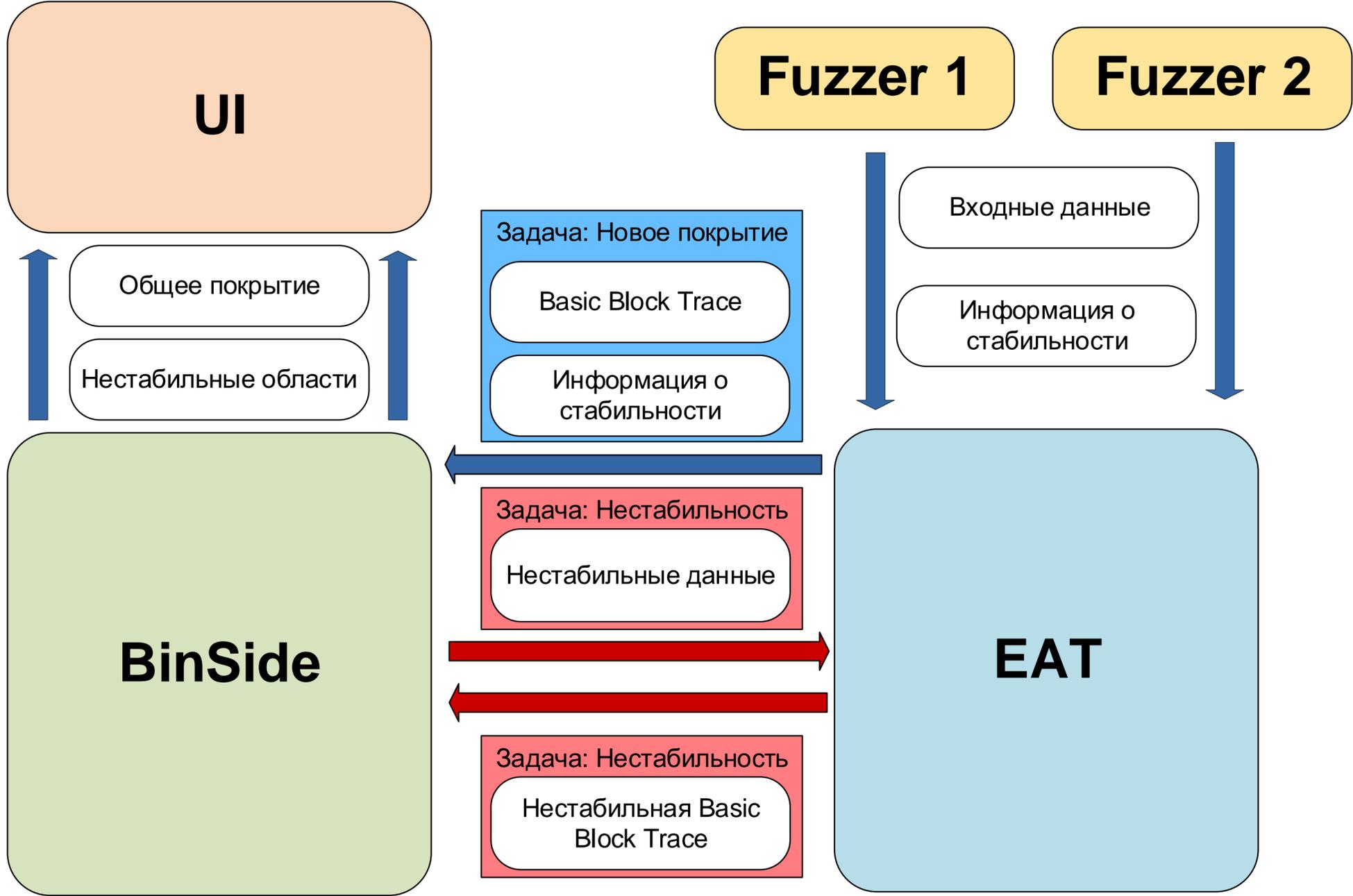
Локализация нестабильного поведения в ПО



```
37     if (data[4] == 'f') {  
38         i++;  
39         int rand_number = dist(e);  
40         if (rand_number % 2 == 0) {  
41             i++;  
42         } else {  
43             i--;  
44         }  
45     }
```

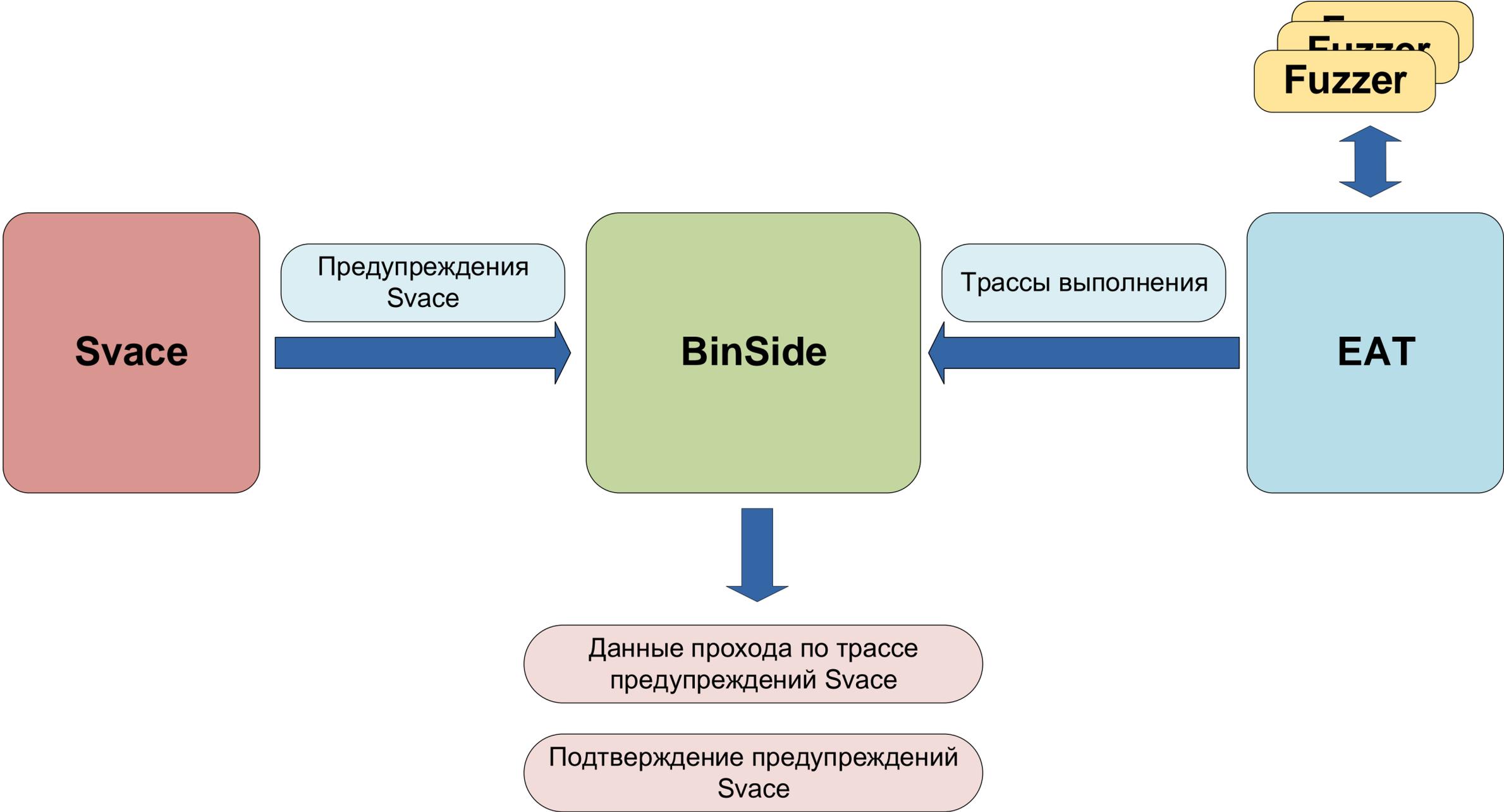
Crusher

Локализация нестабильного поведения в ПО



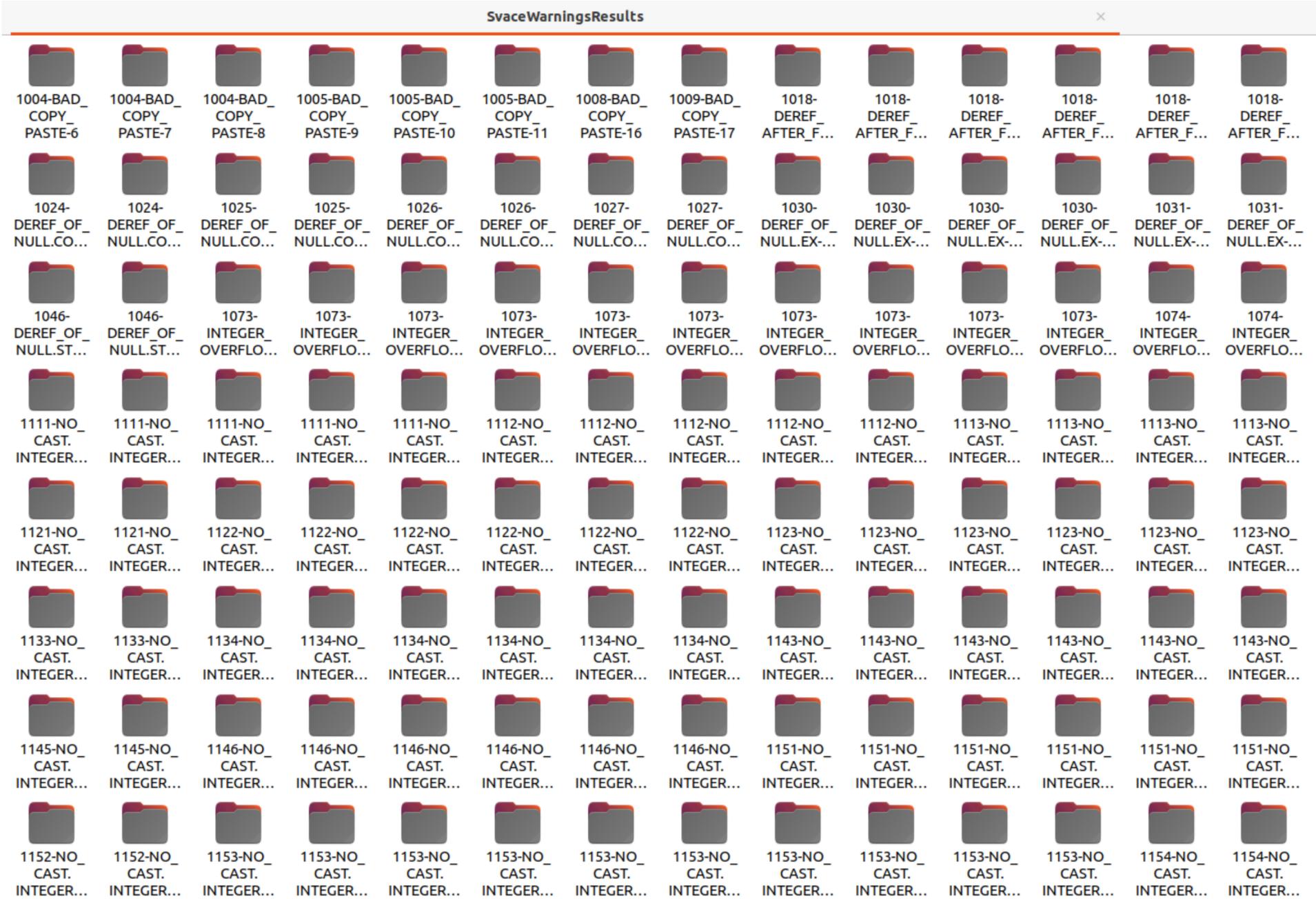
Crusher

Подтверждение предупреждений Svace



Crusher

Подтверждение предупреждений Svace (результаты NGINX)



Crusher

Свace отчеты аварийных завершений

The screenshot displays the Svace web interface for analyzing crash reports. The browser address bar shows the URL: `localhost:8060/history/svacelight/index.html#show/test1/master/43138056b03d188970cbe1e458e3fd93e5882dd8/dd1a3fd3d5d7`. The interface includes a navigation bar with filters for 'test1', 'master', and a date 'Fri Dec 28 16:51:09 MSK 2018'. A 'Show' button and a link to '.svres:export import' are also visible.

The main content area is divided into two panels. The left panel, titled 'WARNING_BY_FUZZER (6)', lists several crash reports. The selected report, 'WARNING_BY_FUZZER (crashed-1) Program can crash here with signal 11' at 'prog.c:5', is highlighted with a blue border. It includes a backtrace:

- [backtrace] crash1 at prog.c:5
- fun at prog.c:25
- main at prog.c:34

Below the backtrace are buttons for 'Add comment', 'History', and dropdown menus for 'Undecided' and 'Unspecified'.

The right panel shows the source code for `/home/fedor/fuzzer/svace-test/test1/prog.c`. The code is as follows:

```
1 #include <stdio.h>
2
3 void crash1(void)
4 {
5     (crashed-1) Program can crash here with signal 11 [backtrace] crash1
6     *(int*)0 = 1;
7 }
8
9 void crash2(void)
10 {
11     *(int*)0 = 2;
12 }
13
14 void fun(int c)
15 {
16     int s = 0;
17     int i;
18     for (i = 0; i < c % 5 + 4; i++) {
19         s += i;
20     }
21     if (c % 2 == 0)
22         s += 1;
23     if (c % 3 == 0)
```

Crusher

Свace покрытие

proj master Tue Jan 31 12:51:16 MSK 2023 Show .svres:export import

WARNING_BY_FUZZER (1)

WARNING_BY_FUZZER (coverage_queue) General previously fixed
report

GENERAL_WARNING:0

UD TP WF FP UC Severity Action History

- [backtrace] calc() at 1.cpp:9
- main at 1.cpp:17
- calc() at 1.cpp:10
- calc() at 1.cpp:7
- calc() at 1.cpp:8
- main at 1.cpp:12
- calc() at 1.cpp:5
- main at 1.cpp:13
- calc() at 1.cpp:6
- __static_initialization_and_destruction_0(int, int) at iostream:74
- _GLOBAL__sub_I_Z4calcv at 1.cpp:19
- main at 1.cpp:18

Add comment

```
/fuzzing-tool/proj/1.cpp
1
2 #include <fstream>
3 #include <iostream>
4
5 void calc() {
6     int a = 2;
7     std::string b = "123";
8     int c = 3;
9     a += 3;
10 }
11
12 int main(int argc, char* argv[]) {
13     if (argc > 2) {
14         std::ofstream stream;
15         stream.open(argv[1]);
16     }
17     std::cout << "That`s ok" << std::endl;
18     calc();
19 }
20
```

Crusher

Valgrind отчеты Clion

The screenshot displays the Clion IDE interface with the following components:

- Project Tree:** Shows the project structure for 'clean_build' (nginx-1.21.4) with subdirectories like 'core' containing various source files such as 'ngx_alloc.c'.
- Code Editor:** Displays the source code for 'ngx_alloc.c', showing functions like 'ngx_alloc' and 'ngx_calloc'. The code includes memory allocation logic using 'malloc' and 'ngx_alloc'.
- Valgrind Memcheck:** A window at the bottom showing the results of a memory check. It reports 14 warnings under the category 'Leak_PossiblyLost', all originating from 'ngx_alloc.c'. The warnings list various memory sizes (e.g., 128 bytes, 16,384 bytes, 4,096 bytes) and their corresponding loss records.
- Frame Information:** A window on the right showing the current stack frame, which is 'ngx_alloc' at the point of the 'p = malloc(size);' assignment.

Crusher

Valgrind отчеты Svace

The screenshot displays the Svace web interface for a project named 'v1.21.4-loc...'. The interface is divided into several sections:

- Top Bar:** Shows the project name, branch (master), and navigation options like 'Review', 'Browse code', 'Reports', and 'Settings'. The user 'admin' is logged in.
- Snapshot:** A dropdown menu shows 'Snapshot 2022-11-10 18:33:00 +0300' and a 'Compare with' option.
- Checkers:** A sidebar on the left lists 'Leak_PossiblyLost null (14)' and 'ngx_cycle.c : 284'. Below this, there are 14 entries with '??:0'.
- Source Code:** The main area shows the source code for 'ngx_alloc.c'. A 'TRACE 1.2' marker is visible at line 22, indicating a 'Binside trace element | 0x3D752'. The code includes headers, variable declarations, and functions like 'ngx_alloc', 'ngx_calloc', and 'ngx_memalign'.
- Message:** A panel on the right shows the message 'EMPTY BINSIDE COMMENT'. Below it, a list of 'Binside trace element' entries is displayed, including addresses like '0x47337EF', '0x3D752', '0x1AFB7', '0x42C96', and '0x1AAC2'.

Спасибо за просмотр