ИСП РАН

Среда анализа бинарного кода «ТРАЛ»

Краткое руководство пользователя

СОДЕРЖАНИЕ

1 Область применения

Среда анализа предназначена для изучения свойств исполняемого бинарного кода. К типовым задачам, которые она способна решать, относятся:

- исследование вредоносного кода, в том числе в ситуациях, когда этот код не существует в виде отдельного исполняемого модуля;
- извлечение алгоритмов для повторного использования из программ и библиотек, исходный код которых утерян или не может быть заново скомпилирован из-за особенностей процесса сборки;
- интеграция программных компонент с закрытыми интерфейсами;
- отладка низкоуровневого кода, например, компонентов ОС и встраиваемого ПО.

2 Схема работы среды анализа бинарного кода ТРАЛ

В основе разработанного в Институте системного программирования РАН подхода к решению задач детального анализа бинарного кода лежит комбинация известных методов статического и динамического анализа. Подход реализован в среде анализа бинарного кода ТРАЛ и состоит из следующих шагов.

Сбор динамических профилей работы исследуемого ПО, которое выполняется в виртуальной машине.

Автоматическое повышение уровня представления профиля, собранного в ходе выполнения ПО: выделение обработчиков прерываний, разметка процессов и потоков выполнения, восстановление стеков вызовов, используемых библиотек, функций и значений параметров. Основным результатом этого этапа является последовательность связанных статических представлений, описывающих потоки данных и управления анализируемого кода.

Исследование восстановленных потоков данных и управления в рамках решения прикладных задач анализа. Среда рассчитана на встраивание в технологические цепочки, где выполняет функции поставщика высокоуровневых данных, изначально отсутствующих на уровне машинного кода. Помимо того, доступен графический интерфейс, рассчитанный на интерактивную работу человека с целью изучения и коррекции восстановленных высокоуровневых данных.

3 Основные функциональные возможности среды ТРАЛ

1. Модульная архитектура среды ТРАЛ позволяет разрабатывать в виде отдельных компонентов различные виды анализа.
2. Среда использует единое промежуточное представление бинарного кода, не зависящее от целевой процессорной архитектуры. Эта особенность позволяет разрабатывать модули анализа таким образом, чтобы один модуль был пригоден для анализа кода сразу всех поддерживаемых средой процессорных архитектур. Среда поддерживает наиболее распространённые процессорные архитектуры x86, x86-64, ARMv7.
3. Среда реализует полносистемный анализ, покрывающий все слои ПО программно-аппаратной платформы, как приложения, так и системный код. Анализ всего ПО ведется совместно, с учётом явных и побочных связей между программами. Среда содержит специализированные модули поддержки гостевых ОС семейств Windows и Linux. В случаях, когда анализируемая система построена на базе другой ОС (известной или неизвестной) или низкоуровневый код работает вне ОС, среда ТРАЛ остаётся работоспособной.
4. Одновременный анализ нескольких динамических профилей позволяет улучшить покрытие анализируемой части системы. Последовательность статических представлений кода учитывает все пути, которые реализовались в ходе сбора профилей, и может быть дополнена по статическим снимкам памяти.
5. В среде реализован высокоточный анализ потоков данных, который основан на классических алгоритмах теории компиляторов. В нём учитываются особенности целевых процессорных архитектур, такие как механизм виртуальной памяти, аппаратное переключение контекста при обработке прерываний и исключений, прямой доступ устройств к памяти (DMA). На базе анализа потоков данных в среде ТРАЛ реализован динамический анализ помеченных данных, в том числе учитывающий зависимости по управлению.
6. Среда анализа поддерживает возможность работы в интерактивном режиме через развитый графический интерфейс пользователя, а также режим работы по задаваемым пользователем скриптам, что востребовано при реализации автоматизированных систем анализа программного кода.
7. Подавляющее большинство алгоритмов анализа, реализованных в среде ТРАЛ, распараллелены на общей памяти и показывают на современных рабочих станциях масштабируемость, близкую к линейной.

8. Реализована двусторонняя интеграция с внешними инструментами анализа программного кода и сетевого трафика.

4 Примеры разработанных модулей

1. Автоматическое построение стеков вызовов для каждого потока выполнения в каждый момент времени, а также дерева вызовов.
2. Автоматическое вычисление параметров вызываемых функций:
   - для всех интересующих функций, которые вызывались в период сбора динамического профиля, могут быть автоматически вычислены значения их параметров;
   - в случаях, когда среди параметров функции есть указатели, могут быть вычислены не только значения указателей, но и содержимое структур, на которые они указывают.
3. Распознавание адресов загрузки и времени жизни исполняемых модулей и динамических библиотек:
   - для проведения анализа необходимы только сами бинарные модули, среда автоматически обнаружит их загрузку и выгрузку даже в случае, когда гостевая ОС неизвестна.
4. Дополнение представления кода, восстановленного по динамическим профилям, данными, полученными от сторонних средств статического анализа. Используемый подход успешно работает с вычисляемыми переходами.
5. Модуль интерактивного построения диаграмм, описывающих фрагменты машинного кода, средствами UML-подобного языка моделирования. Реализованный подход основан на автоматическом выделении блоков кода, связанных по данным или управлению с входами и выходами алгоритма.
6. Модуль абстрактной интерпретации машинного кода, позволяющий формировать ограничения для входных данных с целью реализации ранее не сработавших ветвлений.
7. Графическое представление структуры переключения процессов, потоков выполнения и адресных пространств. Такое представление удобно для получения первоначального представления о составе работающего в анализируемой системе кода с точки зрения ОС.
8. Модуль деобфускации, позволяющий автоматически построить граф потока управления, запутанный преобразованием «диспетчер», в том числе, когда для выполнения кода применяется виртуальная машина.

5 Тестовый сценарий работы со средой анализа

Для проверки работоспособности и начального ознакомления со средой анализа в комплект дистрибутива включён тестовый проект, на котором могут быть проверены основные алгоритмы, реализованные в среде. Дальнейшая часть настоящего описания представляет собой последовательность шагов, которые выполняются при анализе тестового проекта в рамках определённой задачи анализа: выявления кода, ответственного за срабатывание критического программного дефекта.

5.1 Установка среды анализа

Файл дистрибутива представляет собой инсталлятор, который автоматически установит ПО. В ходе установки можно выбрать целевой каталог (по умолчанию – “C:\Program Files\ISP RAS\Trawl”). После окончания установки для запуска используется исполняемый файл Trawl.exe, расположенный в выбранном каталоге.

Для функционирования среды требуется компьютер с 64-разрядным процессором Intel или AMD, не менее чем 8 Гбайт оперативной памяти, и не менее чем 4 Гбайт свободного места на диске. Поддерживаются 64-разрядные операционные системы Windows 7, 8 или 10.

5.2 Первоначальная настройка

По умолчанию среда анализа будет запущена с англоязычным интерфейсом (Рис. 1). Для выбора русского языка необходимо выбрать пункт меню *Tools* → *Options…* и во вкладке *General* изменить язык на русский (Рис. 2). Для применения выбора языка требуется перезапуск.

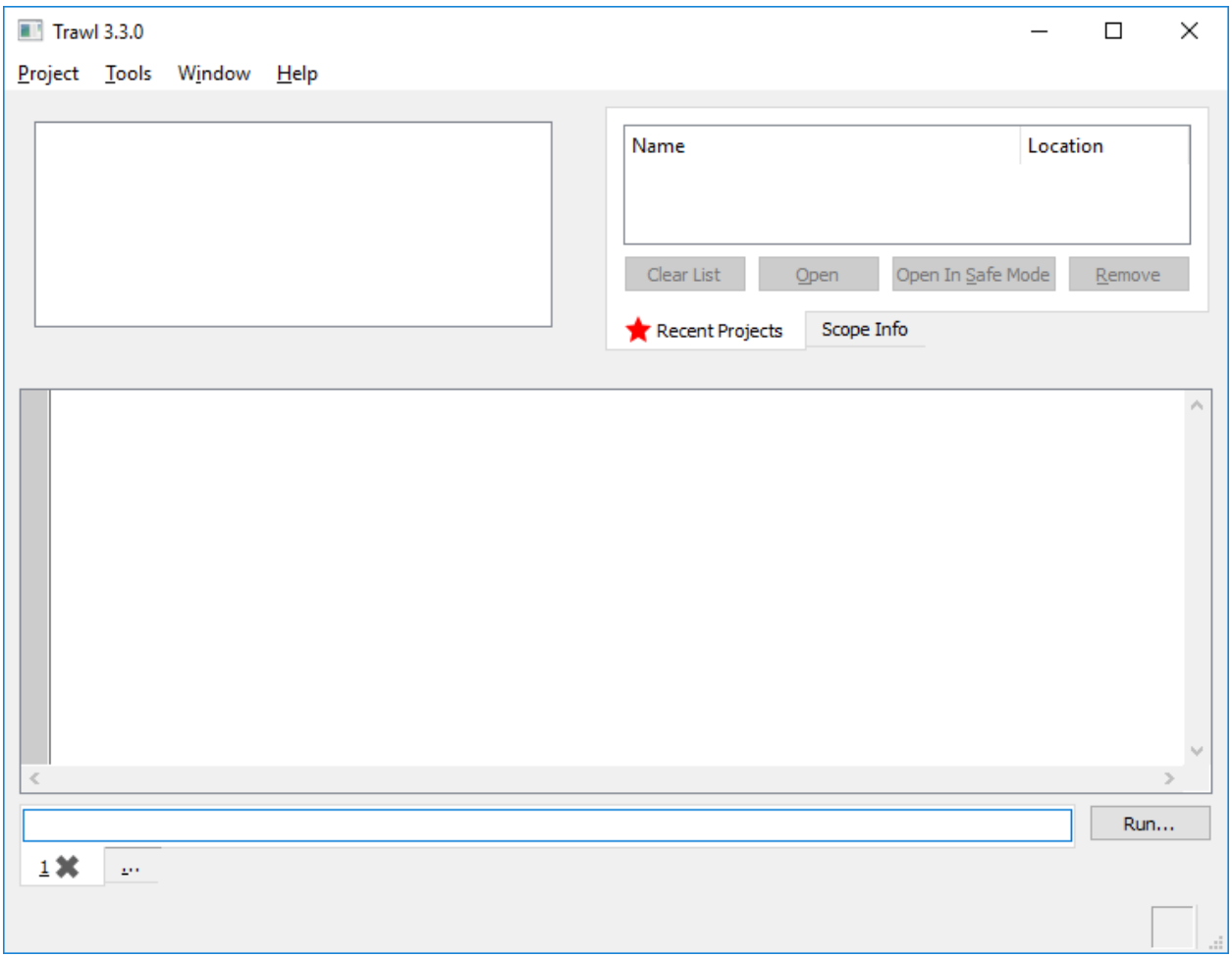

Рисунок 1 – Основное окно среды ТРАЛ (английский язык интерфейса).

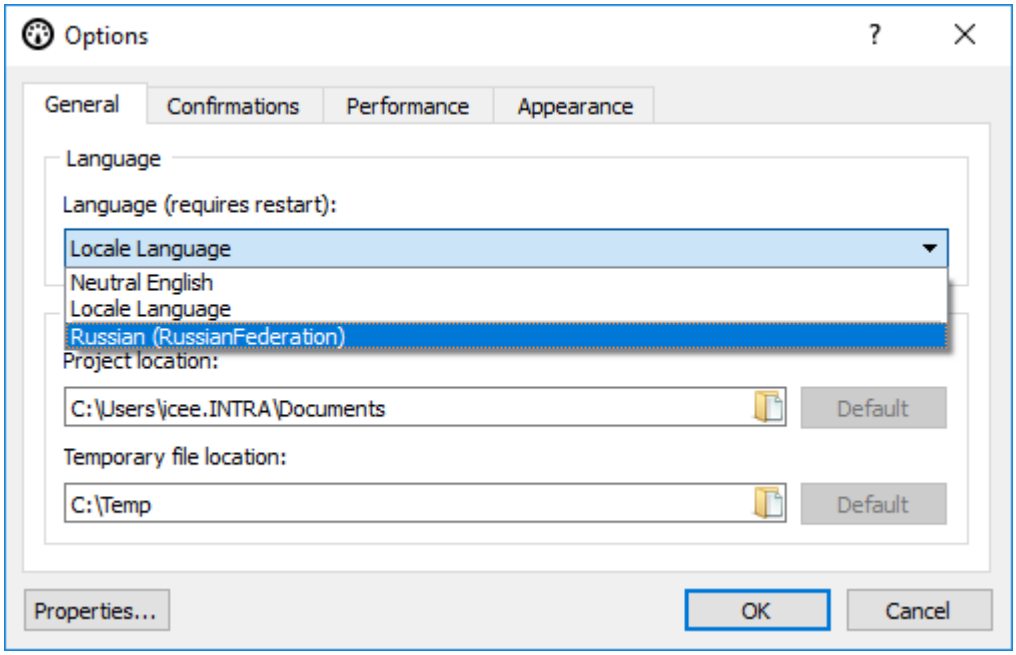

Рисунок 2 – Диалог настроек с выбором языка (английский язык интерфейса).

5.3 Создание проекта

Первое действие, которое необходимо выполнить для работы с прилагаемым динамическим профилем – создание проекта. Для этого следует выбрать пункт меню *Проект* → *Новый проект…*, что приведёт к открытию мастера нового проекта (Рис. 3). На первом шаге нужно выбрать имя проекта и каталог, в котором он будет создан. Можно выбрать любое имя, не содержащее запрещённых в именах файлов символов, и любой каталог, доступный на запись. На соответствующем диске должно быть не менее 4 Гбайт свободного пространства. После заполнения полей *Имя* и *Размещение* нажмите кнопку *Далее* для перехода к следующему шагу.

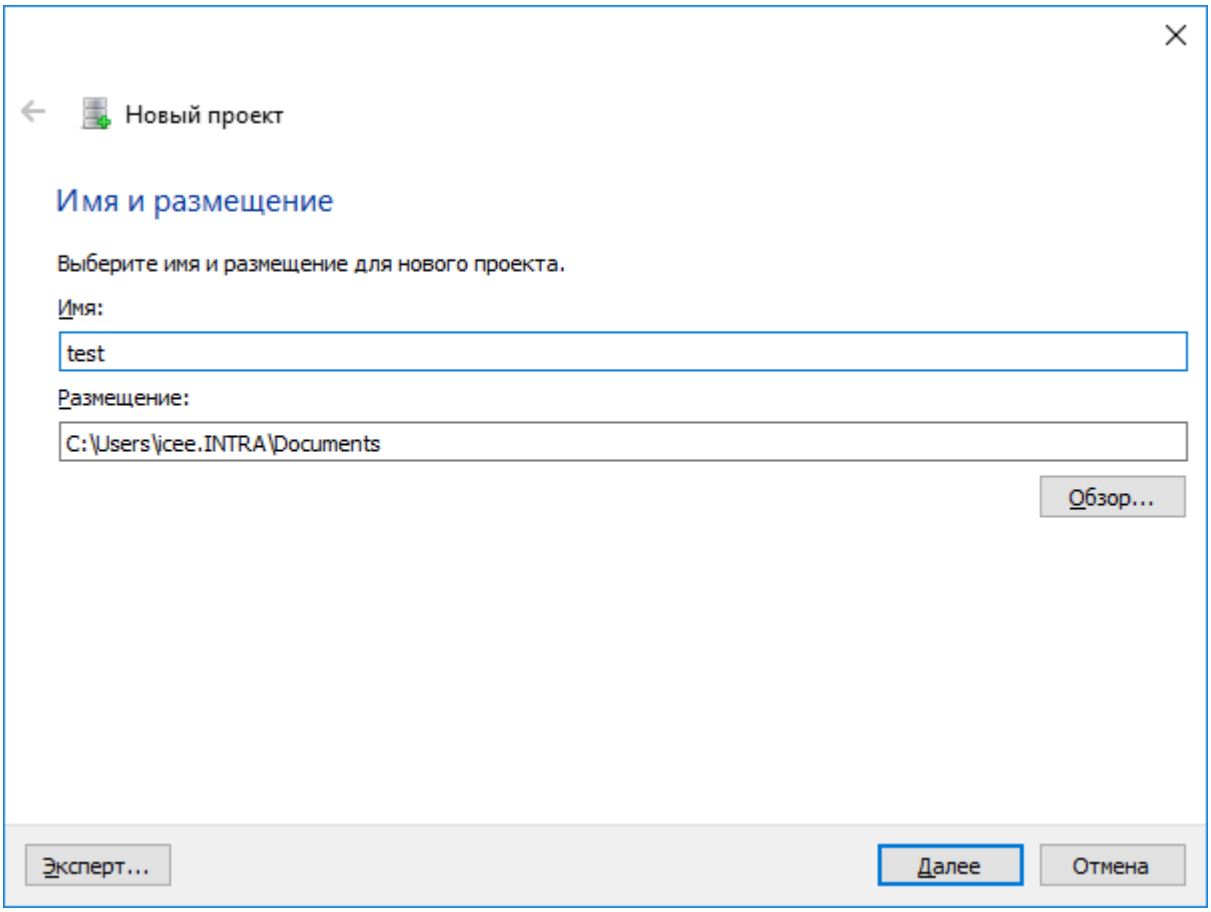

Рисунок 3 – Новый проект, шаг «Имя и размещения».

На шаге «Шаблон» выберите вариант *Импорт существующего динамического профиля*, т.к. работа будет вестись с предоставляемым в комплекте поставки ПО тестовым профилем (Рис. 4).

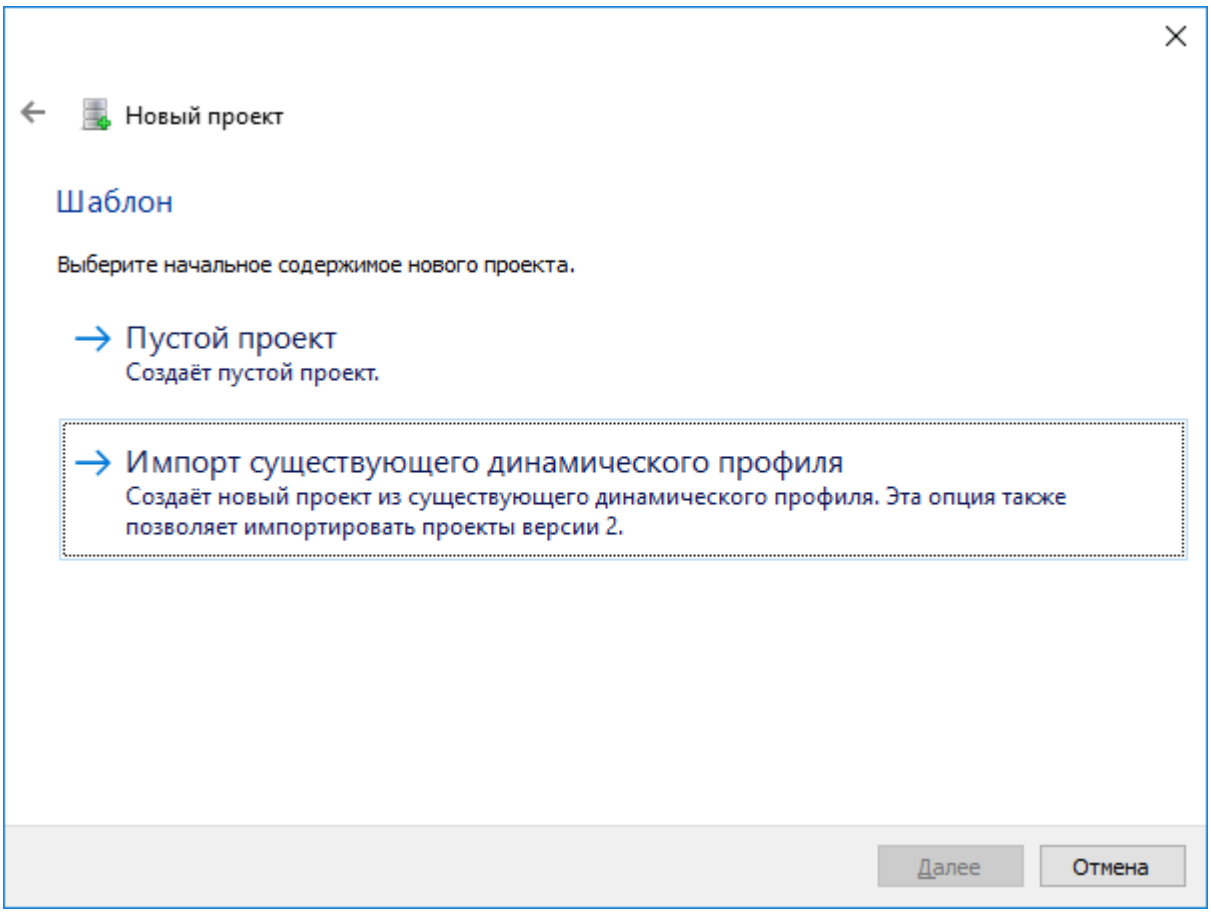

Рисунок 4 – Новый проект, шаг «Шаблон».

На шаге «Импорт существующего динамического профиля» с помощью кнопки *Обзор…* выберите файл *example.ini*, расположенный в подкаталоге *Examples\Profile* того каталога, куда была установлена среда ТРАЛ. После подтверждения выбора файла в нижней части окна текст «Манифест предыдущей версии проверен» (Рис. 5). Нажмите кнопку *Далее* для перехода к завершающему шагу.

На шаге «Сводка по проекту» подтвердите создание проекта нажатием кнопки *Завершить*. После непродолжительной подготовки проект будет открыт в основном окне (Рис. 6).

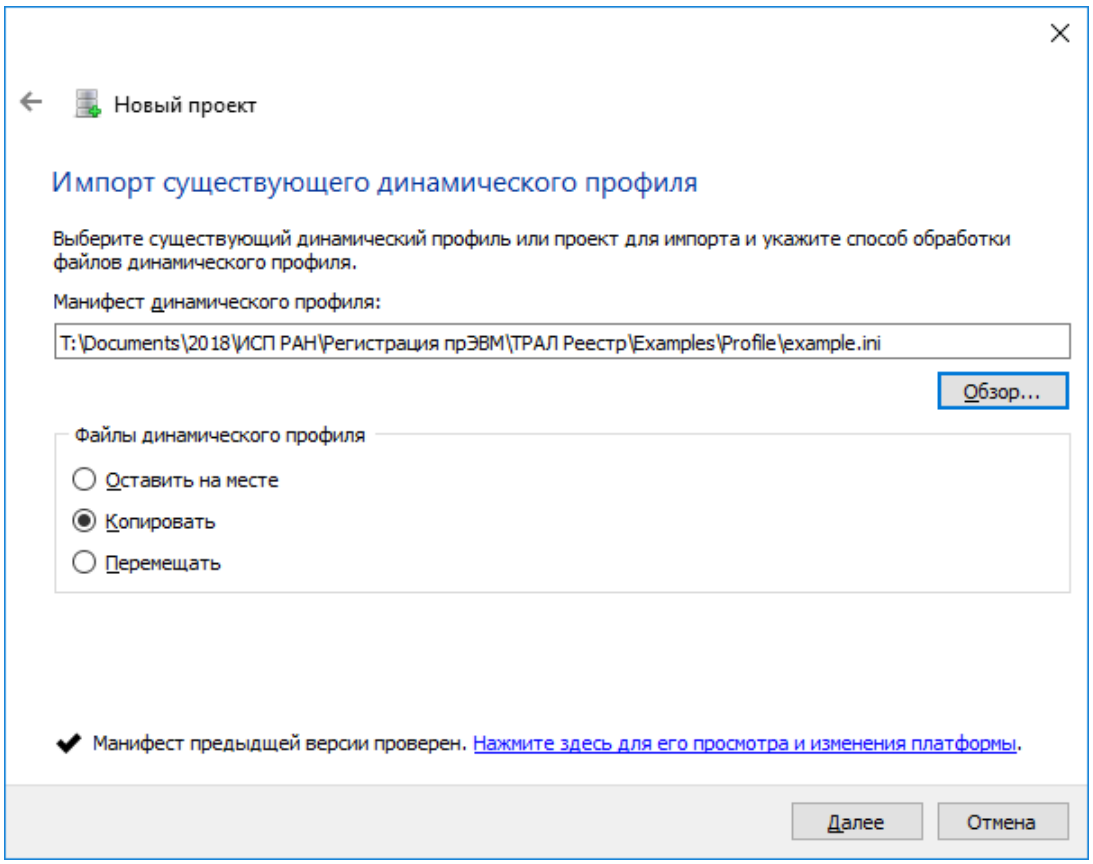

Рисунок 5 – Новый проект, шаг «Импорт существующего динамического профиля».

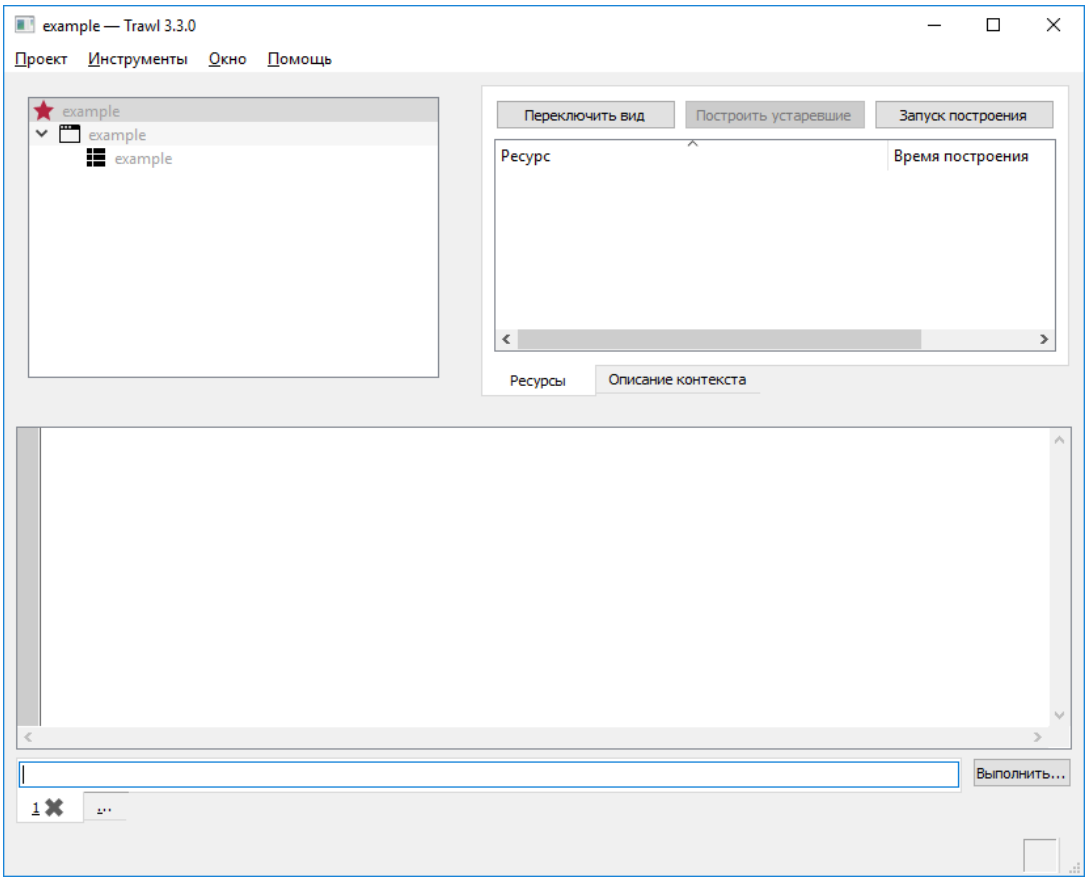

Рисунок 6 – Открытый проект в основном окне среды.

5.4 Предварительная обработка проекта

Далее необходимо запустить этап предварительной обработки проекта, заключающийся в автоматическом повышении уровня представления. Для исследуемых систем на базе Linux (данный профиль получен именно с такой системы) пользователю перед началом обработки необходимо указать используемый файл ядра, что связано с тем, что различные сборки ядра имеют различное расположение системных функций и структур данных. Для этого необходимо выбрать узел *example* второго уровня в дереве проекта и дважды кликнуть по нему (Рис. 7). В результате откроется окно набора профилей (Рис. 8). В этом окне следует выбрать пункт меню *Набор профилей* → *Импортировать образ ядра Linux…* и в появившемся диалоге задать путь к файлу *vmlinux* из каталога *Examples\Modules*, включённого в поставку среды анализа (Рис. 9), после чего нажать кнопку *ОК*. Настройки *Файл символов Linux* и *Адрес загрузки* менять не требуется. После этого окно набора профилей также можно закрыть.

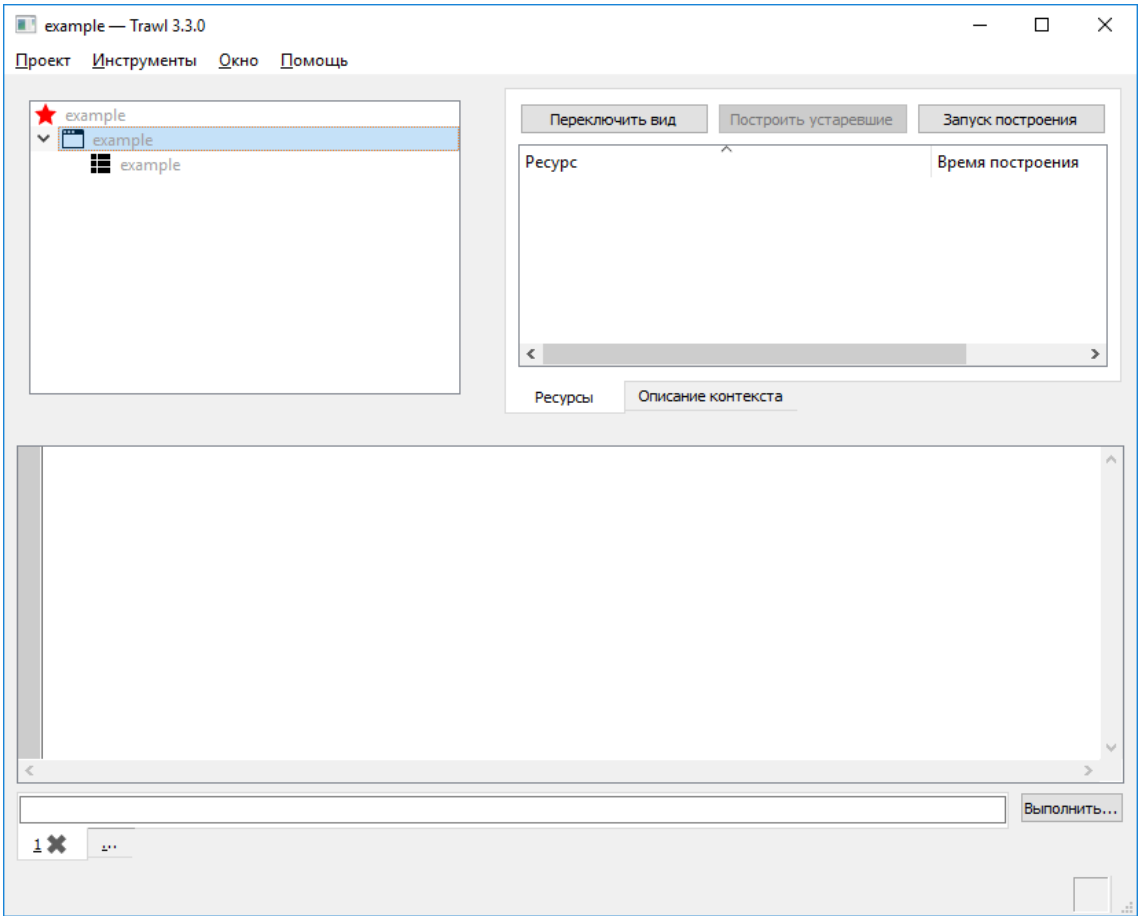

Рисунок 7 – Основное окно с выбранным узлом example второго уровня.

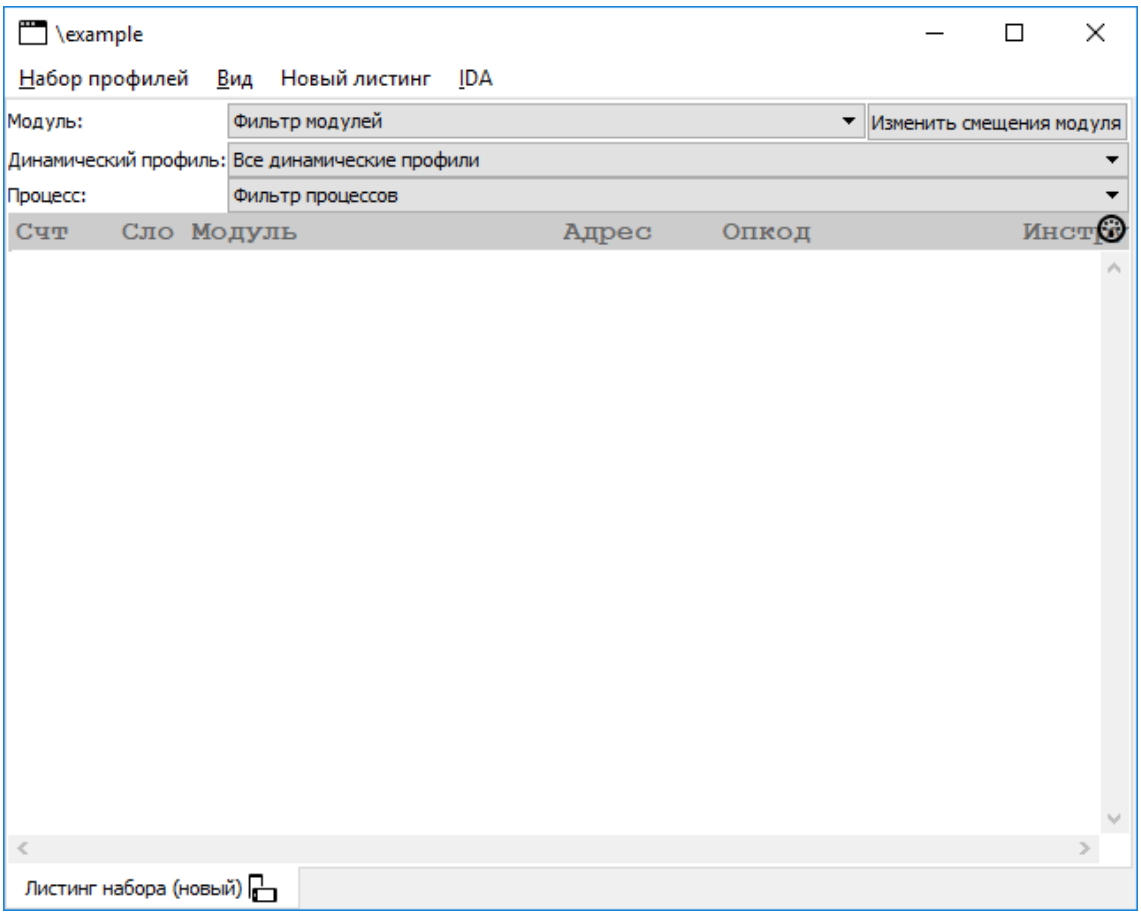

Рисунок 8 – Окно набора профилей.

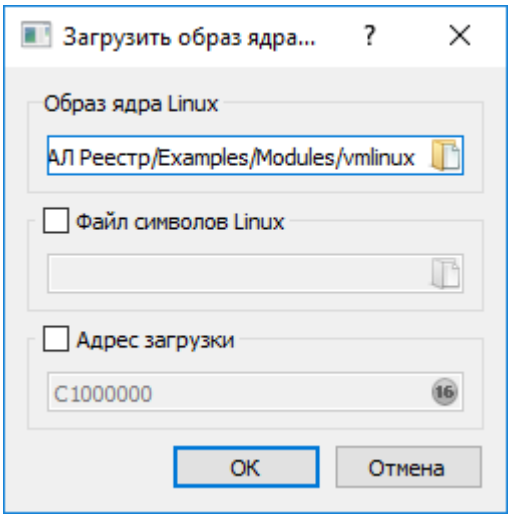

Рисунок 9 – Окно выбора образа ядра Linux.

Следующим шагом является запуск автоматического повышения уровня представления динамического профиля. В основном окне среды выберите пункт меню *Проект → Загрузить и предобработать проект…*. В открывшемся окне могут быть выбраны виды информации, которая будет построена в рамках повышения уровня представления. Для данного примера подходят значения по умолчанию, поэтому достаточно нажать кнопку *Начать построение* (Рис. 10).

При этом будут запрошены подтверждения на перестроение ресурсов «Информация MetaPDS» и «Информация профиля MetaPDS». В обоих случаях нужно ответить утвердительно.

Прогресс автоматической работы системы можно отслеживать в основном окне: в левой его части находится статус текущих задач (с указателем прогресса), а в нижней выводятся сообщения от алгоритмов. После завершения обработки (Рис. 11) двойным нажатием на узел *example* нижнего уровня в дереве проекта откройте окно динамического профиля. Окно содержит большой объём информации, поэтому его удобно развернуть на полный экран (Рис. 12).

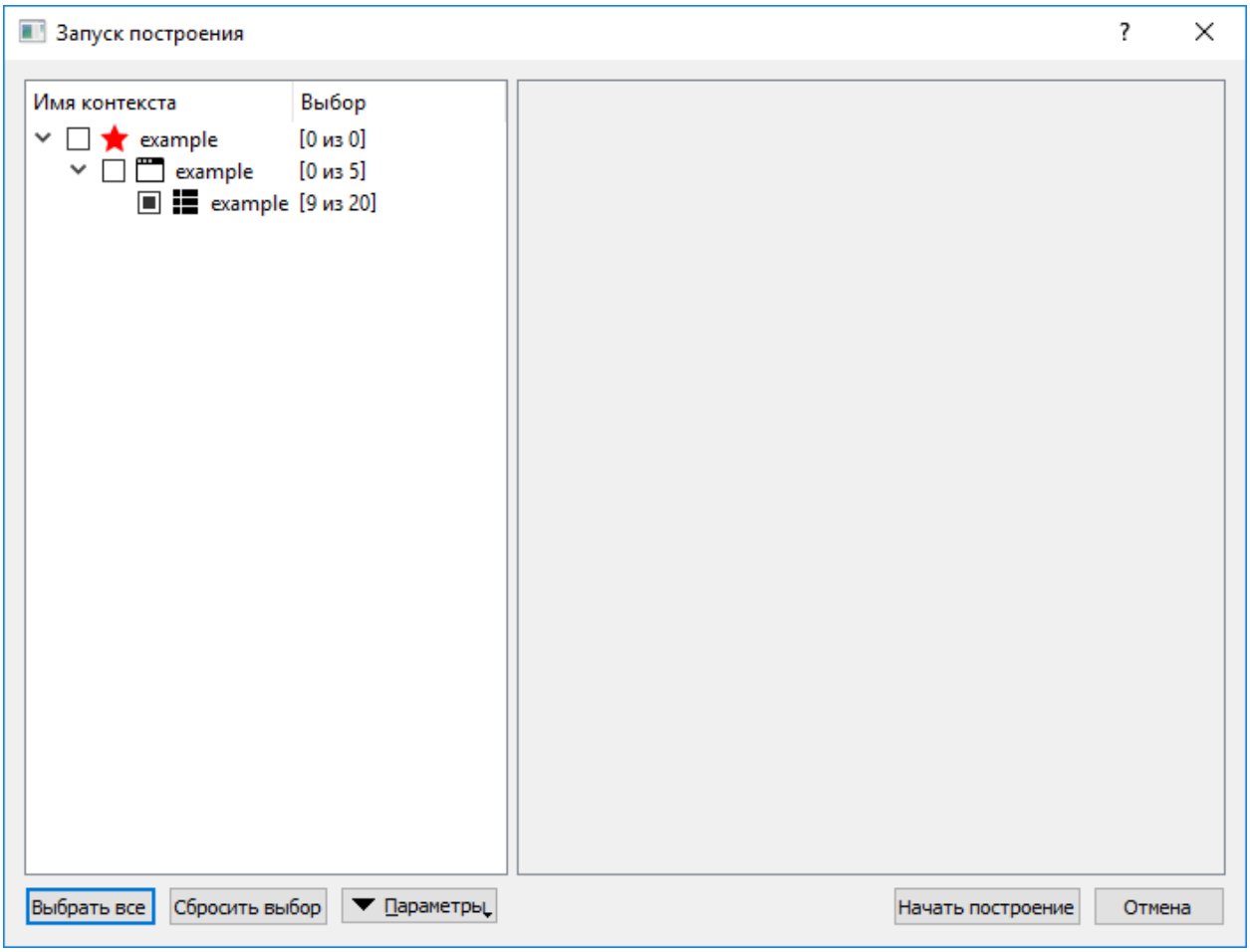

Рисунок 10 – Окно запуска построения.

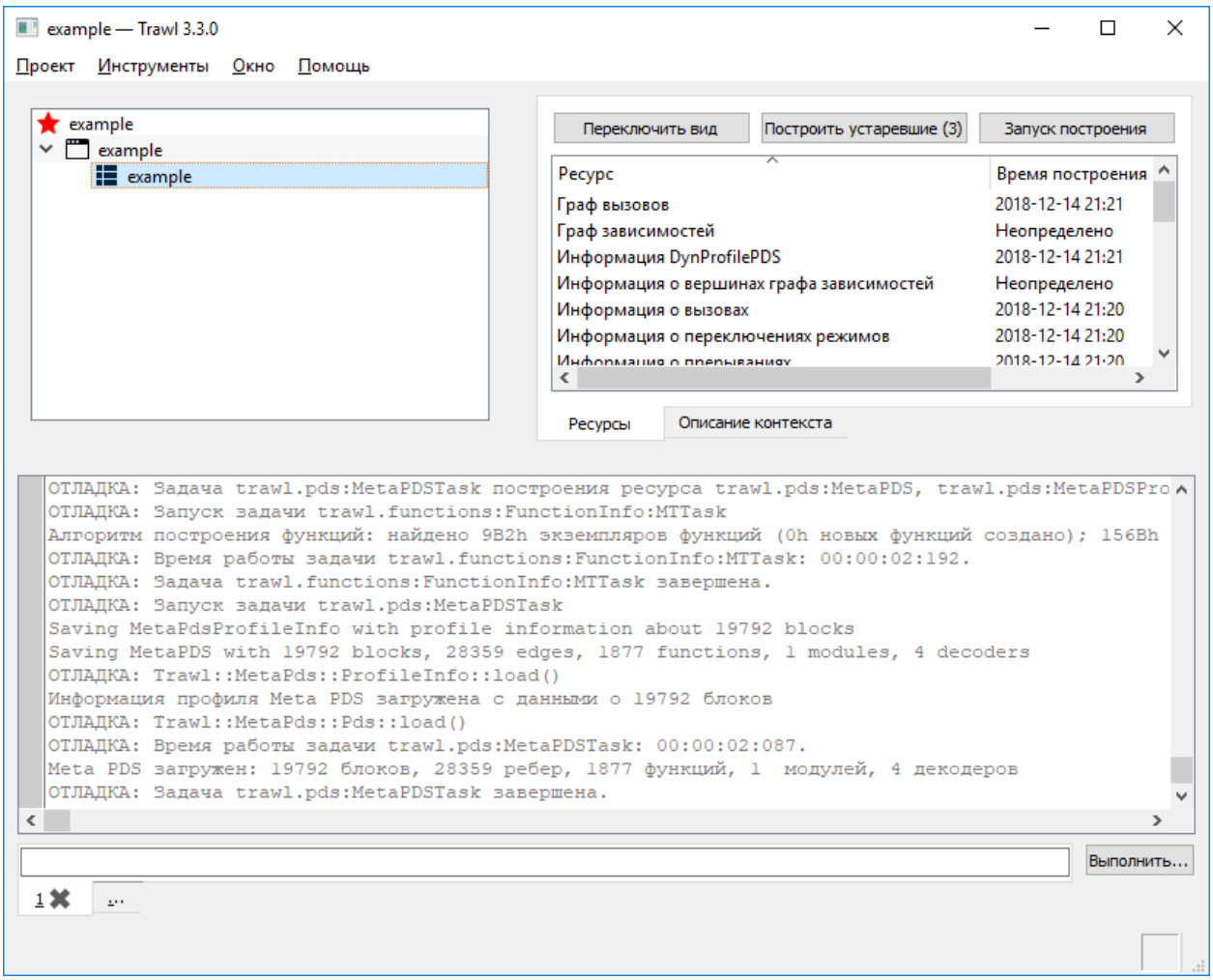

Рисунок 11 – Основное окно среды после завершения предобработки.

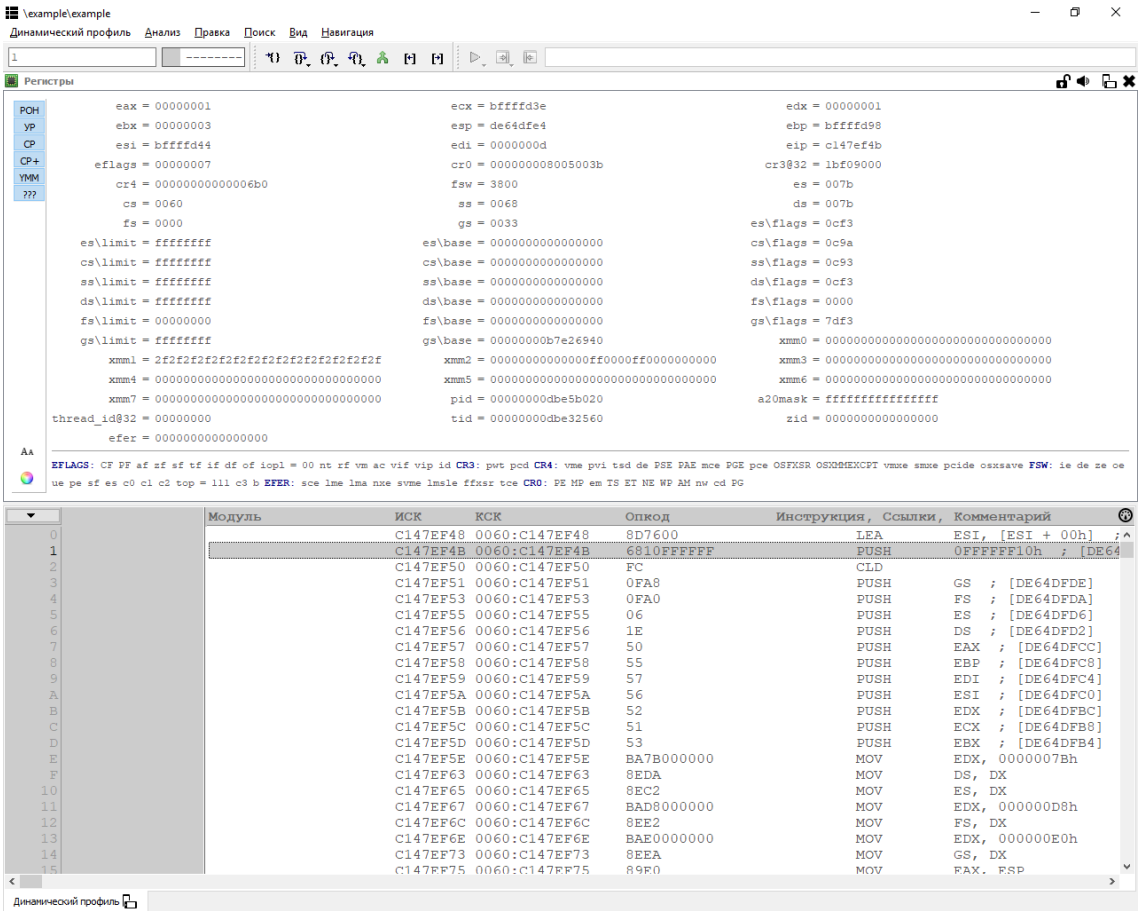

Рисунок 12 – Окно динамического профиля.

5.5 Распознавание модулей

Для того чтобы в окне динамического профиля появилась информация о бинарных модулях (исполняемых файлах и динамических библиотеках), необходимо предоставить среде анализа эти файлы. Для этого выберите пункт меню *Вид* → *Распознавание модулей*. Откроется встроенная в окно динамического профиля панель *Распознавание модулей* (Рис. 13). В верхнем ряду нажмите кнопку *Добавить каталог* (вторая слева) и выберите каталог *Examples\Modules*. Среда анализа распознает формат модулей, расположенных в этом каталоге, и появится их список (Рис. 14). Далее нажмите кнопку *Поиск* и дождитесь окончания работы алгоритма распознавания модулей (Рис. 15). Нажмите кнопку *Создать модули*. После этого панель *Распознавание модулей* можно закрыть.

На этом подготовительная работа для анализа динамического профиля завершена. Дальнейшие действия будут связаны с модельным сценарием анализа поведения программы, суть которого изложена далее.

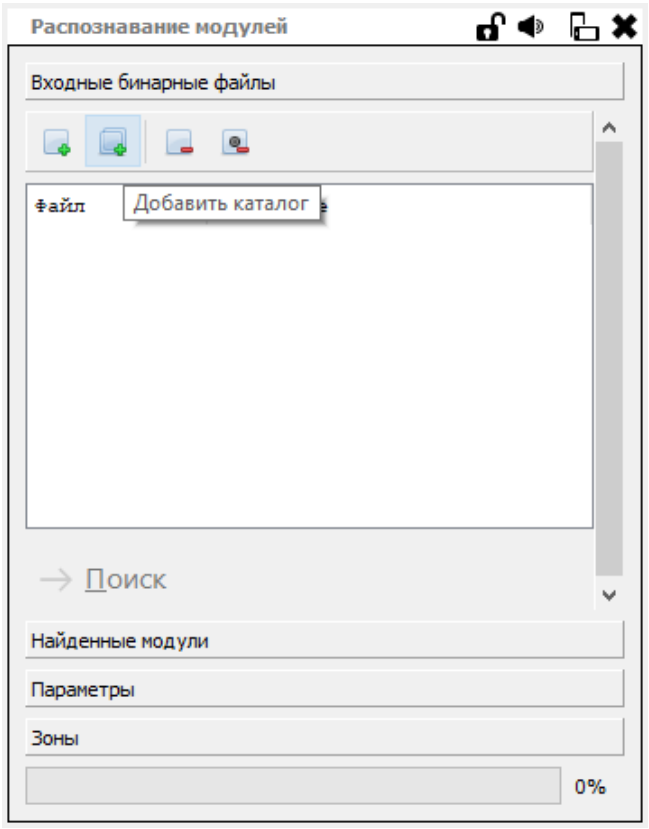

Рисунок 13 – Панель распознавания модулей.

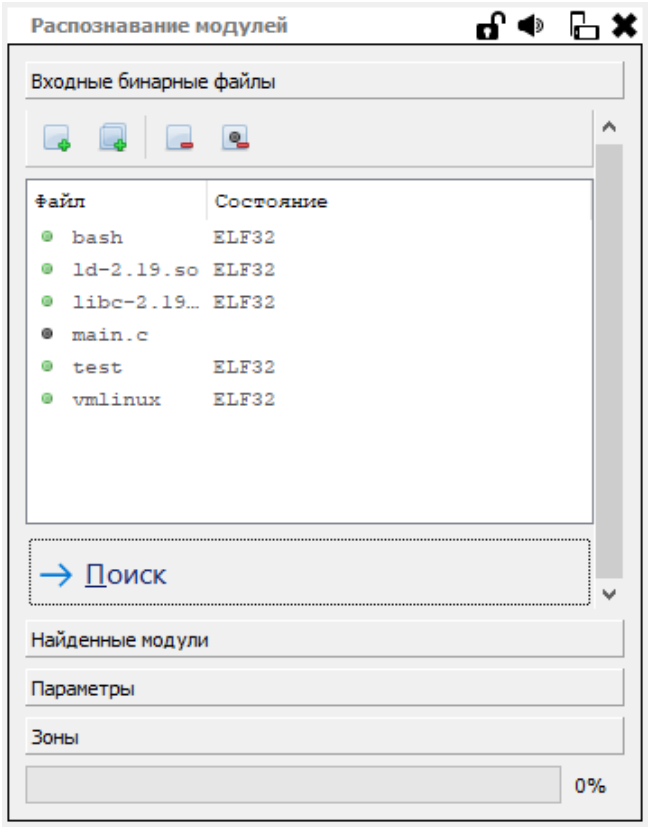

Рисунок 14 – Панель распознавания модулей после выбора каталога.

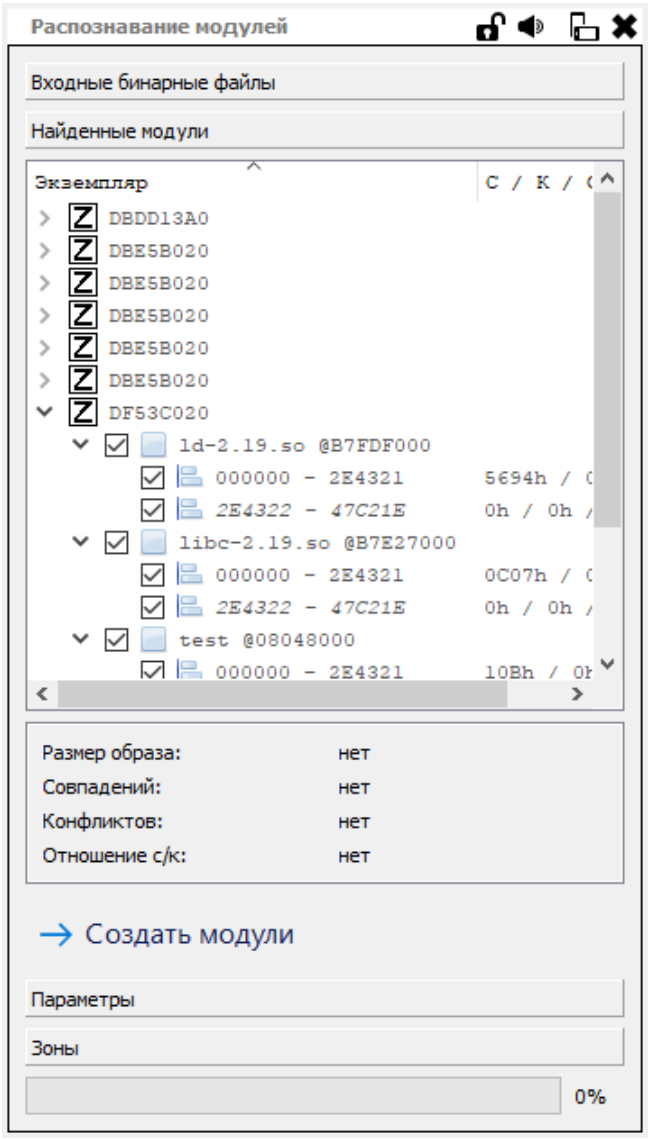

Рисунок 15 – Панель распознавания модулей после завершения поиска.

5.6 Сценарий анализа программы

Предлагаемый динамический профиль содержит выполнение программы *test* внутри ОС Linux (исходный код программы можно посмотреть в каталоге *Examples\Modules*). Программе на вход через параметры командной строки был подан набор данных, который привёл к реализации уязвимости переполнения буфера, перехвату потока управления и выполнению шелл-кода с запуском интерпретатора */bin/sh*. Таким образом, для исследования поведения программы в этом сценарии, необходимо выполнить следующие шаги:

- поиск кода анализируемой программы;
- поиск адресов размещения входных данных;
- отслеживание пути обработки входных данных;
- определение места срабатывания ошибки.

В дальнейших подразделах подробно разобран каждый из этих шагов.

5.7 Поиск кода анализируемой программы

Для того, чтобы найти в динамическом профиле шаги, на которых выполнялась интересующая программа, необходимо открыть панель *Вид → Экземпляры модулей*, найти в списке модуль *test* (Рис. 16) и в контекстном меню соответствующей строки списка (по правой кнопке) выбрать пункт *Найти в динамическом профиле*. После этого в окне профиля в качестве текущего шага будет выбран шаг с номером 2E2CD7, соответствующий первой команде, которая относится к исследуемому модулю. После перехода на этот шаг панель распознавания модулей можно закрыть.

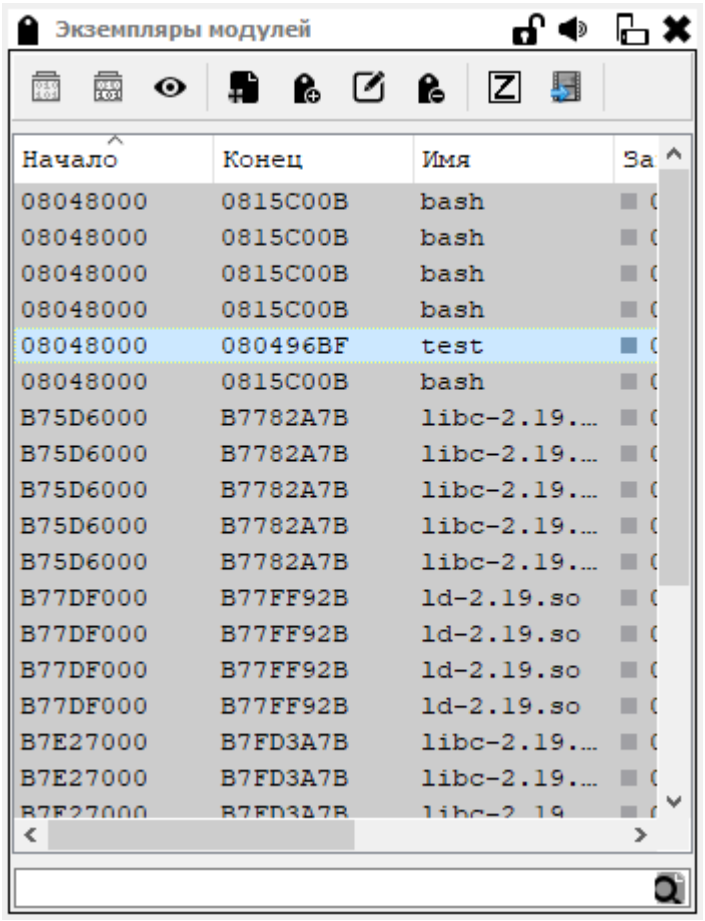

Рисунок 16 – Панель экземпляров модулей.

5.8 Поиск входных данных

В ОС семейства Linux параметры командной строки передаются в программу на стеке в специальном формате: сначала идёт число параметров, а затем указатели на параметры, при этом первый из указателей ссылается на путь к выполняемой программе.

Поскольку динамические профили не содержат данные оперативной памяти, для их получения необходимо воспользоваться алгоритмом восстановления содержимого памяти,

который в большинстве случаев позволяет восстановить интересующие значения на основе анализа команд, которые работали с этими значениям.

Выберите пункт меню *Вид → Восстановление содержимого памяти* и не меняя текущий шаг динамического профиля (2E2CD7) введите в поле *Регион памяти* выражение “v(esp, 16)” без кавычек. Остальные параметры менять не нужно. Нажмите кнопку с двунаправленной стрелкой (посередине) и дождитесь окончания работы алгоритма. После его завершения в нижней части панели будет отображено восстановленное содержимое памяти (Рис. 17).

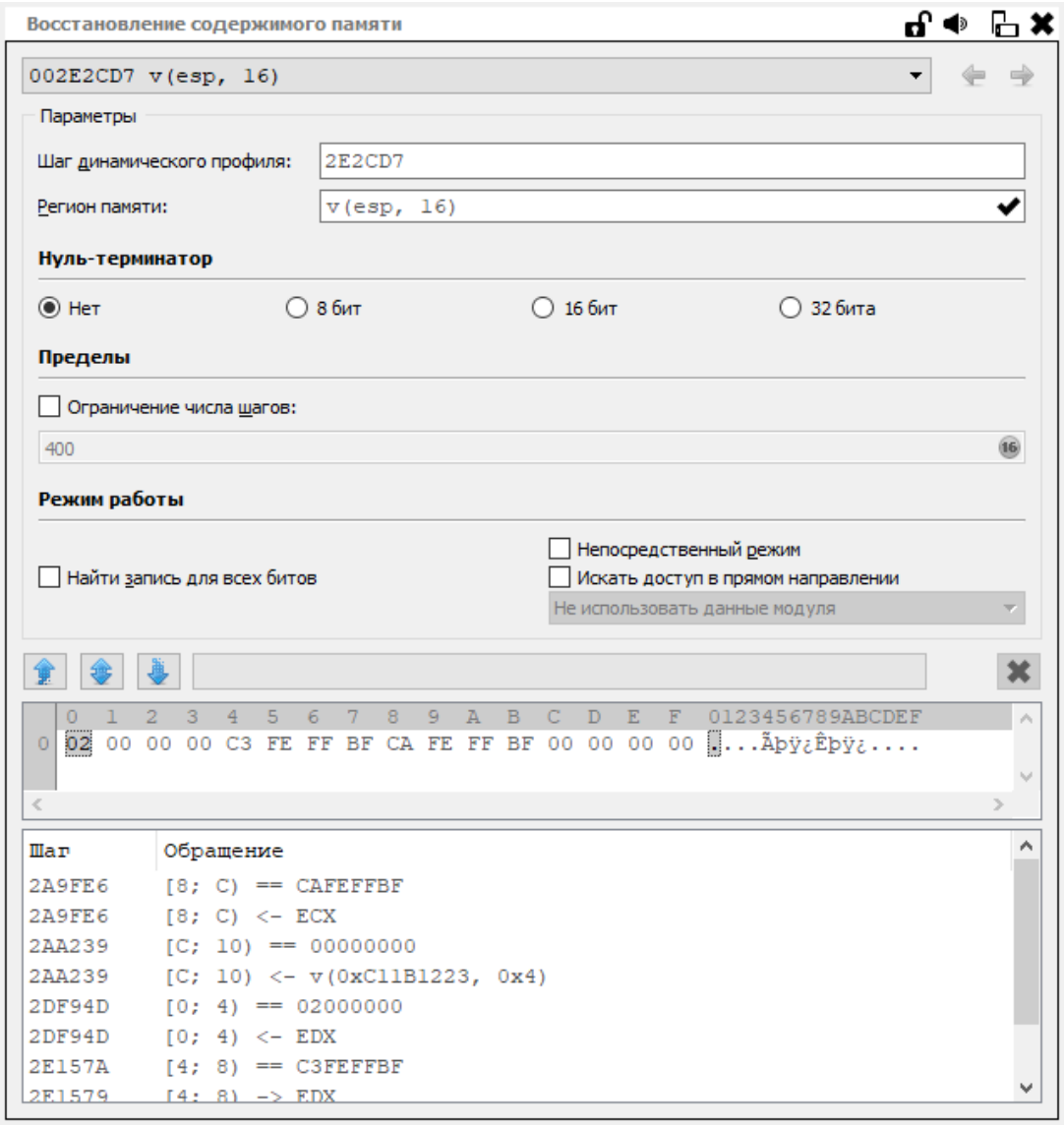

Рисунок 17 – Восстановление содержимого стека.

В общем случае регион памяти задаётся в виде выражения вида “v(*адрес*, *размер*)”, где *адрес* и *размер* могут быть как константами, так и значениями регистров и выражениями над ними (например, “v(esp + 8, edi)”). Буква *v* обозначает виртуальную адресацию.

По восстановленному содержимому стека видно количество параметров (2), указатель на нулевой параметр (0xBFFFFEC3) и на первый параметр (0xBFFFFECA). В дальнейшем нас будет интересовать длина переданной строки, соответствующей первому параметру, но известен только её базовый адрес в памяти. Для получения длины строки можно воспользоваться возможностью восстановления содержимого памяти до достижения нуль-терминатора. Для восстановления строки-параметра, таким образом, нужно задать выражение “v(0xBFFFFECA, 300)” в поле *Регион памяти*, а в переключателе *Нуль-терминатор* выбрать режим *8 бит*. Укажите эти параметры и запустите новое восстановление кнопкой с двунаправленной стрелкой. После окончания работы получим содержимое буфера на Рис. 18.

Длина 300 выбрана наугад: вместо неё можно указать любую другую длину, заведомо большую, чем предполагаемый размер буфера. Чтобы проверить, что длина выбрана верно, достаточно убедиться, что последний байт строки действительно нулевой.

Можно видеть, что в данном случае длина строки составляла 0x8C (без учёта нуль-терминатора), или 140 в десятичной системе счисления. Таким образом, входным данными исследуемой программы является буфер в памяти “v(0xBFFFFECA, 0x8C)”.

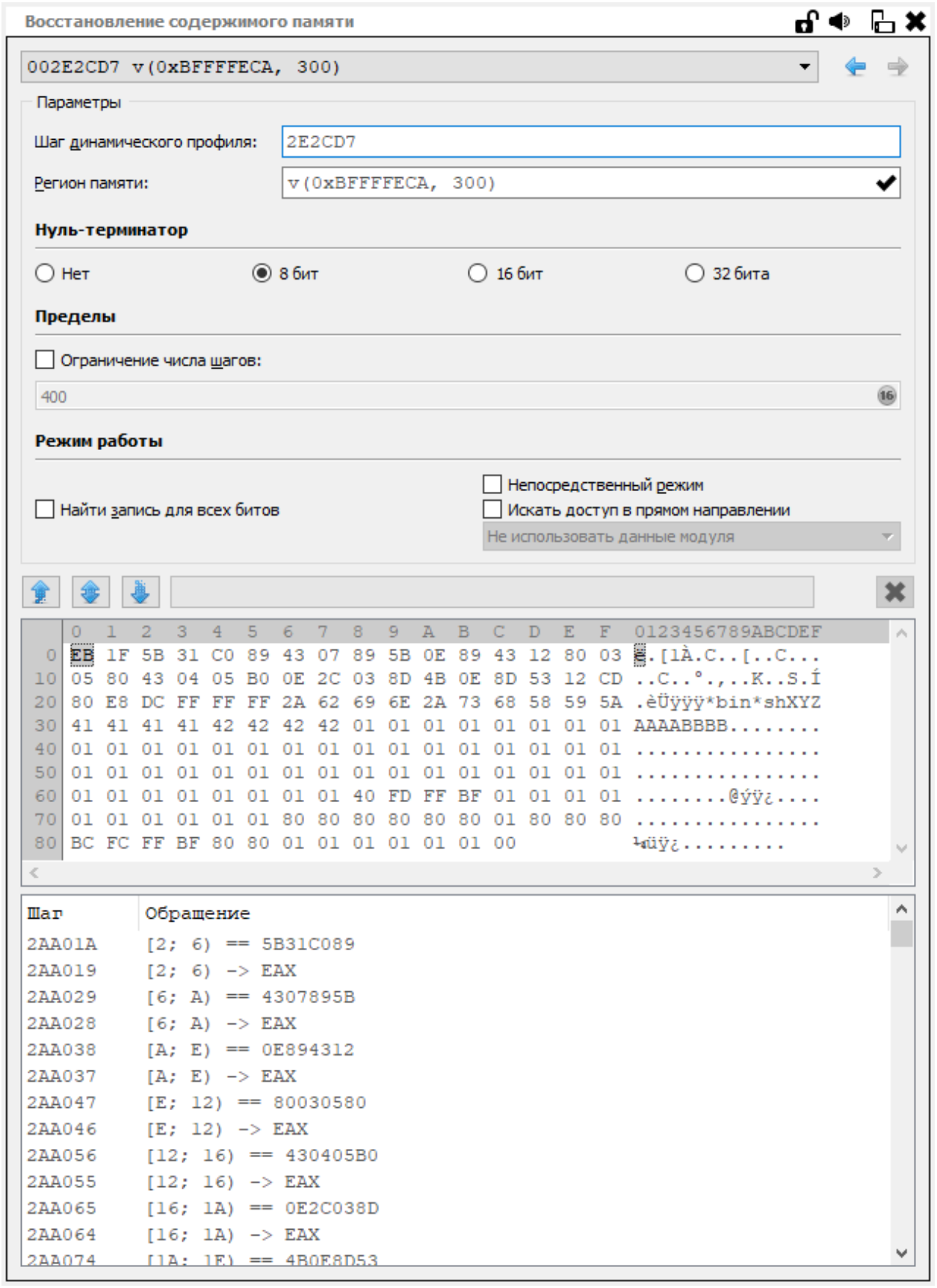

Рисунок 18 – Восстановление содержимого строкового параметра.

5.9 Отслеживание пути обработки входных данных

Для того, чтобы отобрать те шаги динамического профиля, команды в которых обрабатывали входные данные, необходимо воспользоваться алгоритмом динамического анализа помеченных данных. Для этого нужно в контекстном меню (по правой кнопке) на начальном шаге работы программы (т.е. на шаге 2E2CD7) выбрать пункт *Прямой анализ помеченных данных…* и в появившемся диалоговом окне задать следующие параметры (Рис. 19):

- *Имя*: input;
- *Элементы для отслеживания*: v(0xBFFFFECA, 0x8C);
- *Флаги*: *Флаговые зависимости* и *Добавить шаги, убирающие зависимости*.

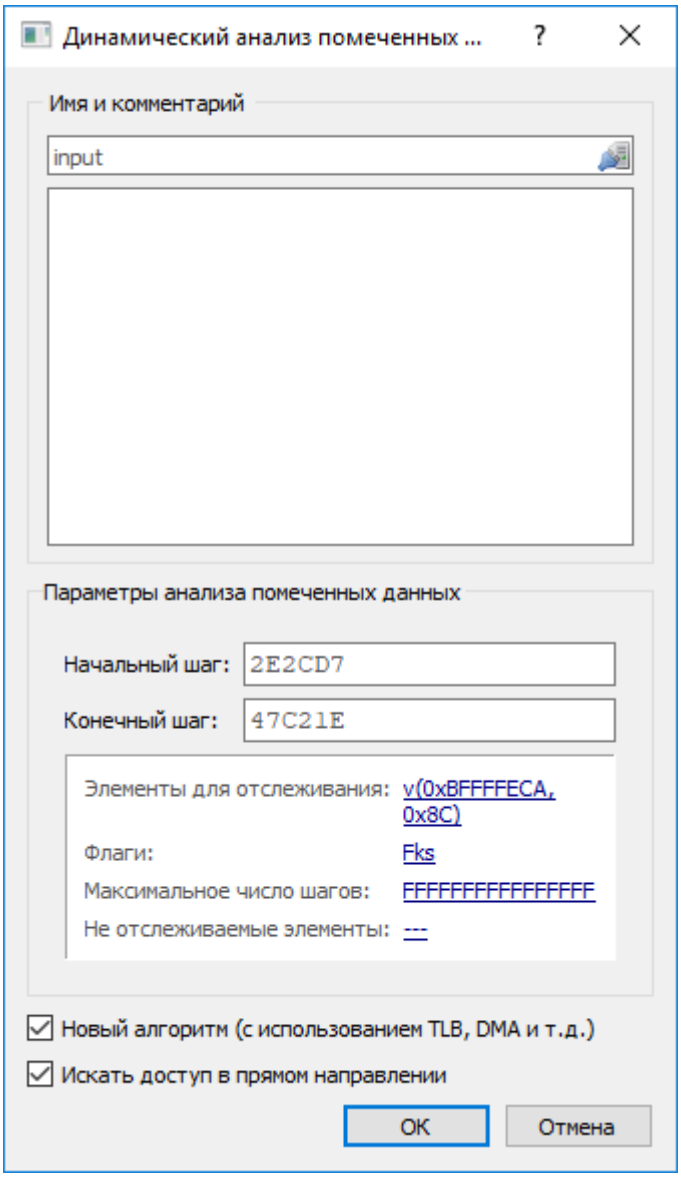

Рисунок 19 – Прямой динамический анализ помеченных данных.

Остальные значения можно оставить по умолчанию. После нажатия кнопки *OK* и завершения обработки в дереве проекта в основном окне среды появится новый узел четвёртого уровня с именем *input*. Соответствующий частичный динамический профиль нужно открыть так же, как и основной профиль: двойным нажатием на этом узле (Рис. 20).

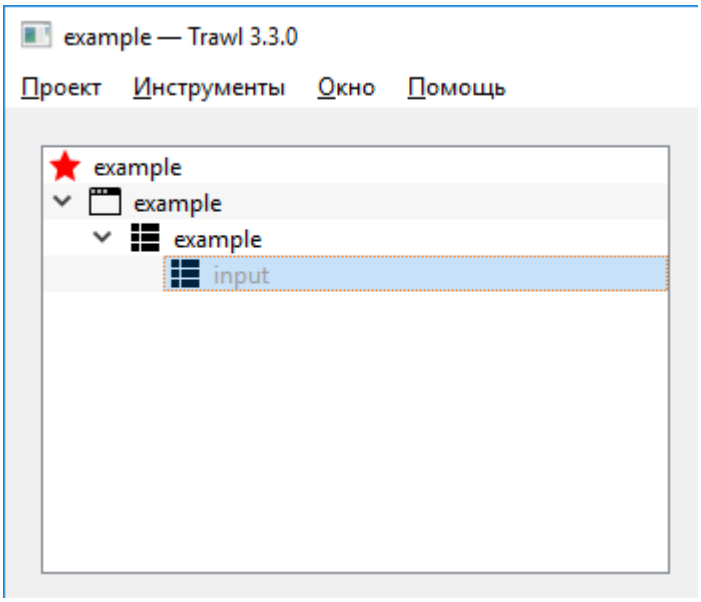

Рисунок 20 – Частичный профиль после завершения анализа помеченных данных.

5.10 Определение места срабатывания ошибки

Полученный частичный профиль (Рис. 21) содержит на порядок меньше шагов, чем полный. Поскольку данный частичный профиль был получен в результате прямого анализа помеченных данных (от входа), его необходимо анализировать, начиная с первых команд. При беглом просмотре частичного профиля можно обнаружить, что в первой части с данными происходят манипуляции внутри библиотеки *libc*, после чего данные обрабатываются кодом исследуемой программы, а затем обработка переход в ядро ОС.

Находящаяся на шаге 85 частичного профиля команда *RET* не предназначена для обработки данных, а выполняет функцию передачи управления (в данном случае – выхода из подпрограммы). В данном случае её появление в профиле – это точка срабатывания ошибки, т.к. она указывает на то, что входные данные повлияли на поток управления программы. Для того, чтобы рассмотреть команды в ближайшей окрестности команды *RET*, необходимо вернуться в окно полного динамического профиля. Удобнее всего это сделать, включив синхронизацию шагов частичного и полного профиля. Для этого нужно вызвать окно настройки соединений (*Вид* → *Соединения…*) и установить две галочки напротив пункта *\example\example* (Рис. 22). Обратите внимание, что для этого нужно, чтобы были открыты оба окна: и окно полного профиля, и окно частичного профиля.

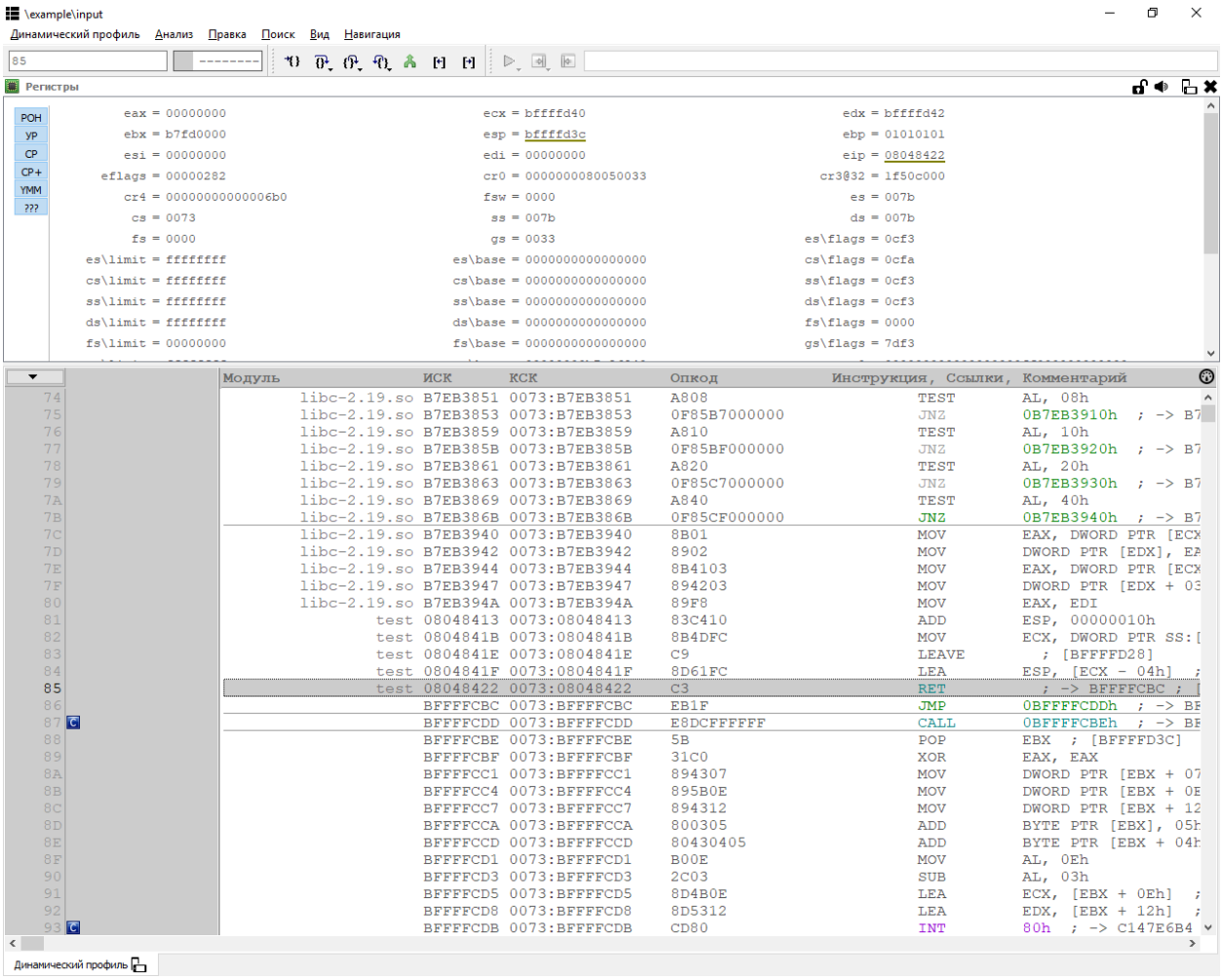

Рисунок 21 – Фрагмент частичного динамического профиля.

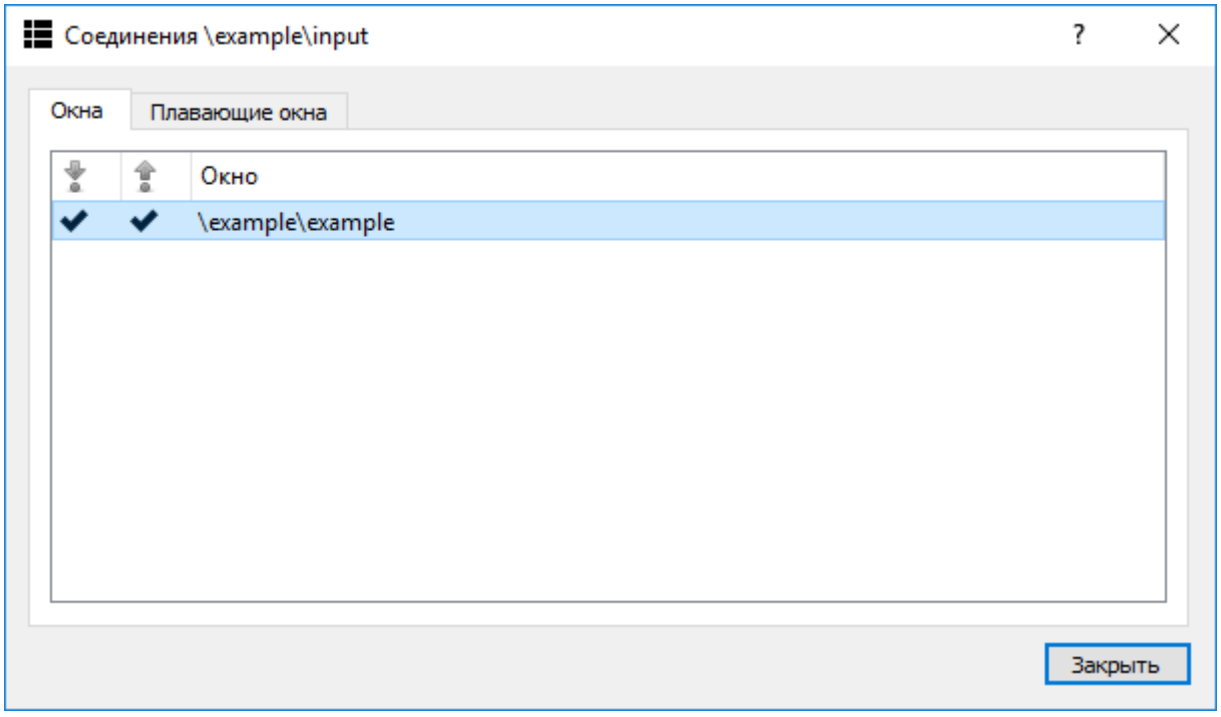

Рисунок 22 – Настройка соединений.

С данного момента включена синхронизация шагов: перемещение в частичном профиле приведёт к перемещению в полном и наоборот (при условии наличия соответствующего шага в частичном профиле). Установите в частичном профиле текущий шаг на позицию 84 (сразу перед выполнением команды *RET*) и перейдите в окно полного профиля. В нём автоматически будет выбран шаг 2E4320 (Рис. 23).

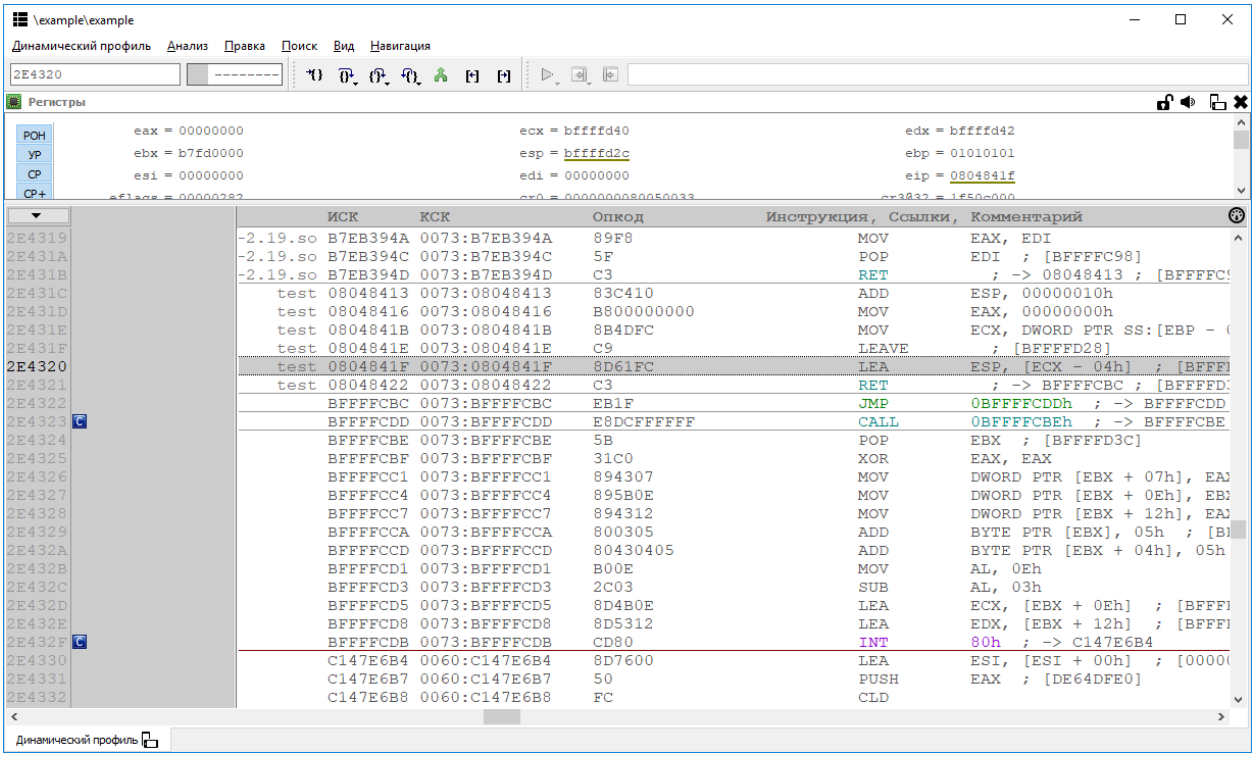

Рисунок 23 – Окрестность команды RET.

В данном случае аналитик может провести исследование последствий перехвата потока управления. Сразу после команды *RET* выполняется код, лежащий на стеке (в адресах вида 0xBFFFF*xxx*). Этот код обращается к ядру Linux через системный вызов *INT 80h*. Перед вызовом готовятся параметры: в регистре *EAX* оказывается значение 0x0B, что соответствует системному вызову execve. На регистрах *EBX*, *ECX*, *EDX* формируются параметры системного вызова. Восстановление содержимого памяти по буферу v(EBX, 0x100) на шаге 2E432F с 8-битным нуль-терминатором позволяет получить строку /bin/sh, которая подтверждает срабатывание шелл-кода (Рис. 24).

5.11 Заключение

В рассмотренном тестовом сценарии была проанализирована сработавшая уязвимость в программе, которая привела к перехвату управления с вызовом шелл-кода. Данный пример показывает возможности среды анализа бинарного кода ТРАЛ по изучению потоков

данных и управления и демонстрирует некоторые из встроенных в среду алгоритмов, а также позволяет оценить её графический интерфейс и скорость работы.

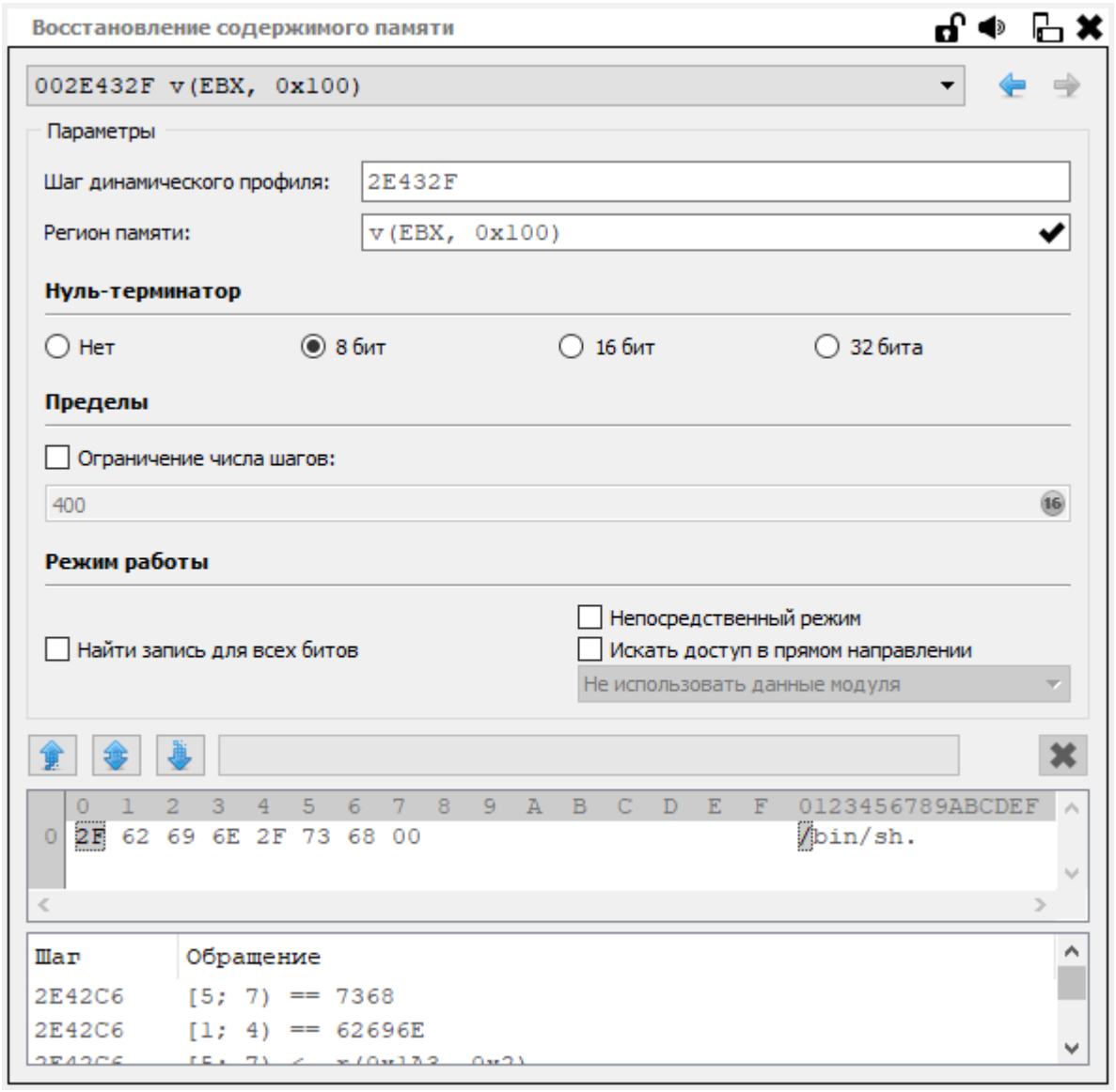

Рисунок 24 – Значение параметра системного вызова execve.

6 Процессы, обеспечивающие поддержание жизненного цикла среды анализа бинарного кода ТРАЛ

6.1 Процессы разработки и совершенствования ПО

Разработка среды анализа бинарного кода ТРАЛ ведется по методологии Agile с привлечением современных средств повышения качества кода.

1) Для хранения кода используется система контроля версий git, и изменения в основной ветке проходят инспекцию кода (code review) другими разработчиками.
2) Для проверки работоспособности системы созданы тесты, причем используются как модульные тесты, проверяющие функционал отдельных компонентов, так и интеграционное тестирование, при котором автоматические сценарии проверяют корректность работы системы при типичных вариантах использования. Среди таких сценариев – снятие динамического профиля, предварительная обработка и создание вспомогательной информации по динамическому профилю, запуск различных алгоритмов анализа и сравнение результатов с эталонными.
3) При разработке используется практика непрерывной интеграции (continuous integration): при помощи сервера тестирования Jenkins происходят регулярные автоматические сборки с последующим запуском тестов, упомянутых в предыдущем пункте.
4) При написании кода разработчики должны придерживаться строгих правил оформления кода; нарушение этих правил не позволит пройти этап инспекции кода.
5) Периодически производится проверка кода среды на предмет наличия ошибок статическим анализатором Svace, разработанным в ИСП РАН.

6.2 Поддержка пользователей ПО

Установка на рабочей станции дистрибутива среды анализа бинарного кода ТРАЛ не требует настройки ОС, установки внешнего ПО или каких-либо других дополнительных действий.

Пользователи, которым достаточно для решения их задач возможностей уже реализованных в среде анализа алгоритмов, могут приступать к работе со средой после краткого обучения, проводимого на стороне разработчика (ИСП РАН). В ходе своей работы среда ТРАЛ автоматически ведёт журналы действий и ошибок, которые могут быть отправлены разработчику в случае обнаружения некорректного поведения или возникновения запроса на улучшение и доработку.

Имеется возможность расширить обучение пользователей вопросами самостоятельной разработки модулей расширений. Для обеспечения независимой разработки дополнительно передается комплект заголовочных файлов, библиотек и документации. Помимо того, ИСП РАН предлагает разработку специализированных модулей анализа на своей стороне по запросу заказчика.

Информация о сбоях в работе среды ТРАЛ, проблемах производительности, ошибках целевого функционала передаются пользователями среды непосредственно ответственным сотрудникам ИСП РАН, без использования публичных Интернет-ресурсов управления ошибками. Это обеспечивает должный уровень конфиденциальности для контрольных примеров (фрагменты исследуемых динамических профилей, снимки памяти), передаваемых пользователем среды в ИСП РАН для оценки и исправления программного дефекта.

Со своей стороны ИСП РАН постоянно совершенствует среду анализа, применяя в жизненном цикле разработки передовые методики. Добавление новых и улучшение существующих алгоритмов ведется в инициативном порядке. Обновления среды ТРАЛ передаются пользователям среды через согласованные с ними каналы распространения обновлений.

6.3 Необходимый персонал для разработки и поддержки

Для разработки и поддержки программного продукта необходима соответствующая квалификация разработчиков. Это вызвано следующими причинами.

1) Специфическая предметная область, требующая глубоких знаний одновременно в нескольких областях: устройство современной аппаратуры и операционных систем, компиляторные технологии, компьютерная безопасность, технологии разработки ПО.
2) Наличие среди алгоритмов, реализованных в среде анализа, алгоритмов, впервые разработанных сотрудниками ИСП РАН.
3) Требования к производительности системы, из-за чего необходимо применять эффективные алгоритмы, в т.ч. хорошо масштабируемые по нескольким вычислительным ядрам.

В силу приведенных причин коллектив разработчиков ТРАЛ формируется из специалистов, получивших профильное образование: выпускников ВМК МГУ (магистерская программа «Компиляторные технологии») и ФУПМ МФТИ (магистерская программа «Системное программирование»).

Для гарантийного обслуживания задействовано 3 научных сотрудника, для Технической поддержки задействованы 3 научных сотрудника, для модернизации программного обеспечения задействованы 5 научных сотрудников и инженеров.

Адрес электронной почты, по которому можно обратиться по вопросам, связанным со средой анализа бинарного кода ТРАЛ — trawl@ispras.ru.