

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:

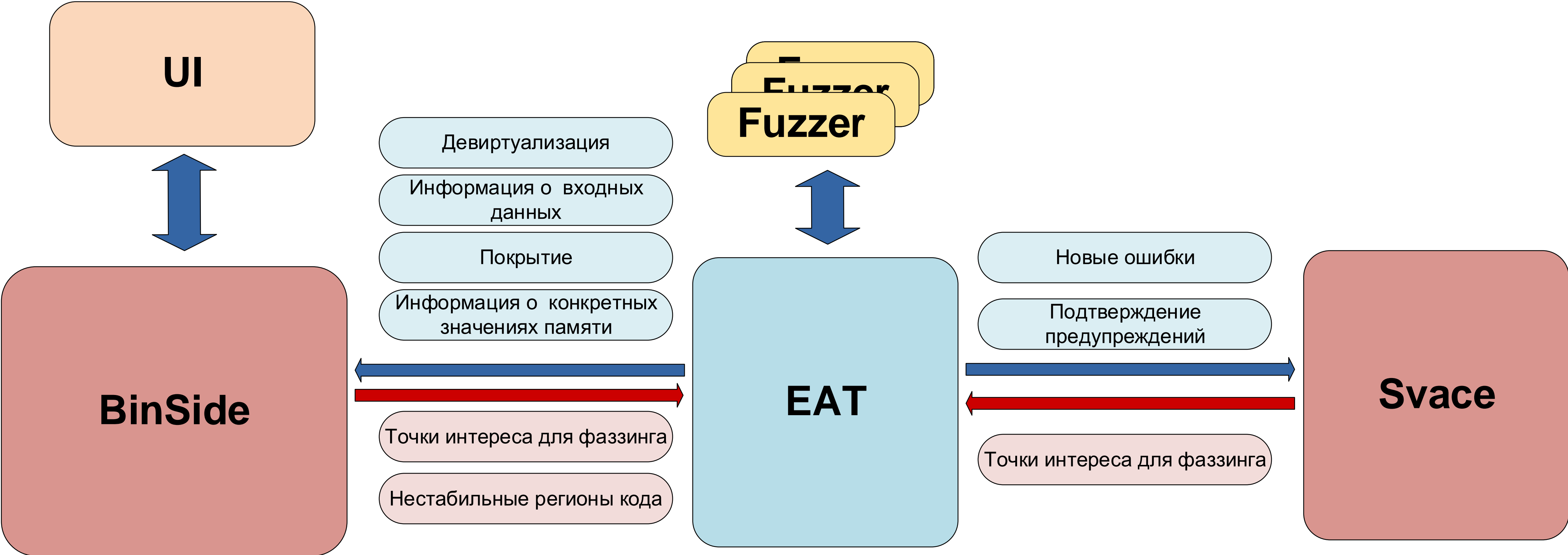
- разметка каждого предупреждения — это временные затраты;
- не полное понимание приоритета исправления найденных ошибок;
- нет гарантии, что человек правильно классифицирует ошибки;
- нет контекста на котором проявляется ошибка.

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;

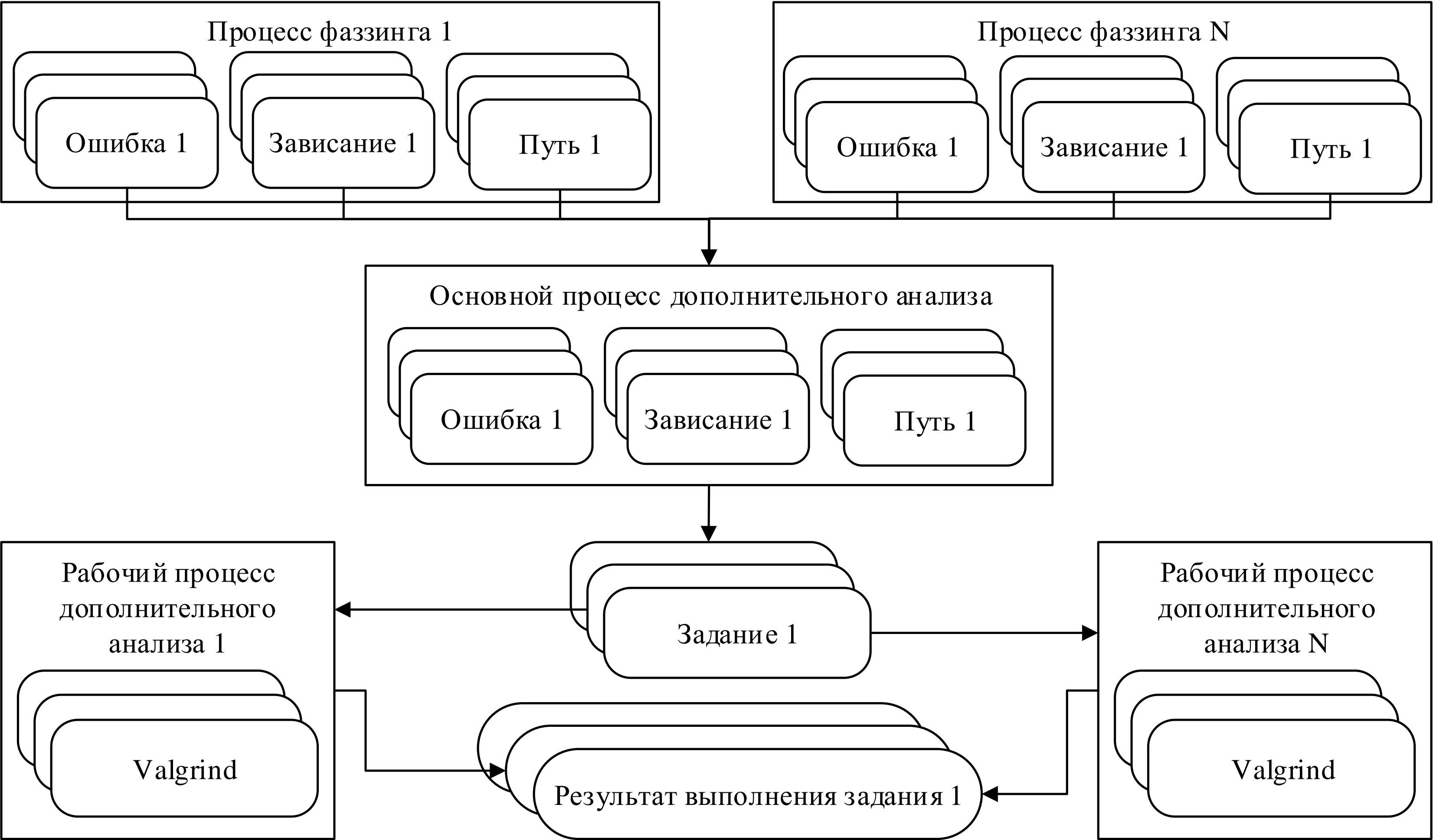
Crusher

Объединение статического и динамического анализа



Crusher

Модуль дополнительного анализа



Crusher

Интегрированные средства динамического анализа:

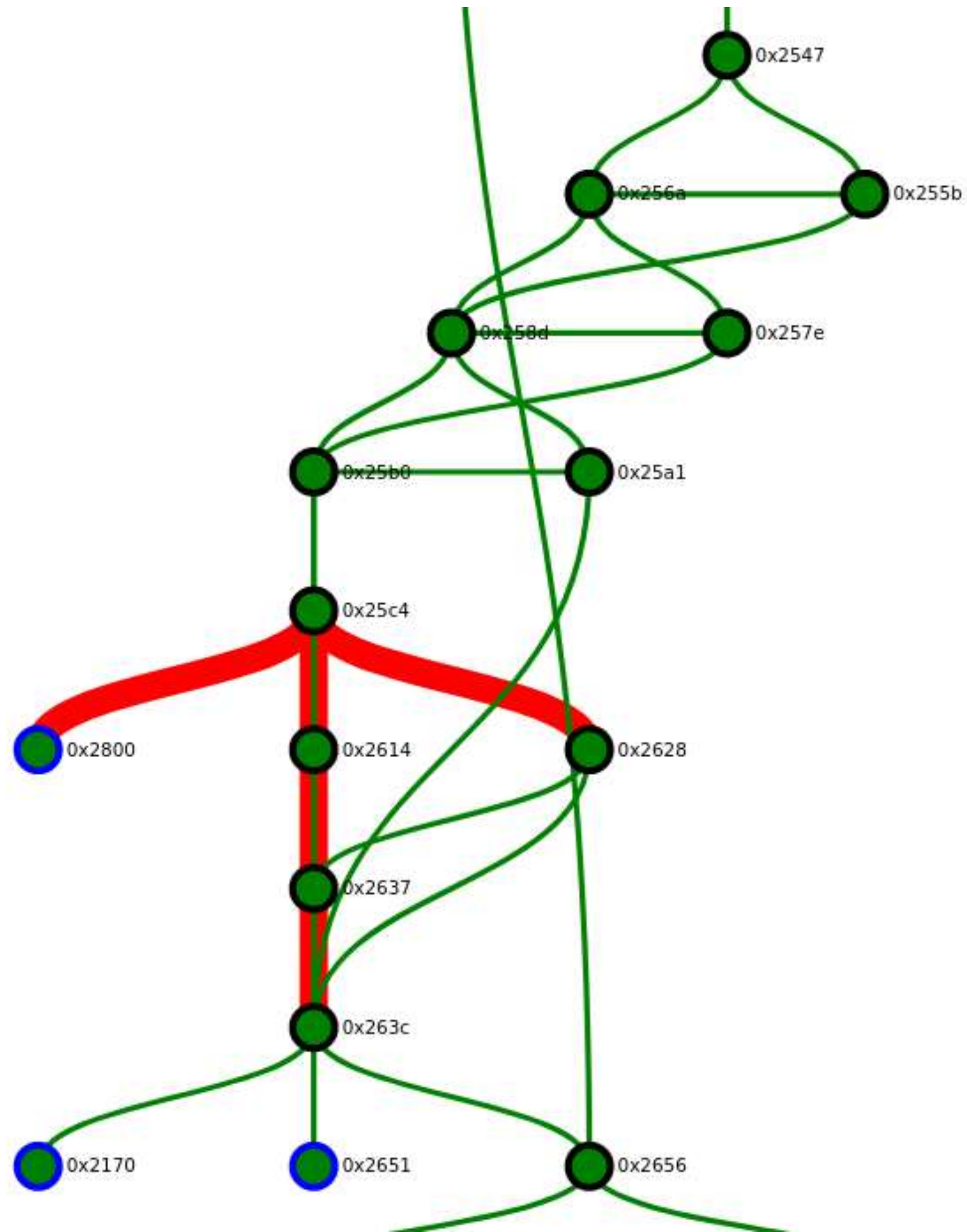
- Valgrind;
- Средства оценки критичности (exploitable, casr);
- DrMemory;
- QASan.

Результаты расширенного анализа:

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

Crusher

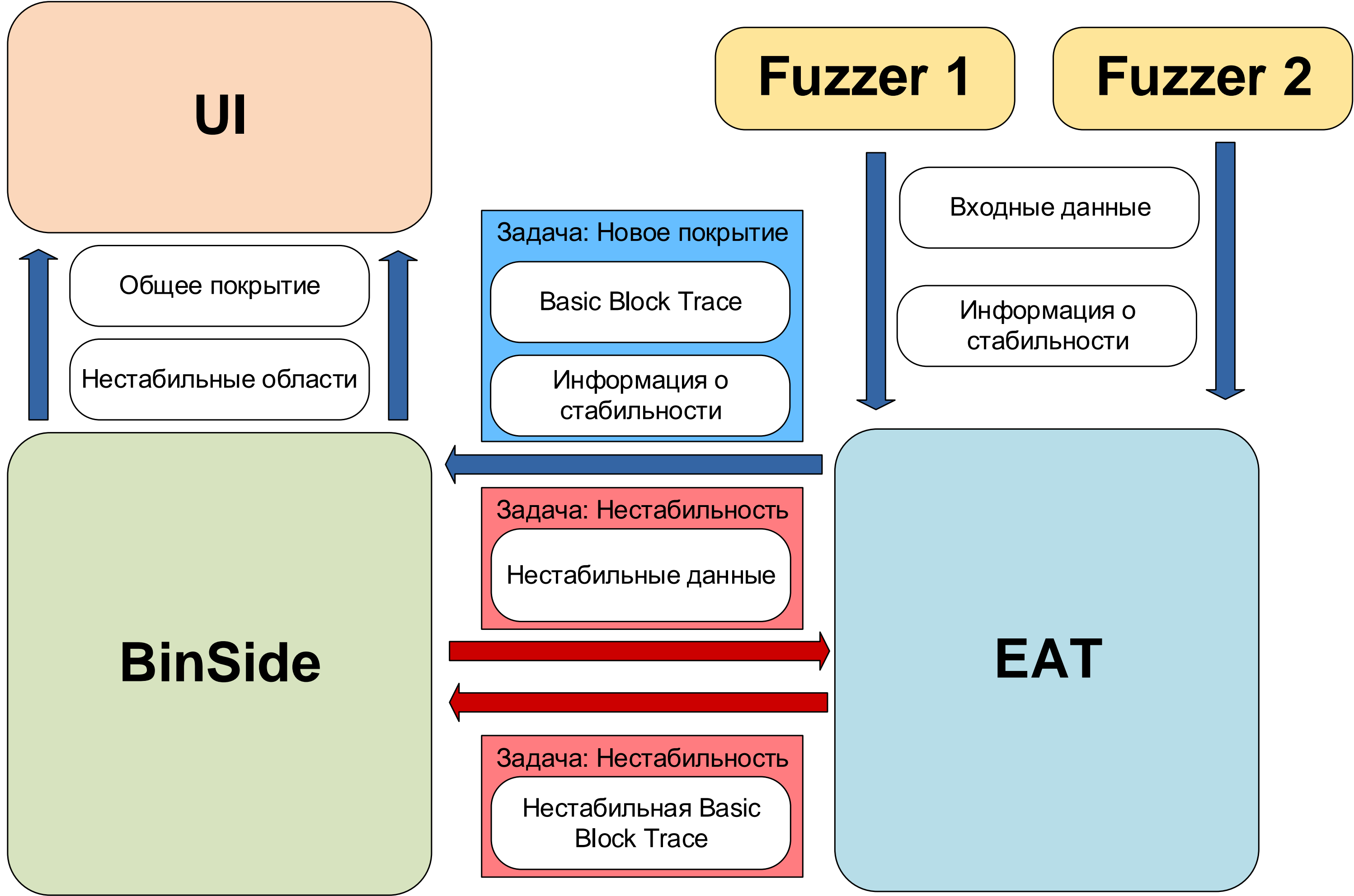
Локализация нестабильного поведения в ПО



```
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     }
```

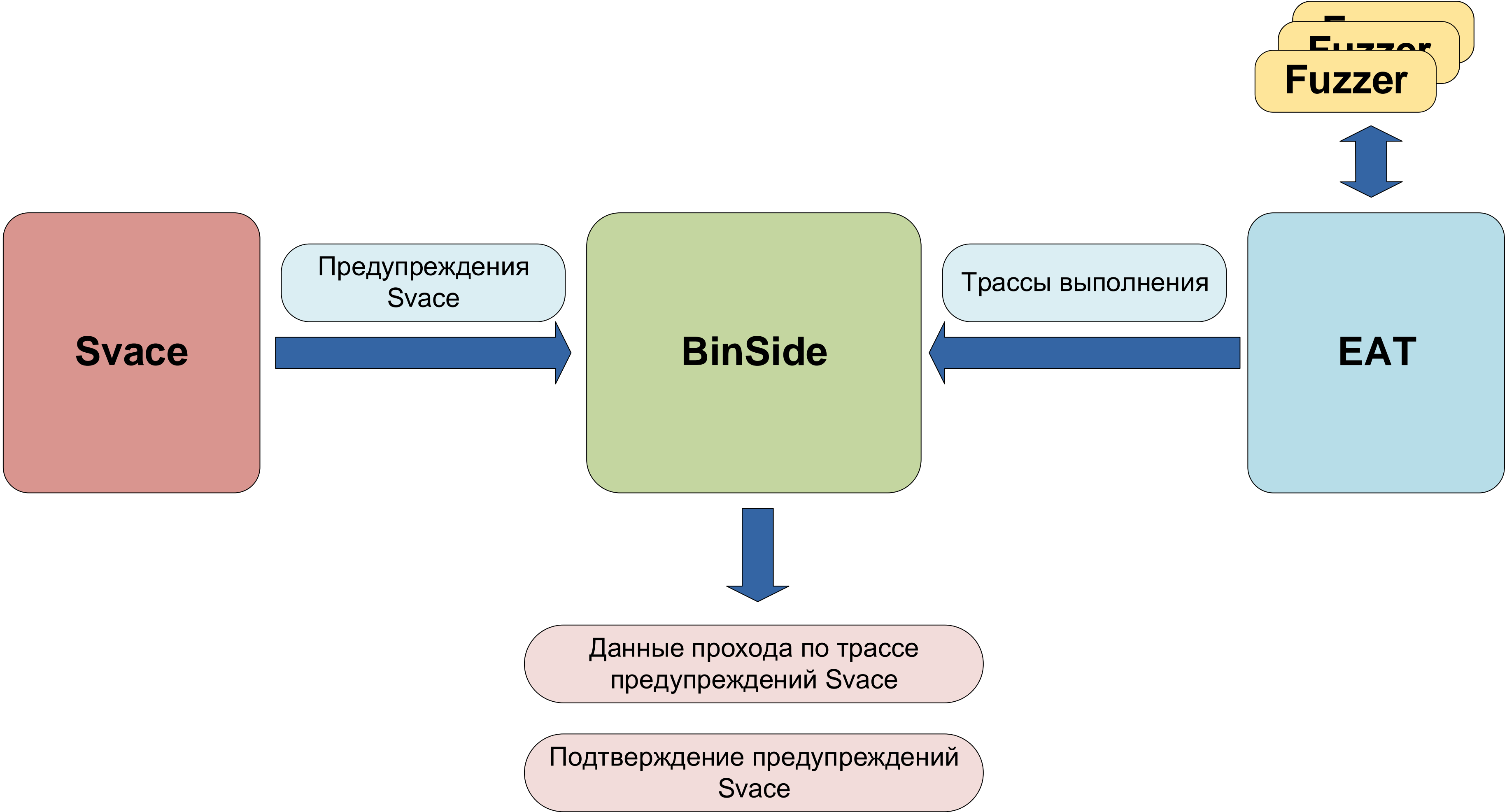
Crusher

Локализация нестабильного поведения в ПО



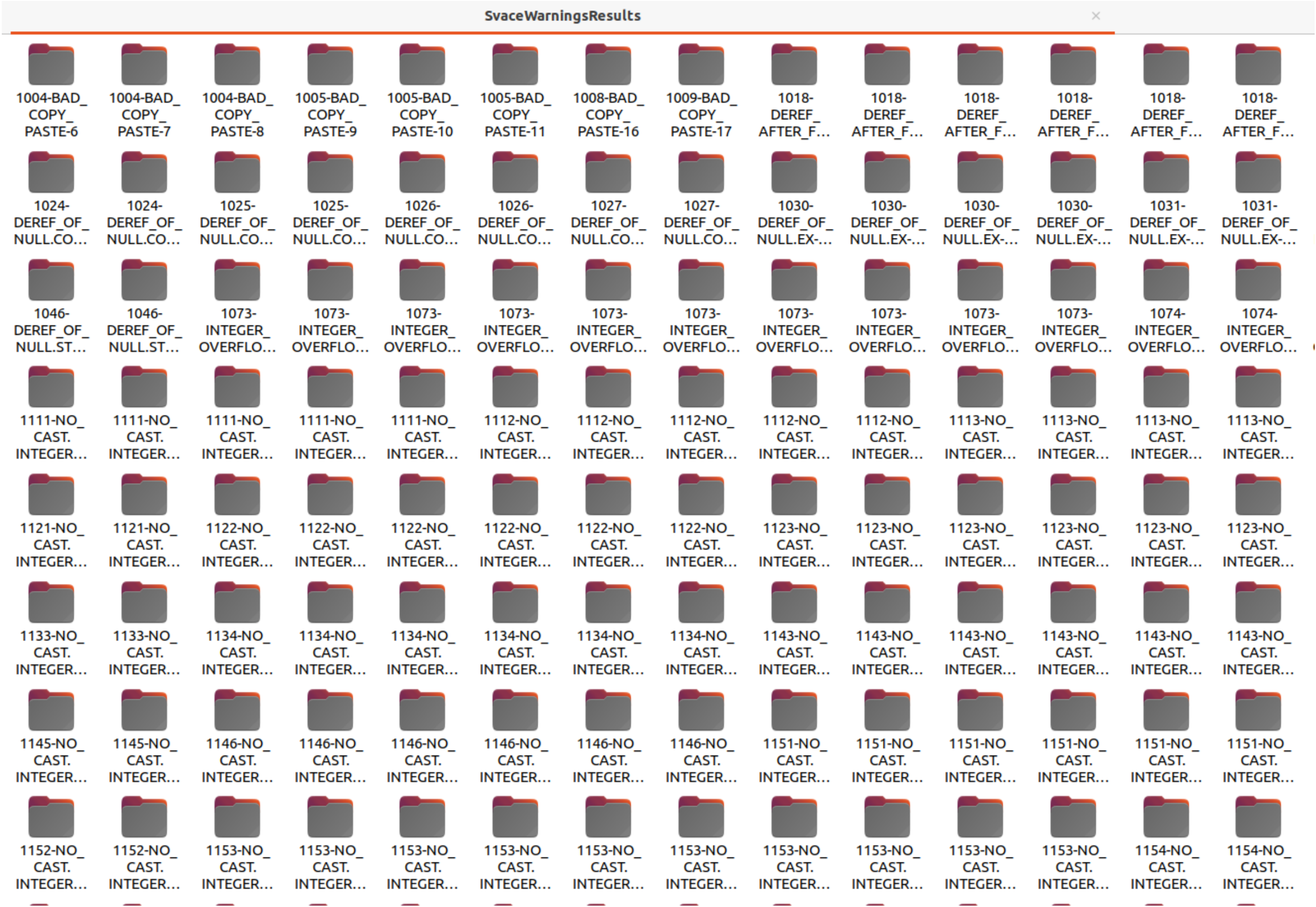
Crusher

Подтверждение предупреждений Svace



Crusher

Подтверждение предупреждений Svace (результаты NGINX)



Crusher

Свace отчеты аварийных завершений

The screenshot displays the Svace web interface. At the top, there is a browser window with the URL `localhost:8060/history/svacelight/index.html#show/test1/master/43138056b03d188970cbe1e458e3fd93e5882dd8/dd1a3fd3d5d7`. Below the browser, the interface shows a navigation bar with "test1" and "master" dropdowns, a date "Fri Dec 28 16:51:09 MSK 2018", and a "Show" button. To the right of the navigation bar are links for ".svres:export import".

The main content area is divided into two columns. The left column displays a list of "WARNING_BY_FUZZER (6)" reports. Each report includes a "new" button and a description: "WARNING_BY_FUZZER (crashed-4) Program can crash here with signal 11" at `prog.c:10`. The fourth report, "WARNING_BY_FUZZER (crashed-1) Program can crash here with signal 11" at `prog.c:5`, is highlighted with a blue border and contains a backtrace:

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

Below the backtrace is an "Add comment" link and a "History" dropdown menu with "Undecided" and "Unspecified" options.

The right column displays 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

Space 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 an IDE window with the following components:

- Project Explorer:** Shows the directory structure of the project, including `clean_build`, `nginx-1.21.4`, and `core` subdirectories.
- Code Editor:** Displays the source code for `ngx_alloc.c`. The code includes:

```
13 ngx_uint_t ngx_pagesize_shift;  
14 ngx_uint_t ngx_cacheline_size;  
15  
16  
17 void *  
18 ngx_alloc(size_t size, ngx_log_t *log)  
19 {  
20     void *p;  
21  
22     p = malloc(size);  
23     if (p == NULL) {  
24         ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,  
25                     "malloc(%uz) failed", size);  
26     }  
27  
28     ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, log, 0, "malloc: %p:%uz", p, size);  
29  
30     return p;  
31 }  
32  
33  
34 void *  
35 ngx_calloc(size_t size, ngx_log_t *log)  
36 {  
37     void *p;  
38  
39     p = ngx_alloc(size, log);  
40 }
```
- Run Valgrind Memcheck:** Shows 14 warnings from `ngx_alloc.c`. The first warning is expanded to show:

```
128 bytes in 1 blocks are possibly lost in loss record 4 of 23 1 warning  
0x483B7F3 malloc  
0x145832 ngx_alloc ngx_alloc.c:22  
0x12D409 ngx_crc32_table_init ngx_crc32.c:117  
0x1228E2 main ngx.c:274  
16,384 bytes in 1 blocks are possibly lost in loss record 16 of 23 1 warning  
16,384 bytes in 1 blocks are possibly lost in loss record 17 of 23 1 warning  
16,384 bytes in 1 blocks are possibly lost in loss record 18 of 23 1 warning  
16,384 bytes in 1 blocks are possibly lost in loss record 19 of 23 1 warning  
16,384 bytes in 1 blocks are possibly lost in loss record 20 of 23 1 warning  
4,096 bytes in 1 blocks are possibly lost in loss record 8 of 23 1 warning  
4,096 bytes in 1 blocks are possibly lost in loss record 9 of 23 1 warning  
4,280 bytes in 1 blocks are possibly lost in loss record 10 of 23 1 warning  
4,280 bytes in 1 blocks are possibly lost in loss record 11 of 23 1 warning  
4,280 bytes in 1 blocks are possibly lost in loss record 12 of 23 1 warning  
4,544 bytes in 1 blocks are possibly lost in loss record 13 of 23 1 warning  
4,672 bytes in 1 blocks are possibly lost in loss record 14 of 23 1 warning  
8 bytes in 1 blocks are possibly lost in loss record 1 of 23 1 warning
```
- Frame Information:** Shows the current stack frame for `ngx_alloc`, with the line `p = malloc(size);` highlighted in green.

Crusher

Valgrind отчеты Svace

The screenshot displays the Svace application interface, which is used for analyzing Valgrind reports. The main window shows a list of memory leaks on the left, with the first one selected: "Leak_PossiblyLost null (14)" from "ngx_cycle.c: 284". The central pane shows the source code of the file, with a "TRACE 1.2" marker highlighting a "Binside trace element | 0x3D752" at line 22. The code includes headers for ngx_config.h and ngx_core.h, and defines variables for ngx_pagesize, ngx_pagesize_shift, and ngx_cacheline_size. It also shows the ngx_alloc function, which is the source of the memory leak. The right pane shows the "Message" section, which is currently empty, and a list of "Binside trace elements" with their addresses and file names.

Snapshot: Snapshot 2022-11-10 18:33:00 +0300

Checkers

Files

Filters

Snapshot information

Source code

Show table

Leak_PossiblyLost null ngx_cycle.c:284

Trace

Comments

Details

Message:

EMPTY BINSIDE COMMENT

1. Role: defect

1. Binside trace element | 0x47337EF

2. Binside trace element | 0x3D752

3. Binside trace element | 0x1AFB7

4. Binside trace element | 0x42C96

5. Binside trace element | 0x1AAC2

```
1
2
3  /*
4   * Copyright (C) Igor Sysoev
5   * Copyright (C) Nginx, Inc.
6   */
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  ngx_uint_t  ngx_pagesize;
13  ngx_uint_t  ngx_pagesize_shift;
14  ngx_uint_t  ngx_cacheline_size;
15
16
17  void *
18  ngx_alloc(size_t size, ngx_log_t *log)
19  {
20      void *p;
21
22      p = malloc(size);
23      if (p == NULL) {
24          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
25                      "malloc(%uz) failed", size);
26      }
27
28      ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, log, 0, "malloc: %p:%uz", p, size);
29
30      return p;
31  }
32
33
34  void *
35  ngx_calloc(size_t size, ngx_log_t *log)
36  {
37      void *p;
38
39      p = ngx_alloc(size, log);
40
41      if (p) {
42          ngx_memzero(p, size);
43      }
44
45      return p;
46  }
47
48
49  #if (NGX_HAVE_POSIX_MEMALIGN)
50
51  void *
52  ngx_memalign(size_t alignment, size_t size, ngx_log_t *log)
53  {
54      void *p;
55      int  err;
56
57      err = posix_memalign(&p, alignment, size);
58
59      if (err) {
60          ngx_log_error(NGX_LOG_EMERG, log, err,
61                      "posix_memalign(%uz, %uz) failed", alignment, size);
62          p = NULL;
63      }
64
65      ngx_log_debug3(NGX_LOG_DEBUG_ALLOC, log, 0,
```

Спасибо за просмотр