# Методы и программные средства, поддерживающие комбинированный анализ бинарного кода ⋆

*В.А. Падарян, А.И. Гетьман, М.А. Соловьев, М.Г. Бакулин, А.И. Борзилов, В.В. Каушан, И.Н. Ледовских, Ю.В. Маркин, С.С. Панасенко*
*{ vartan, thorin, eyescream, bakulinm, helendile, korpse, il, ustas, spanasenko}@ispras.ru*

**Аннотация.** В статье рассматриваются разработанные в ИСП РАН методы и инструменты анализа бинарного кода и их применение к задачам восстановления алгоритмов и форматов данных. Предметом анализа выступает исполняемый код различных процессорных архитектур общего назначения в отсутствии исходных кодов, отладочной информации и привязки к определенным версиям операционных систем. Подход состоит из сбора детальной трассы выполнения уровня машинных команд; метода последовательного повышения уровня представления; выделения кода алгоритма и последующей структуризации как кода, так и форматов обрабатываемых данных. Были достигнуты важные результаты: разработано промежуточное представление, позволяющее проводить большую часть предварительных обработок и выделение кода алгоритма без привязки к особенностям определенной машины; разработан метод и инструмент автоматизированного восстановления форматов сетевых сообщений и файлов. Разработанные инструменты были интегрированы в единую среду анализа, поддерживающую совместное их использование; архитектура среды также описана в статье. Приводятся примеры применения к реальным программам.

**Ключевые слова:** бинарный код, статический анализ, динамический анализ, интегрированная среда.

## *1. Введение*

На протяжении нескольких лет в ИСП РАН ведутся работы по разработке интегрированной среды анализа бинарного кода. Потребность в такой среде возникает у любого специалиста, пытающегося провести анализ исполняемого (бинарного) кода достаточно большой программы с целью достичь понимания ее устройства. Последующее применение достигнутого понимания зависит от того, в какой прикладной области работает специалист. Достаточно типичными ситуациями, когда требуется понять устройство программы по ее

⋆ Работа поддержана грантом РФФИ 11-07-00450-а



бинарному коду, являются: изучение вредоносного кода разработчиками антивирусов, оценка свойств алгоритмов во время сертификационных испытаний, восстановление закрытых протоколов для их последующей реализации в программах с открытым исходным кодом, выявление ошибок и выделение среди них уязвимостей, и т.п.

Распространенная ранее практика предполагала применение интерактивного дизассемблера (фактически безальтернативным инструментом до сих пор является IDA Pro) и интегрированного отладчика. Из-за легкости обнаружения отладчиков, работающих в одном с исследуемой программой окружении, предпочитают отладчики, работающие в виртуальных машинах. Связка из IDA Pro и отладчика позволяет справляться с некоторыми базовыми проблемами анализа бинарного кода, например, различать в рамках отдельного выполнения код и данные, определять целевые адреса в случае косвенной адресации. Не решив последнюю задачу, невозможно гарантировать корректное представление изучаемой программы даже в виде декодированных машинных команд.

Приведенный способ анализа имеет принципиальное ограничение: он позволяет изучать только небольшие фрагменты кода, автоматизация работы в этом случае крайне затруднительна. В результате проведенных ранее исследований был предложен метод динамического анализа [1, 2], позволяющий преодолеть указанное ограничение, а также и некоторые другие, характерные для интерактивной отладки. Идея метода заключается в получении детальной трассы выполнения на уровне машинных команд и ее последующем глубоко автоматизированном анализе. В последующих работах метод был развит [3, 4] за счет соединения динамического и статического анализа с целью взаимной компенсации слабых сторон этих подходов, когда они применяются по отдельности.

Метод был реализован в интегрированной среде анализа (СА) бинарного кода (рис. 1). В качестве средства получения трассы, как правило, используется виртуальная машина QEMU, поскольку в ней был реализован механизм детерминированного воспроизведения [5], позволяющий расширить классы анализируемых программ. Реализованные в СА алгоритмы поэтапно поднимают уровень представления, восстанавливая события уровня ОС и системы программирования, и сохраняя полученные результаты в виде разметки трассы. На основе собранных и размеченных трасс строится статико-динамическое представление, отражающее изменение кода с течением времени. Имеется возможность экспортировать это представление в IDA Pro, но и сама СА обладает широкими возможностями, опирающимися на результаты трассировки. К их числу относятся: навигация в трассах и статическом коде, выделение кода отдельных алгоритмов, восстановление структуры этих алгоритмов и форматов обрабатываемых данных, описание функций и построение модельных примеров, содержащих извлеченные из бинарного кода реализации алгоритмов.

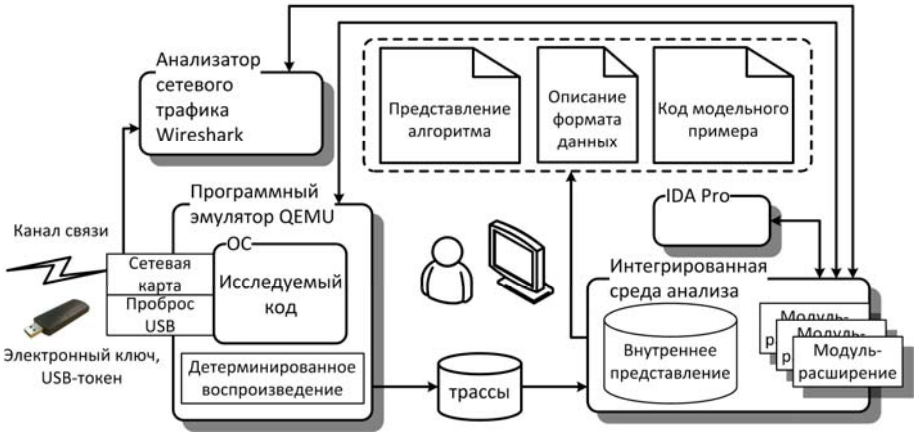


*Рис. 1. Внешняя архитектура среды анализа бинарного кода.*

Основные представленные в статье результаты затрагивают следующие вопросы: поэтапное повышение уровня представления (раздел 2), которое выполняется в автоматическом режиме при импорте в CA только что записанной трассы; выделение алгоритма на основе динамического слайсинга бинарного кода (раздел 3); и восстановление форматов данных (раздел 4). В Разделе 5 описана внутренняя архитектура CA.

## 2. Предварительное повышение уровня представления

Типовой ситуацией является задача анализа бинарного кода приложения, работающего под управлением известной ОС (Windows, Linux, BSD и т.п.). Как уже упоминалось выше, инструментами анализа будут выступать IDA Pro и отладчик. Неизбежные затруднения в данном случае будут обусловлены косвенной адресацией и размером анализируемого кода. Поскольку предполагается, что анализ происходит не на этапе разработки ПО, исполняемый файл не содержит никакой отладочной информации, доступен только объектный (машинный) код и, возможно, таблицы экспортируемых функций. Источником знаний о том, что делается программой, является семантика машинных команд, вызовы библиотечных функций, в том числе API ОС и библиотек поддержки времени выполнения. Используется неявное предположение, что вся функциональность заключена в одном пользовательском процессе.

Расширение класса анализируемых программ сразу же вызывает нарастающие технические трудности. Простая задача – анализ в IDA Pro кода программы, состоящей из нескольких модулей, – требует либо получения снимка



адресного пространства процесса в некоторый момент времени, причем необходимо определить, в какой именно, либо дополнения базового функционала IDA Pro. Более тяжелые задачи возникают при анализе поведения вредоносного кода, например, когда приходится отслеживать его распространение в адресных пространствах нескольких процессов, памяти ОС.

Следующим направлением для расширения класса задач является анализ кода BIOS, гипервизоров, систем, перехватывающих обращения ОС к аппаратуре, например, систем полнодискового шифрования, а также самих ОС, когда они представляют собой модифицированную известную или неизвестную закрытую ОС. В этом случае нет гарантий корректной интерпретации изучаемого кода по вызовам известных библиотечных функций, поскольку этих функций либо нет, либо они сами являются предметом анализа.

Следует еще упомянуть, что исследуемый код, как правило, не является «замкнутым». Например, вредоносное ПО обычно взаимодействует с управляющими серверами, от которых получает команды, код «полезной» нагрузки, а также передает на эти сервера данные со скомпрометированных компьютеров.

Все эти, а также и другие особенности различных классов программ учитывались при разработке подхода к анализу. Была поставлена цель создать максимально общий подход, подходящий для разных процессорных архитектур, разных типов устройств и разных классов программ. Такие требования привели к тому, что разработанный подход способен довольствоваться одной только семантикой выполняющихся машинных команд (см. рис. 2): происходит постепенное повышение уровня представления до уровня, когда восстановленный алгоритм может быть передан для последующего анализа прикладному специалисту, не владеющему знаниями об особенностях реализации. Типовым выходным представлением является двудольный граф, где один класс вершин – обрабатываемые данные, второй класс – функции (блоки кода) эту обработку выполняющие.

Максимально низкий начальный уровень анализа – главное отличие предложенного подхода от известных аналогов, например, BitBlaze [6] или DroidScope [7], где предметом анализа выступают трассы, содержащие не только машинные команды, либо анализ ограничен адресным пространством только одного процесса. При полносистемном анализе в трассирующей виртуальной машине, как правило, разработанной на основе QEMU [7, 8], работает зафиксированная версия известной ОС, и знание о ее внутреннем устройстве активно используется.


*Рис. 2. Различные уровни представления и диапазоны применения методов анализа.*

Платой за общность является необходимость выполнять предварительное повышение уровня представления (рис. 3), заканчивающееся восстановлением потоков данных и управления. Метод предварительного повышения не требует каких-либо априорных знаний об устройстве ОС, работающей в анализируемой системе, и пригоден для большинства процессорных архитектур общего назначения. Более того, метод применим в случае модифицированной версии известной ОС, и даже в случае анализа кода неизвестной ОС.

Устранение артефактов требуется вследствие особенностей применяемых средств трассировки. Примерами дефектов могут служить отсутствие части строковых инструкций (инструкции с префиксом REP) для архитектуры x86 и неверное указание адресов инструкций из слотов задержки для архитектуры MIPS64.

Первичная разметка трассы выделяет в ней команды, относящиеся к различным процессам, нитям и зонам. Под зоной понимается период действия адресного пространства виртуальной памяти. Отображение адресов в современных процессорных архитектурах поддерживается аппаратно и не всегда соотносится с понятием процесса: например, ОС при копировании содержимого буфера памяти между различными процессами переключает адресные пространства.




*Рис. 3. Последовательное повышение уровня представления трассы.*

Следующие четыре алгоритма разбиваются на две пары, которые могут выполняться параллельно.

Восстановление точек возникновения программных и аппаратных прерываний, исключений с выявлением диапазона шагов трассы, в которых выполнялся код обработчика прерывания.

После выявления обработчиков прерываний в трассе происходит сопоставление машинных команд вызовов функций и соответствующих возвратов. Одним из результатов этого сопоставления является то, что для каждой нити восстанавливается стек вызовов для каждого момента времени.

Крайне важным алгоритмом является выявление диапазонов времени, когда не происходило никаких модификаций кода. Различение кода и данных в общем случае – не решаемая задача. В данном случае различение происходит на основе трассы: поддерживаются три множества адресов памяти, используемых для чтения данных, записи, и извлечения выполняемых команд. Выявление факта модификации исполняемого кода приводит к созданию

нового поколения, характеризуемого номером. Для каждого поколения определен период жизни в терминах номеров шагов трассы.

В периодах постоянства кода можно выполнять поиск загруженных в память и выполнявшихся модулей, что впоследствии позволяет заменять адреса памяти символьными метками, извлеченными из бинарных файлов соответствующих модулей.

Наиболее важный этап – построение статико-динамического представления. Для каждого поколения кода строится граф, описывающий поток управления. Среди ребер выделяются ребра вызова функций и возврата в них, ребра, передающие управление между различными поколениями.

На основе полученного статико-динамического представления происходит выявление в коде функций. Следует отметить, что на практике регулярно встречается весьма неожиданное устройство тел функций. Например, точка входа может быть расположена после команд возврата, несколько функций могут иметь общие фрагменты кода и т.п.

После применения перечисленных алгоритмов анализа, большая часть которых работает в полностью автономном режиме, пользователь имеет возможность изучать построенное высокоуровневое представление, извлекать из него слайсы на основе анализа зависимостей по данным и управлению, формально описывать восстановленные функции и т.п. Начиная с уровня потоков данных и управления, появляется возможность проводить анализ (дальнейшее повышение уровня представления) полностью изолировав все особенности целевой процессорной архитектуры. Таким образом, добавление поддержки новой архитектуры сводится к разработке декодера и модификации части этапов предварительной обработки.

## *3. Выделение кода алгоритма*

Базовой задачей при восстановлении алгоритма является выделение относящегося к нему кода. Так как любой алгоритм является последовательностью действий по преобразованию входных данных в выходные, для выделения применяются алгоритмы слайсинга (построения срезов) [9]. В общем виде на выходе выделения кода алгоритма должен быть получен подграф графа потока управления программы, в котором оставлены только те ребра, базовые блоки и команды в них, которые относятся к реализации алгоритма.

### 3.1. Выделение кода алгоритма по трассе

Для определения шагов трассы, относящихся к алгоритму, выполняется динамический слайсинг: либо прямой по входным данным, либо обратный по выходным данным, либо строится чоп как пересечение прямого и обратного слайсов. Входные и выходные данные должны быть заданы пользователем. Задача идентификации входов и выходов алгоритма частично



автоматизируется при помощи различных компонентов среды анализа, рассмотрение которых находится за рамками данной статьи.

Проведение слайсинга по трассе требует последовательного рассмотрения зависимостей в машинных командах и поддержания множества помеченных регистров и ячеек памяти. Один из способов организации такого слайсинга был предложен в [10].

Основная проблема динамического анализа – потенциальная неполнота покрытия программы – оказывает свое серьезное влияние на слайсинг по трассе. В связи с этим важно было решить задачи объединения результатов слайсинга по нескольким трассам и подготовки входных данных для снятия трасс, гарантированно улучшающих покрытие кода алгоритма. Первая задача решается на базе восстанавливаемого из трассы межпроцедурного графа потока управления программы: по результатам выделения алгоритма в каждой трассе помечаются отдельные ребра, базовые блоки и команды в них. В итоге помеченные хотя бы один раз элементы графа считаются относящимися к алгоритму.

В некоторых случаях, кроме того, требуется проведение статического слайсинга как обеспечивающего консервативную полноту выделения кода алгоритма. Кроме того, разработанный метод анализа позволяет получать дополнительные базовые блоки и ребра не только из трасс, но и в результате взаимодействия с дизассемблером IDA Pro или симулятором QEMU. В этих случаях подход к описанию машинных команд в виде списков зависимостей, предложенный в [10], не может быть применен: списки зависимостей строятся не для команды вообще, а для команды в данной точке трассы. Тем самым рассматриваются конкретные адреса ячеек памяти и для команд с нетривиальным внутренним потоком управления конкретный вид данного потока. В то же время в статическом слайсинге необходимо рассматривать всю совокупность возможных зависимостей, обусловленных выполнением команды.

Для того чтобы задачи проведения статического слайсинга и подбора входных данных для улучшения покрытия могли быть решены в общем виде для всех целевых процессорных архитектур, в рамках среды анализа было разработано промежуточное представление, удобное для описания поведения машинных команд. Представление обеспечивает возможность отслеживания потоков данных и выполнения компиляторных преобразований (таких как продвижение и свертка констант), что необходимо для корректного рассмотрения ячеек памяти при статическом слайсинге и не может быть достигнуто без знания операционной семантики операций.

### 3.2. Промежуточное представление Pivot

Промежуточное представление Pivot [11] обеспечивает возможность единообразного описания операционной семантики машинных команд, что

позволяет абстрагироваться от особенностей набора команд целевой машины в различных видах анализа потоков данных и семантики программы.

В представлении Pivot каждая машинная команда представлена в виде последовательности простых операторов, которые взаимодействуют с тремя видами переменных:

- временными переменными, находящимися в форме с единичным статическим присваиванием (SSA) и локальными в рамках набора операторов одной машинной команды;
- элементами адресных пространств, единообразно описывающими регистры и диапазоны памяти в виде троек ($S$, $a$, $s$), где $S$ – адресное пространство, $a$ – адрес в нем, а $s$ – размер переменной;
- битами слова состояния Pivot, такими как флаги переноса, переполнения, нуля, знака и т.п.

Основными операторами являются:

- оператор инициализации временной переменной константным значением;
- оператор применения операции;
- оператор ветвления, в т.ч. условного в зависимости от битов слова состояния;
- операторы загрузки и выгрузки, выполняющие пересылки между временными переменными и элементами адресных пространств.

Каждая машина может иметь свой набор операций, однако на практике многие операции оказываются общими для всех машин (арифметико-логические операции, операции с плавающей точкой и т.д.). Операция принимает в качестве параметров набор временных переменных и формирует результат также во временной переменной. Кроме того, операция может читать и записывать некоторые биты слова состояния. Для каждой операции набор таких битов фиксирован и не зависит от конкретных значений параметров. В сочетании с минимальным набором операторов и SSA-формой для временных переменных это проектировочное решение позволило соблюсти баланс между простотой анализа потоков управления и данных поверх представления Pivot с одной стороны и его относительной компактности (особенно для архитектур с большим количеством побочных эффектов, таких как x86) с другой стороны. Одновременное достижение этих свойств является отличительной особенностью представления Pivot по сравнению с решающими близкие задачи промежуточными представлениями в других средах анализа бинарного кода, в частности Vine [6] и BAP [12].

Описания операций, адресных пространств и операционной семантики команд каждой машины в компактном виде представляются в виде бинарных файлов, называемых архивами машин. Эти файлы подгружаются средой анализа при работе с трассами соответствующих машин.



### 3.3. Применение промежуточного представления Pivot

Наличие единого промежуточного представления позволило решить в рамках среды следующие задачи.

1. Выполнение инструментируемой конкретной интерпретации машинных инструкций, при помощи чего может быть реализован, в том числе, слайсинг по трассе. При этом одновременно вычисляются конкретные адреса, необходимые для корректной обработки косвенной адресации, и поддерживаются множества помеченных элементов в каждом адресном пространстве.
2. Проведение статического слайсинга на межпроцедурном графе потока управления для более полного выделения алгоритма. Свойства промежуточного представления позволяют применить известные алгоритмы статического слайсинга без существенных изменений.
3. Проведение символьных вычислений, что позволяет для не реализованного в трассе перехода подбирать входные данные, обеспечивающие его выполнение. При этом строится система уравнений, состоящая из условий вдоль соответствующего пути в программе: рассматриваются отдельные операторы, составляющие инструкции, влияющие на поток управления, и переводятся в уравнения. Полученная система подается на вход SMT-решателю Z3, а результат работы последнего отображается обратно на модель адресных пространств в начальной точке. На основе этих данных может быть снята новая трасса, улучшающая покрытие кода программы.

## *4. Восстановление форматов данных*

Внутри программы данные хранятся и передаются в виде переменных, имеющих некоторые типы в системе типов языка программирования, на котором написана программа. В процессе компиляции операции с переменными этих типов преобразуются в конструкции на языке ассемблера соответствующей процессорной архитектуры. Задача восстановления типов, то есть получение высокоуровневых типов данных отдельных переменных по ассемблерной программе, является частью задачи декомпиляции, равно как и идентификация самих переменных. Подходы к решению этих задач зависят как от процессорной архитектуры, так и от особенностей компилятора. С другой стороны, в ходе обмена данными с другими системами программа может воспользоваться либо одним из видов внешней памяти (например, жестким диском), либо каналом связи, таким как сеть Ethernet. При этом данные принимают форму файлов и сетевых пакетов, формат которых должен поддерживаться другими системами и, таким образом, в общем случае не связан с особенностями конкретной архитектуры и компилятора. Например, поддержка одних и тех же протоколов стека TCP/IP реализуется в самых разных ОС и архитектурах, а формат исполняемых файлов PE может

применяться в процессорных архитектурах x86, MIPS, ARM, PowerPC и многих других.

В работах [13, 14] приведено описание того, что обычно вкладывается в понятие «формат данных»:

- границы отдельных полей базовых типов в сообщении (в случае работы с файлом весь файл рассматривается как сообщение);
- группировка полей базовых типов в поля-последовательности и поля-записи с учетом вложенности;
- семантика отдельных полей, в частности поля длины; поля-разделители, определяющие длины последовательностей; поля-указатели, хранящие смещения других полей; ключевые поля, которые могут содержать предопределенный набор констант, специфичных для данного формата; поля-флаги, состоящие из групп отдельных битов;
- дополнительная информация для полей специальных типов: возможные значения ключевых полей (константы протокола); связь между полями-указателями и полями, на которые они указывают; связь между полями-последовательностями и полями, задающими их длину.

Особенность большинства современных вычислительных систем заключается в том, что программа может обрабатывать только данные, которые лежат в оперативной памяти и на регистрах. Поэтому в процессе обработки любому объекту, такому как файл или сетевое сообщение, соответствует буфер в оперативной памяти, в котором в некоторый момент времени находятся данные этого объекта. Следовательно, задачу восстановления формата данных можно свести к восстановлению формата буфера памяти, содержащего этот объект. Вообще говоря, может не существовать единого буфера, содержащего весь объект в один момент времени. Сложности, возникающие в таких ситуациях, будут описаны ниже. Далее рассматривается частный случай, когда все сетевое сообщение или файл считываются за одну операцию чтения и, соответственно, все данные исследуемого объекта попадают в один буфер в памяти. После того, как буфер определен, схема анализа может быть представлена в виде последовательности следующих шагов.

- Выделение алгоритма обработки данных, содержащихся в буфере, с привязкой конкретных частей буфера к машинным командам, в которых они обрабатываются.
- Восстановление плоского формата сообщения: определение границ отдельных полей на основе анализа отдельных команд доступа к данным буфера.



- Восстановление иерархического формата сообщения путем группировки отдельных полей в записи и последовательности на основе структурного анализа графа потока управления выделенного алгоритма.
- Восстановление семантической информации полей.

При выделении алгоритма отличают случаи разбора полученных извне программы данных и генерации данных для отправки. В первом случае для выделения алгоритма применяется классический анализ распространения помеченных данных [15]. Во втором случае может быть использован, например, алгоритм, описанный в [16].

Восстановление полей производится в соответствии с алгоритмом анализа диапазонов, использующихся отдельными инструкциями при доступе к буферу [17]. Для разрешения возникающих конфликтов применяется жадный алгоритм.

При группировке полей в виде записей и последовательностей используется эвристическое предположение о том, что поля, доступ к которым осуществляется в цикле, монотонно и непрерывно, от меньших смещений к большим, образуют последовательность, причем на каждой итерации цикла обрабатываются поля, образующие одну запись. Для осуществления этого анализа предварительно производится выделение циклов (в том числе развернутых) в CFG и поиск границ их выполнения в трассе.

Идея *выведения семантики* основана на алгоритме выведения типов в задаче декомпиляции. Вводится понятие *источник семантики*, то есть место в программе, в котором семантика используемых данных точно известна, после чего восстанавливаются зависимости по данным между этими источниками и данными в буфере. Список источников семантики приведен в работе [18]. Его можно дополнить специфическими конструкциями в CFG программы, например условными переходами, прерывающими выполнение цикла по условию, по которым можно определять поля длины и разделители.

Дополнительными трудностями, возникающими при восстановлении форматов являются:

- неполное восстановление формата из-за ограничений динамического анализа;
- шифрование/дешифрование данных;
- работа программы с несколькими источниками/потребителями и форматами данных;
- поступление данных по частям (считывание файла, сообщение из двух пакетов).

Для преодоления неполноты восстановления используется объединение форматов, полученных при анализе нескольких трасс, с помощью модифицированного алгоритма Нидлмана-Вунша [19], изначально применявшегося в биоинформатике. В ходе этого алгоритма выполняется обобщение каждого формата, то есть получение частичной грамматики, и

выравнивание грамматик, то есть сопоставление элементов двух грамматик и их слияние.

Для обработки шифрованного трафика используются *модели функций* шифрования и дешифрования. При выделении отдельных источников и потребителей данных кроме моделей функций ввода-вывода используется понятие *идентификатор источника*. Для обработки данных, поступающих по частям, вводится понятие *виртуальный буфер* и анализ параметров функций ввода-вывода. Более подробно подход к обработке шифрованного трафика описан в [14].

Одной из целей восстановления форматов данных, помимо описания алгоритма, является автоматическая генерация разборщиков сетевых сообщений для систем DPI, IDS и IPS. Примером системы разбора сетевых сообщений общего назначения с открытым исходным кодом является Wireshark. Разборщики в данной системе пишутся на языке Си, что делает затруднительной их автоматическую генерацию. Поэтому для реализации прототипа системы автоматического создания разборщиков по восстановленным форматам данных была выбрана система IDS Bro, для которой доступен компилятор binpac, создающий библиотеку разбора по описанию формата в виде грамматики. Пример с восстановлением формата и генерацией разборщика для него приводится в разделе 6.

Дальнейшим развитием рассмотренного подхода в случае анализа сетевых сообщений является автоматическое получение автомата протокола по коду реализации, что позволит решить общую задачу восстановления протоколов. На данный момент известно несколько работ в этом направлении [12, 20, 21].

## *5. Архитектура среды анализа*

При проектировании архитектуры принимаемые решения базировались на требованиях, обусловленных исследовательским характером среды и ее достаточно быстрым развитием, большим объемом анализируемых трасс и вспомогательных данных, а также практикой использования среды. Среди таких требований можно выделить три основных.

1. Среда должна быть построена на модульной архитектуре: добавление поддержки новой процессорной архитектуры, формата хранения, алгоритма анализа или компонента отображения информации не должно требовать изменения инфраструктуры.
2. Среда должна в полной мере задействовать возможности современной аппаратуры: использование нескольких вычислительных ядер должно быть поддержано как на уровне архитектуры в целом, так и на уровне отдельных алгоритмов анализа.
3. Среда должна проводить анализ в фоновом режиме, позволяя пользователю продолжать интерактивную работу с графическим интерфейсом. Интерфейс должен сохранять высокий уровень



отзывчивости вне зависимости от вычислительной нагрузки, обусловленной выполняемым анализом.

### 5.1. Архитектура потоков выполнения

При работе среда использует несколько потоков выполнения, распределяемых между вычислительными ядрами (рис. 4). Два потока с повышенным приоритетом существуют постоянно и выполняют два цикла обработки сообщений: первый поток обслуживает графический интерфейс, а второй – неграфические объекты среды, доставляя сообщения от одних к другим. Оба эти потока не нагружаются вычислительными задачами, за счет чего обеспечивается должный уровень отзывчивости.

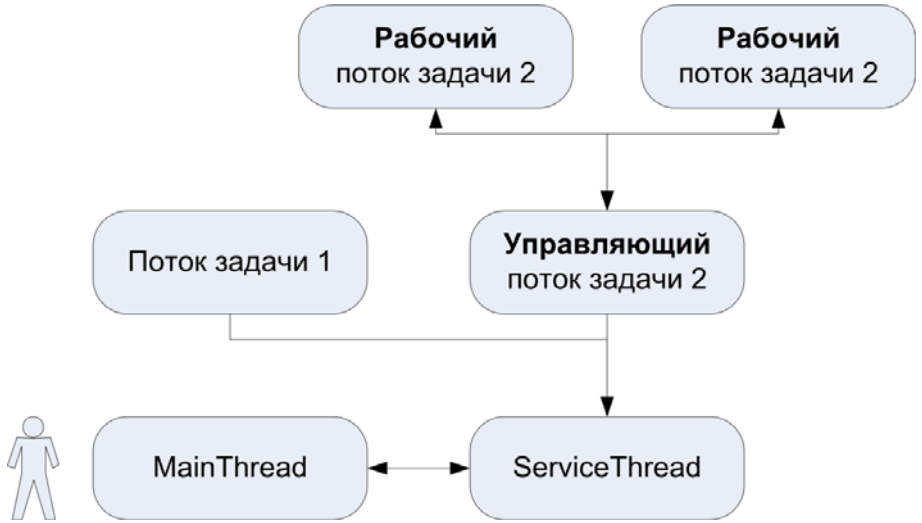


*Рис. 4 – Потоки выполнения.*

Алгоритмы анализа работают в дополнительных потоках, создаваемых по требованию. При запуске вычислительной задачи для нее создается управляющий поток. В зависимости от природы задачи вычисления могут проводиться либо в этом единственном потоке, либо могут быть созданы дополнительные рабочие потоки. В последнем случае управляющий поток организует взаимодействие рабочих потоков и формирование общего результата.

### 5.2. Рабочее пространство

Рабочее пространство организуется в виде дерева, пример которого приведен на рис. 5, со следующими вершинами:

- вершина рабочего пространства: корень дерева, отвечающий исследуемой программной системе в целом;
- вершина гнезда трасс, соответствующая набору трасс, снятых с одного и того же начального состояния машины, и хранящая результаты статического анализа этого набора;
- вершина трассы, хранящая результаты ее динамического анализа.

При этом вершины трасс могут образовывать поддеревья произвольной высоты за счет возможности работы с подтрассами – подпоследовательностями шагов в родительской трассе или подтрассе.

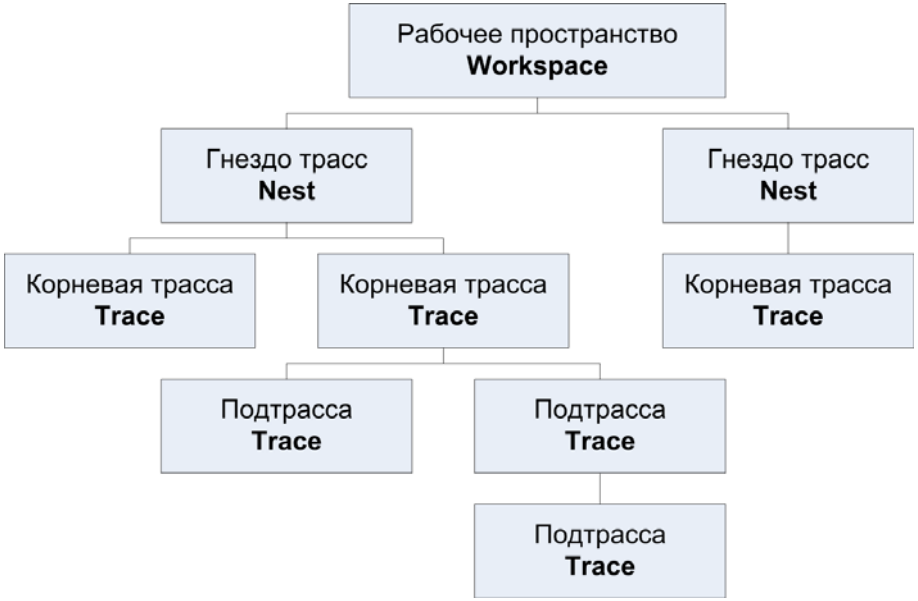


*Рис. 5. Дерево рабочего пространства.*

Вершины в дереве рабочего пространства называются *контекстами* и предоставляют возможность единообразной работы с данными, соответствующими различным уровням анализа. В рамках каждого контекста поддерживается база данных типа ключ-значение, файловое хранилище и реестр ресурсов, более подробно описываемый в следующем подразделе.

Гибкая структура дерева рабочего пространства позволяет совместно применять как динамические, так и статические методы анализа, проводить анализ по нескольким трассам, а также агрегировать в рамках одного рабочего пространства несколько физически обособленных компонентов исследуемой системы. Так, возможно создать, например, рабочее пространство, где одному



гнезду будет соответствовать клиент, а другому – сервер клиент-серверного приложения. При этом среда анализа позволяет гнездам одного рабочего пространства относиться к разным процессорным архитектурам: например, клиентское приложение может работать на мобильной платформе с процессором ARM, а серверная часть – на машине x86.

## 5.3. Управление ресурсами и задачами

Для управления набором построенных алгоритмами анализа данных используются *реестры ресурсов* в каждом из контекстов дерева рабочего пространства. *Ресурсом* называется объект, соответствующий некоторому виду данных и обладающий определенным интерфейсом, например, база данных процессов, потоков выполнения и зон в гнезде или разметка вызовов в трассе.

Объекты-ресурсы снабжены счетчиками ссылок, что позволяет отслеживать количество их пользователей и автоматически управлять их освобождением. Кроме того, ко всем ресурсам предъявляется требование потоковой безопасности, и каждый из них снабжается синхронизационным примитивом типа read-write lock, который захватывается всякий раз при вызове публичного метода ресурса (рис. 6).

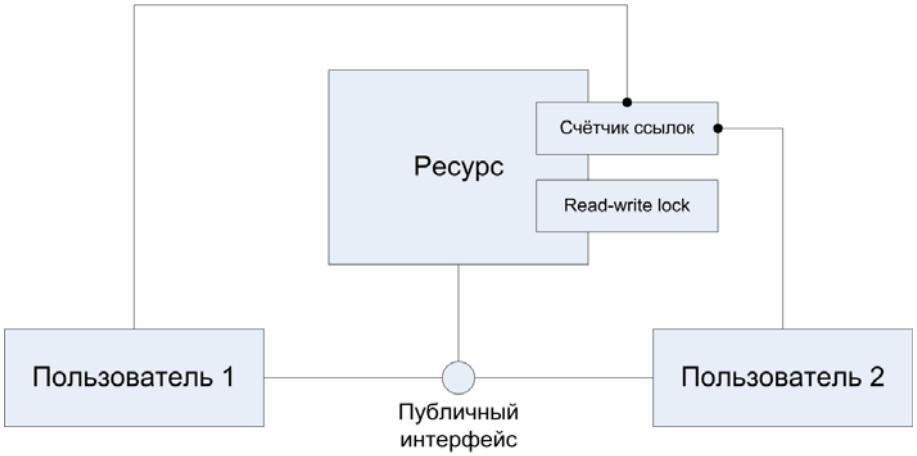


*Рис. 6 – Ресурс и его пользователи.*

Ресурсы могут в произвольный момент быть *отозваны*. Это означает, что данные, предоставляемые этим ресурсом, более не являются актуальными. При отзыве ресурса необходимо обеспечить решение двух задач. Во-первых, все текущие пользователи ресурса должны быть уведомлены о том, что данный ресурс больше не пригоден к использованию. Во-вторых, необходимо

освободить объект ресурса (здесь важно напомнить о том, что среда работает в большом количестве потоков выполнения, не во все из которых возможно доставлять сообщения асинхронно).

Для решения этих задач реализован следующий механизм. При осуществлении каждого доступа к ресурсу, как было указано выше, захватывается синхронизационный примитив. Непосредственно перед таким захватом среда автоматически проверяет признак, отозван ли данный ресурс. Если да, то выбрасывается исключение, обрабатываемое вызывающим: уменьшается счетчик ссылок ресурса, а вызывающий либо заканчивает текущую обработку с ошибкой, либо действует иным уместным способом. Когда все ссылки на отозванный ресурс будут удалены, среда автоматически освободит его.

Реестры ресурсов, привязанные ко всем контекстам в дереве рабочего пространства, устанавливают соответствие между строковыми идентификаторами интерфейсов (*точками монтирования*) и ресурсами, реализующими эти интерфейсы (рис. 7). Один ресурс может реализовывать несколько интерфейс, и один интерфейс может быть реализован в нескольких ресурсах.

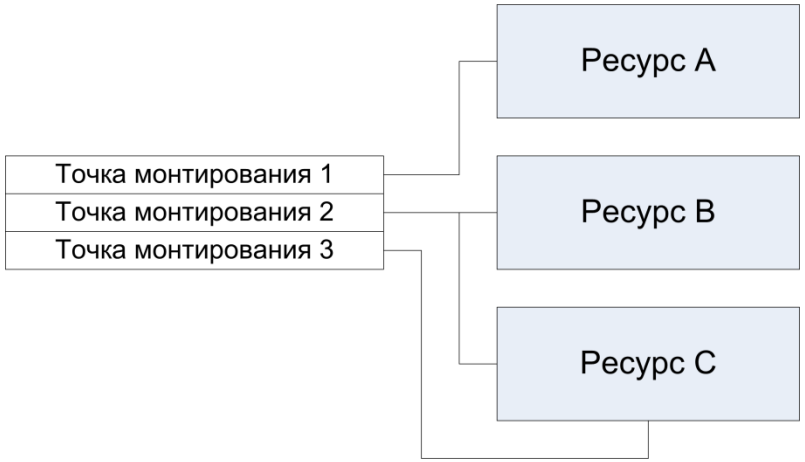


*Рис. 7 – Реестр ресурсов.*

Построение данных для ресурсов, как и все остальные виды длительных обработок, в среде реализуются в виде *задач*. Задача представляет собой обособленный алгоритм, работающий в отдельном потоке выполнения с возможностью порождения дополнительных рабочих потоков. Задача при запуске связывается с контекстом дерева рабочего пространства, в котором она работает. Контекст поддерживает список работающих в нем задач и не



может быть закрыт или удален до тех пор, пока все они не завершат работу. Задачи обладают механизмом для печати информационных сообщений и передачи информации о текущем прогрессе и состоянии, что отображается в графическом интерфейсе среды. По желанию пользователя задача может быть приостановлена или отменена.

Задача получает на вход набор параметров и после окончания работы формирует результат. Все параметры и результаты представляются единообразно в виде объектов *параметров задачи*. Кроме того, в процессе работы задача может асинхронно выдавать промежуточные результаты. Например, алгоритм поиска в трассе всех обращений к ячейке памяти может выдавать шаги по мере их нахождения, что позволяет пользователю начать анализ результатов, не дожидаясь полного завершения задачи.

### 5.4. Модули расширения

Модули расширения среды анализа представляют собой динамически подгружаемые библиотеки и разбиты на следующие два класса.

1. Инфраструктурные модули расширения (ИМР) предоставляют реализации базовых интерфейсов среды. ИМР обеспечивают поддержку процессорных архитектур, форматов исполнимых файлов, форматов трасс, интеграцию с трассирующими симуляторами и т.п.
2. Функциональные модули расширения (ФМР) обогащают среду поддержкой новых видов анализа, а также связанными с этим компонентами хранения и отображения данных. ФМР реагируют на основные события среды, в первую очередь события загрузки или выгрузки одного из контекстов дерева рабочего пространства. В ответ на такое событие ФМР добавляет или удаляет предоставляемые ресурсы в реестре этого контекста. Для хранения данных используются база данных типа ключ-значение и файловое хранилище контекста. Такая схема позволяет реализовывать произвольные виды анализа и обеспечивает возможность взаимодействия между модулями расширения без необходимости внесения изменений в ядро среды.

## *6. Примеры применения среды*

### 6.1. Пример выделения и построения модели алгоритма

Данный пример показывает работу методов выделения алгоритмов. Был выбран алгоритм генерации лицензии в программе управления лицензиями. Данная программа получала на вход файл, содержащий идентификатор машины, а на выходе генерировала файл, содержащий лицензию в бинарном виде. Для начала анализа была снята с процесса генерации выходного файла. В полученной трассе были восстановлены модули и подключена символьная информация. Далее был найден вызов функции WriteFile, соответствующий выводу лицензии в файл. Были восстановлены значения параметров вызова и,

соответственно, данные лицензии. По характерным адресам был определена точка вызова функции main. Для извлечения алгоритма генерации ключа был применен обратный слайсинг с учетом адресных зависимостей для буфера лицензии, от точки вызова функции WriteFile до точки вызова функции main. Полученный слайс был далее отфильтрован для удаления кода ядра ОС. Результат был представлен в виде дерева вызовов – последовательности вызовов функций, реализующих алгоритм, с учетом их вложенности. Результат представлен на рис. 8.

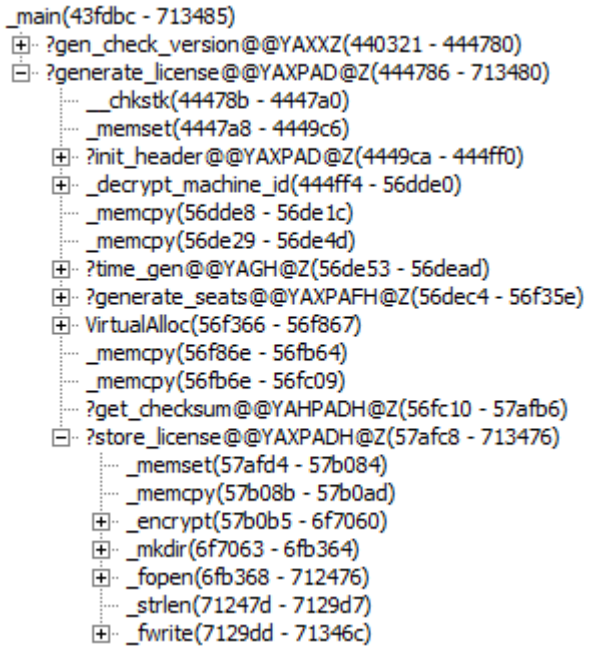


*Рис. 8 – Последовательность вызовов функций, реализующих алгоритм, с учетом их вложенности.*

Как видно из рисунка, алгоритм состоит из двух основных частей – функций gen_check_version для извлечения из реестра версии программы и проверки ее соответствия версии кода и generate_license, отвечающей за непосредственную генерацию лицензии. Для описания этих частей были проанализированы все перечисленные функции, определены и описаны их параметры. В частности, функция get_checksum получает на вход два параметра – адрес (addr) и размер (size) буфера данных, по которому считается контрольная сумма. Они передаются через стек и могут быть описаны, как Mem(ESP + 4, 4) и Mem(ESP +8, 4), где Mem(*адрес*, *размер*) – набор ячеек памяти, начиная с адреса *адрес*



размером *размер*, а ESP – значение указателя стека в момент начала исполнения функции. В ячейке Mem(ESP, 4) лежит адрес возврата. В свою очередь, сам буфер данных (buf) может быть описан как Mem(addr, size). Выходной параметр checksum является целым числом и в соответствии с используемым соглашением о связях cdecl находится в регистре EAX.

По результатам анализа были построены графы зависимостей на уровне вызовов функций для каждой из двух частей. Граф для основного алгоритма генерации лицензии приведен на рис. 9.

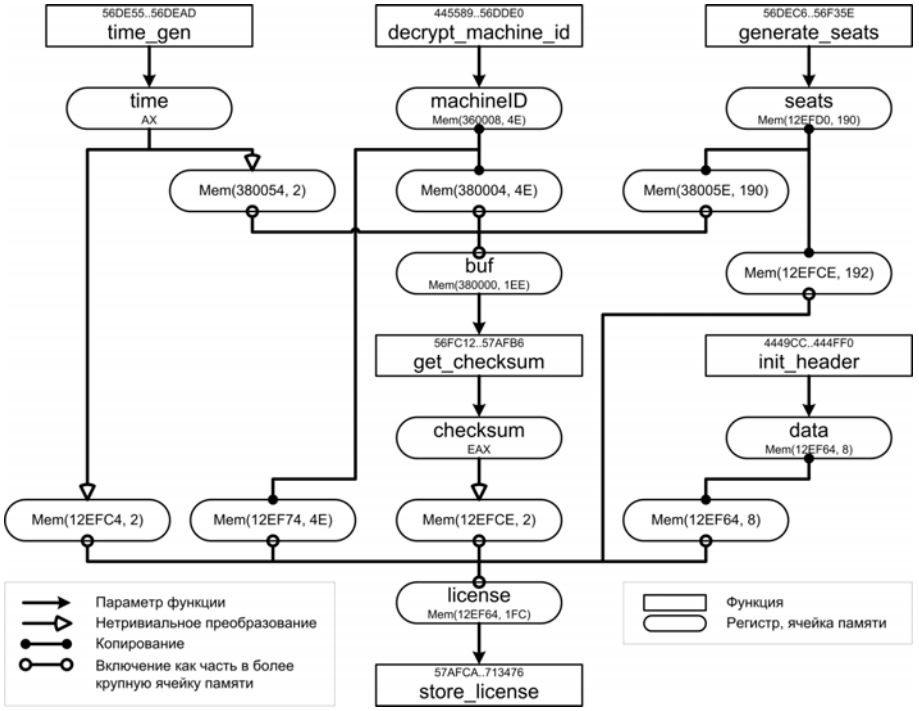


*Рис. 9 – Схема алгоритма генерации лицензии.*

Как видно из двух рисунков, в лицензию в зашифрованном виде (функция encrypt) входят следующие данные, вычисляемые соответствующими функциями:

- идентификатор машины (функция decrypt_machine_id, берущая данные из входного файла);
- время создания лицензии (функция time_gen);
- количество рабочих мест (функция generate_seats);

- контрольная сумма по перечисленным выше данным (функция get_checksum);
- заголовок лицензии (функция init_header).

## 6.2. Пример восстановления формата данных

В качестве примера для демонстрации восстановления форматов данных выбран протокол DNS, описание которого содержится в документах RFC 1350 и RFC 3425. Описание формата приведено кратко на рис. 10 (а). Выбор обусловлен тем, что этот формат имеет достаточно сложную структуру и большое количество полей с различной семантикой. Анализ протокола DNS был выполнен по программе nslookup, входящей в стандартную поставку Windows 2000. Трасса снималась в процессе отправки прямого DNS-запроса до получения ответа от DNS-сервера, его разбора и вывода на экран в текстовом виде. Размер буфера сообщения составил 46 байт, время анализа – 3 с. Фрагмент восстановленного формата приведен на рис. 10 (б). По восстановленному формату была получена частичная грамматика, листинг которой приведен на рис. 11 для трансляции формата в библиотеку разборки пакетов с помощью компилятора binpac. На вход полученной программе-разборщику был подан набор сообщений-ответов на прямые DNS-запросы, отличные от исходного, которые были полностью и без ошибок разобраны. Для дополнения грамматики путем анализа нескольких трасс, помимо прямого DNS запроса были проанализированы также обратный и SOA DNS запросы, что позволило расширить применимость разборщика и на эти варианты сообщений.


*Рис. 10 – (а) Формат DNS сообщения согласно RFC 1350, (б) восстановленный формат ответа на прямой DNS-запрос.*



```
0:buffer;
{
      1:field:uint16:simple:empty;
      2:field:uint8:flag(1+1+1+4+1):empty;
      3:field:uint8:flag(4+2+1+1):empty;
      4:field:uint16:length:empty;
      5:field:uint16:length:empty;
      6:field:uint16:simple:empty;
      7:field:uint16:simple:empty;
      8:sequence:?:10:4;
      9:sequence:?:11:5;
}
```

*Рис. 11 – Описание формата для генерации разборщика компилятором binpac.*

## *7. Обзор близких работ*

По открытым публикациям хорошо известны работы, в которых предлагались расширяемые среды анализа бинарного кода. Большинство публикаций с результатами высокого уровня принадлежат исследовательским коллективам ведущих университетов США.

В 2005 году была опубликована работа [22] (Университет Висконсина), в которой представлена платформа CodeSurfer/x86, поддерживающая статический анализ бинарного кода. Для более точного восстановления потока управления авторами был предложен алгоритм VSA [23], строящий верхнее приближение множества значений для регистров и ячеек памяти во всех точках программы. Работа с восстановленным графом потока управления производилась инструментами системы CodeSurfer, рассчитанной на статический анализ Си-программ.

В проекте BitBlaze [6] (Университет Беркли) были получены наиболее близкие результаты по сравнению с теми, что представлены в данной статье. В рамках данного проекта была разработана инфраструктура (промежуточное представление Vine [6], эмулятор TEMU [8]), облегчающая анализ бинарного кода, а также значительное количество программных инструментов, этой инфраструктурой пользующихся. Основная направленность работ – поддержка восстановления алгоритмов из бинарного кода, в том числе вредоносного. Авторы BitBlaze предлагают получать по трассам выполнения статическое представление [24] и повышать его уровень до блок-схем [25].

Блок-схемы представляют собой двудольные графы, совмещающие в себе потоки данных, управления и отношение производитель-потребитель. Помимо того, коллективом BitBlaze были разработаны инструменты поиска ошибок в бинарном коде [26, 27] на основе инфраструктурных компонент, реализующих символьную интерпретацию.

Ответвлением проекта BitBlaze стал проект BAP, Binary Analysis Platform [12], ведущийся в Университете Карнеги-Меллон. Базовым компонентом инфраструктуры является промежуточное представление BIL, и средства трансляции в него. Основным направлением работ является автоматический поиск ошибок на основе символьной интерпретации [28].

С развитием рынка мобильных устройств возник интерес к анализу бинарного кода прошивок и прикладных программ смартфонов. Система DroidScope [7] переназначена для восстановления алгоритмов реализованных на платформе Android. Модифицированная версия эмулятора QEMU используется для сбора трасс различного уровня: машинные команды (ARM или x86), события ОС, команды байт-кода Dalvik. Совмещение трасс и автоматический анализ помеченных данных позволяют в итоге восстанавливать алгоритм в виде двудольного графа.

## *8. Заключение*

В статье представлена среда анализа, позволяющая проводить исследования бинарного кода, опираясь только на знание о работе машинных команд целевой процессорной архитектуры. Это позволяет анализировать код широкого класса программ – прикладных, код драйверов и ядра ОС, код BIOS. Модульная архитектура среды поддерживает в настоящий момент такие архитектуры, как x86, x86-64 и ARM, и, начиная с уровня потоков данных и управления, может быть расширена на любую другую архитектуру без изменения самих алгоритмов анализа.

В составе среды реализованы методы восстановления алгоритмов и форматов данных, которые этими алгоритмами обрабатываются. Разработано промежуточное представление Pivot, используемое для статического слайсинга и организации символьных вычислений.

Архитектура среды, в рамках которой реализованы алгоритмы анализа, позволяют эффективно задействовать ресурсы современных многоядерных рабочих станций.

Дальнейшие работы в данной области предполагают развитие методов автоматического выявления ошибок и их последующей классификации.

**Список литературы**


[1]. Андрей Тихонов, Арутюн Аветисян, Вартан Падарян. Методика извлечения алгоритма из бинарного кода на основе динамического анализа. // Проблемы информационной безопасности. Компьютерные системы. №3, 2008. Стр. 66-71





[2]. А.И. Аветисян, В.А. Падарян, А.И. Гетьман, М.А. Соловьев. О некоторых методах повышения уровня представления при анализе защищенного бинарного кода. // Материалы XIX Общероссийской научно-технической конференции «Методы и технические средства обеспечения безопасности информации», 2010. Стр. 97-98.

[3]. А.Ю.Тихонов, А.И. Аветисян. Комбинированный (статический и динамический) анализ бинарного кода. // Труды Института системного программирования РАН, том 22, 2012 г. стр. 131-152. DOI: 10.15514/ISPRAS-2012-22-9.

[4]. Alexander Getman, Vartan Padaryan, and Mikhail Solovyev. Combined approach to solving problems in binary code analysis. // Proceedings of 9th International Conference on Computer Science and Information Technologies (CSIT'2013), pp. 295-297.

[5]. К. Батузов, П. Довгалюк, В. Кошелев, В. Падарян. Два способа организации механизма полносистемного детерминированного воспроизведения в симуляторе QEMU. // Труды Института системного программирования РАН, том 22, 2012 г. Стр. 77-94. DOI: 10.15514/ISPRAS-2012-22-6.

[6]. Dawn Song, David Brumley, HengYin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Prateek Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. // International Conference on Information Systems Security, 2008, LNCS 5352, pp. 1-25.

[7]. Lok Kwong Yan and Heng Yin. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. // In Proceedings of the 21st USENIX conference on Security symposium (Security'12). USENIX Association, Berkeley, CA, USA, pp. 29-29.

[8]. Heng Yin and Dawn Song. TEMU: Binary Code Analysis via Whole-System Layered Annotative Execution. / EECS Department University of California, Berkeley, Technical Report No. UCB/EECS-2010-3, January 11, 2010, p. 14.

[9]. M. Harman, S. Danicic, Y. Sivagurunathan, D. Simpson. The next 700 slicing criteria. // Second UK Workshop on Program Comprehension, 1996.

[10]. Падарян В. А., Гетьман А. И., Соловьев М. А. Программная среда для динамического анализа бинарного кода. // Труды Института системного программирования РАН, том 16, 2009 г. Стр. 51-72.

[11]. Падарян В. А., Соловьев М. А., Кононов А. И. Моделирование операционной семантики машинных инструкций. // Программирование, № 3, 2011 г. Стр. 50-64.

[12]. David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. BAP: a binary analysis platform. // In Proceedings of the 23rd international conference on Computer aided verification (CAV'11), Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 463-469.

[13]. А. И. Гетьман, Ю. В. Маркин, В. А. Падарян, Е. И. Щетинин. Восстановление формата данных. // Труды Института системного программирования РАН, том 19, 2010 г. Стр. 195-214.

[14]. А. И. Аветисян, А. И. Гетьман. Восстановление структуры бинарных данных по трассам программ. Труды Института системного программирования РАН, том 22, 2012 г. Стр. 95-118. DOI: 10.15514/ISPRAS-2012-22-7.

[15]. J. Newsome, D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. // In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2005.

[16]. J. Caballero, P. Poosankam, C. Kreibich, D. Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. // In Proceedings of the 16th ACM conference on Computer and communications security (CCS), 2009, pp. 621-634.

[17]. W. Cui, M. Peinado, K. Chen, H. J. Wang, L. Irun-Briz. Tupni: automatic reverse engineering of input formats. // In CCS'08: Proceedings of the15th ACM conference on Computer and communications security, 2008.
[18]. Z. Lin, X. Zhang, and D. Xu. Automatic reverse engineering of data structures from binary execution. // In Proceedings of the Network and Distributed System Security Symposium, 2010.
[19]. S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. // Journal of Molecular Biology, 48(3):443–453, 1970.
[20]. Y. Wang, Z. Zhang , D. Yao, B. Qu, L. Guo. Inferring protocol state machine from network traces: a probabilistic approach. // In Proceeding of the 9th international conference on Applied cryptography and network security (ACNS), 2011, pp. 1-18.
[21]. P.M. Comparetti, G. Wondracek, C. Kruegel, E. Kirda. Prospex: Protocol Specification Extraction. // In Proceedings of the 30th IEEE Symposium on Security and Privacy, 2009, pp. 110-125.
[22]. Gogul Balakrishnan, Radu Gruian, Thomas Reps, and Tim Teitelbaum. CodeSurfer/x86—A platform for analyzing x86 executables. // In Proceedings of the 14th international conference on Compiler Construction (CC'05), Springer-Verlag, Berlin, Heidelberg, pp. 250-254.
[23]. Gogul Balakrishnan and Thomas Reps. Analyzing Memory Accesses in x86 Executables. // In Proceedings of Compiler Construction, Springer-Verlag, New York, 2004, pp. 5-23.
[24]. Domagoj Babić, Lorenzo Martignoni, Stephen McCamant, and Dawn Song. Statically-directed dynamic automated test generation. // In Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11). ACM, New York, USA, pp. 12-22.
[25]. Dan Caselden, Alex Bazhanyuk, Mathias Payer, Stephen McCamant and Dawn Song. HI-CFG: Construction by Binary Analysis, and Application to Attack Polymorphism. // In Proceedings of 18th European Symposium on Research in Computer Security, Egham, UK, 2013. LNCS 8134, pp. 164-181.
[26]. Prateek Saxena, Pongsin Poosankam, Stephen McCamant, and Dawn Song. Loop-extended symbolic execution on binary programs. // In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, USA, pp. 225-236.
[27]. Juan Caballero, Pongsin Poosankam, Stephen McCamant, Domagoj Babić, and Dawn Song. Input generation via decomposition and re-stitching: finding bugs in Malware. In Proceedings of the 17th ACM conference on Computer and communications security (CCS '10). ACM, New York, USA, pp. 413-425.
[28]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. Unleashing Mayhem on Binary Code. // In Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12). IEEE Computer Society, Washington, USA, pp. 380-394.



# Methods and software tools for combined binary code analysis ⋆

*V. A. Padaryan, A. I. Getman, M. A. Solovyev, M.G. Bakulin, A.I. Borzilov, V.V. Kaushan, I.N. Ledovskich, U.V. Markin, S.S. Panasenko*
*{ vartan, thorin, eyescream, bakulinm, helendile, korpse, il, ustas, spanasenko}@ispras.ru*

**Abstract.** This paper presents methods and tools for binary code analysis that have been developed in ISP RAS and their applications in fields of algorithm and data format recovery. The analysis subject is executable code of various general purpose CPU architectures. The analysis is carried out in lack of source code, debug records, and without specific OS version requirements. The approach consists of collecting a detailed machine instruction level execution trace; method for successive presentation level increase; extraction of code belonging to the algorithm followed by structuring of both code and data formats it processes. Important results have been achieved: an intermediate representation has been developed, that allows for carrying out most of the preliminary processing tasks and algorithm code extraction without having to focus on specifics of a given machine; and a method and software tool have been developed for automated recovery of network message and file formats. The tools have been incorporated into a unified analysis platform that supports their combined use. The architecture behind the platform is also described in the paper. Examples of its application to real programs are given.

**Keywords:** binary code, static analysis, dynamic analysis, integrated software platform.

## References

[1]. Tikhonov A.YU., Avetisyan A.I., Padaryan V.A., Metodika izvlecheniya algoritma iz binarnogo koda na osnove dinamicheskogo analiza [Methodology of exploring of an algorithm from binary code by dynamic analysis]. Problemy informatsionnoj bezopasnosti. Komp'yuternye sistemy. 2008, №3. pp. 66-71 (in Russian)
[2]. Avetisyan A.I., Padaryan V.A., Getman A.I., Solov'ev M.A. O nekotorykh metodakh povysheniya urovnya predstavleniya pri analize zashhishhennogo binarnogo koda [Some Approaches To Raising Representation Level In Analysis Of Protected Binary Code]. Materialy Obshherossijskoj nauchno-tekhnicheskoj konferentsii «Metody i tekhnicheskie sredstva obespecheniya bezopasnosti informatsii», 2010. pp. 97-98. (in Russian)
[3]. Tikhonov A.YU., Avetisyan A.I. Kombinirovannyj (staticheskij i dinamicheskij) analiz binarnogo koda. [Combined (static and dynamic) analysis of binary code]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 22, 2012, pp. 131-152. DOI: 10.15514/ISPRAS-2012-22-9. (in Russian)

⋆ The paper is supported by RFBR grant 11-07-00450-a

[4]. Getman A.I., Padaryan V.A., Solov'ev M.A. Combined approach to solving problems in binary code analysis. Proceedings of 9th International Conference on Computer Science and Information Technologies (CSIT'2013), pp. 295-297.
[5]. Batuzov K.A., Dovgalyuk P., Koshelev V.K., Padaryan V.A. Dva sposoba organizatsii mekhanizma polnosistemnogo determinirovannogo vosproizvedeniya v simulyatore QEMU [Two Approaches To Full-System Deterministic Replay QEMU]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 22, 2012, pp. 77-94. DOI: 10.15514/ISPRAS-2012-22-6. (in Russian)
[6]. Song D., Brumley D., Yin H., Caballero J., Jager I., Kang M.G., Liang Z., Newsome J., Poosankam P., Saxena P. BitBlaze: A New Approach to Computer Security via Binary Analysis. International Conference on Information Systems Security, 2008, LNCS 5352, pp. 1-25.
[7]. Yan L.K., Yin H. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. Proceedings of the 21st USENIX conference on Security symposium (Security'12). USENIX Association, Berkeley, CA, USA, pp. 29-29.
[8]. Yin H., Song D. TEMU: Binary Code Analysis via Whole-System Layered Annotative Execution. EECS Department University of California, Berkeley, Technical Report No. UCB/EECS-2010-3, January 11, 2010, p. 14.
[9]. Harman M., Danicic S., Sivagurunathan Y., Simpson D.. The next 700 slicing criteria. Second UK Workshop on Program Comprehension, 1996.
[10]. Padaryan V.A., Getman A.I., Solov'ev M.A. Programmnaya sreda dlya dinamicheskogo analiza binarnogo koda [Software environment for dynamic analysis of binary code]. Trudy ISP RAN [The Proceedings of ISP RAS], vol 16, 2009, pp. 51-72 (in Russian).
[11]. Padaryan V.A., Solov'ev M.A., Kononov A.I.. Simulation of operational semantics of machine instructions. Programming and Computer Software, May 2011, Volume 37, Issue 3, pp 161-170, DOI 10.1134/S0361768811030030
[12]. Brumley D., Jager I., Avgerinos T., Schwartz E. J. BAP: a binary analysis platform. Proceedings of the 23rd international conference on Computer aided verification (CAV'11), pp. 463-469.
[13]. Getman A.I., Markin YU.V., Padaryan V.A., Shhetinin E.I. Vosstanovlenie formata dannykh [Data format recovery]. Trudy ISP RAN [The Proceedings of ISP RAS], 2010, vol. 19, pp. 195-214 (in Russian)
[14]. Avetisyan A.I., Getman A.I. Vosstanovlenie struktury binarnykh dannykh po trassam program [Recovery the structure of binary data structures from program traces]. Trudy ISP RAN [The Proceedings of ISP RAS], 2012, vol. 22, pp. 95-118. DOI: 10.15514/ISPRAS-2012-22-7. (in Russian)
[15]. Newsome J., Song D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. Proceedings of the Network and Distributed System Security Symposium (NDSS), 2005.
[16]. Caballero J., Poosankam P., Kreibich C., Song D. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. Proceedings of the 16th ACM conference on Computer and communications security (CCS), 2009, pp. 621-634.
[17]. Cui W., Peinado M., Chen K., Wang H. J., Irun-Briz L. Tupni: automatic reverse engineering of input formats. Proceedings of the15th ACM conference on Computer and communications security, 2008.
[18]. Lin Z., Zhang X., Xu D. Automatic reverse engineering of data structures from binary execution. Proceedings of the Network and Distributed System Security Symposium, 2010.



[19]. Needleman S. B., Wunsch C. D. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology, 48(3):443–453, 1970.
[20]. Wang Y., Zhang Z., Yao D., Qu B., Guo L. Inferring protocol state machine from network traces: a probabilistic approach. Proceeding of the 9th international conference on Applied cryptography and network security (ACNS), 2011, pp. 1-18.
[21]. Comparetti P.M., Wondracek G., Kruegel C., Kirda E. Prospex: Protocol Specification Extraction. Proceedings of the 30th IEEE Symposium on Security and Privacy, 2009, pp. 110-125.
[22]. Balakrishnan G., Gruian R., Reps T., Teitelbaum T. CodeSurfer/x86—A platform for analyzing x86 executables. Proceedings of the 14th international conference on Compiler Construction (CC'05), Springer-Verlag, Berlin, Heidelberg, pp. 250-254.
[23]. Balakrishnan G., Reps T. Analyzing Memory Accesses in x86 Executables. Proceedings of Compiler Construction, Springer-Verlag, New York, 2004, pp. 5-23.
[24]. Babić D., Martignoni L., McCamant S., Song D. Statically-directed dynamic automated test generation. Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11). ACM, New York, USA, pp. 12-22.
[25]. Caselden D., Bazhanyuk A., Payer M., McCamant S., Song D. HI-CFG: Construction by Binary Analysis, and Application to Attack Polymorphism. Proceedings of 18th European Symposium on Research in Computer Security, Egham, UK, 2013. LNCS 8134, pp. 164-181.
[26]. Saxena P., Poosankam P., McCamant S., Song D. Loop-extended symbolic execution on binary programs. Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, USA, pp. 225-236.
[27]. Caballero J., Poosankam P., McCamant S., Babić D., Song D. Input generation via decomposition and re-stitching: finding bugs in Malware. Proceedings of the 17th ACM conference on Computer and communications security (CCS '10). ACM, New York, USA, pp. 413-425.
[28]. Cha S. K., Avgerinos T., Rebert A., Brumley D. Unleashing Mayhem on Binary Code. Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12). IEEE Computer Society, Washington, USA, pp. 380-394.