На правах рукописи

Гетьман Александр Игоревич

ВОССТАНОВЛЕНИЕ ФОРМАТОВ СЕТЕВЫХ СООБЩЕНИЙ И ФАЙЛОВ ПО БИНАРНЫМ ТРАССАМ ПРОГРАММ

Специальность 05.13.11 –
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва
2013

Работа выполнена в Федеральном государственном бюджетном учреждении науки Институте системного программирования Российской академии наук.

Научный руководитель: Аветисян Арутюн Ишханович доктор физико-математических наук, доцент

Официальные оппоненты: Крюков Виктор Алексеевич, доктор физико-математических наук, профессор, заведующий отделом Федерального государственного бюджетного учреждения науки Институт прикладной математики им. В. Келдыша Российской академии наук

Хорошилов Алексей Владимирович, кандидат физико-математических наук, старший научный сотрудник Федерального государственного бюджетного учреждения науки Института системного программирования Российской академии наук

Ведущая организация: Федеральное государственное бюджетное учреждение науки Вычислительный центр им. А.А. Дородницына Российской академии наук

Защита диссертации состоится 13 июня 2013 г. в 15 часов на заседании диссертационного совета Д 002.087.01 при Федеральном государственном бюджетном учреждении науки Институте системного программирования Российской академии наук по адресу: 109004, Москва, ул. Александра Солженицына, д. 25.

С диссертацией можно ознакомиться в библиотеке Федерального государственного бюджетного учреждения науки Института системного программирования Российской академии наук.

Автореферат разослан "___" _____ 2013 г.

Учёный секретарь
диссертационного совета,
кандидат физ.-мат. наук /Прохоров С.П./

Общая характеристика работы

Актуальность. В настоящее время ведутся активные исследования и разработки в области анализа бинарного кода, как в ведущих научных центрах, так и в коммерческих организациях. В результатах таких исследований заинтересованы многие организации, решающие задачи сертификации ПО, обратной инженерии, обеспечения информационной безопасности, рефакторинга, синтеза программ по их спецификациям и др.

Анализ бинарного кода, в первую очередь, связан с выделением и анализом реализованных в программах алгоритмов, с целью их понимания и формального описания. Полноценное восстановление алгоритмов невозможно без восстановления форматов и структур данных, передаваемых по сети в виде сетевых сообщений или считываемых с устройств внешней памяти, где они хранятся в виде файлов.

Восстановленные форматы данных находят применение не только при решении задач формального описания алгоритмов. Эта информация также требуется при обеспечении сетевой безопасности и эффективного управлении сетевыми потоками. Для снижения количества уязвимостей, которые могут эксплуатироваться злоумышленниками, используются системы автоматизированного тестирования программ. Одним из распространённых методов тестирования является фаззинг, который используется в системах Packet Vaccine и ShieldGen. Наличие описания структуры входных данных позволяет значительно повысить эффективность и снизить время тестирования. Распространённым методом предотвращения эксплуатации обнаруженных, но ещё не устранённых разработчиком ПО уязвимостей, являются программы фильтрации входного трафика, такие как Shield.

Для предотвращения несанкционированного доступа по сети используются системы обнаружения и предотвращения вторжений (IPS и IDS), такие как Snort и Bro. Во всех этих системах выполняется разбор сетевых пакетов, с использованием технологии DPI для оценки потенциальной опасности их содержимого. Для применения этой технологии также необходимо описание структуры сетевых пакетов.

Рост объёмов и видов сетевого трафика приводит к необходимости его классификации с целью эффективного управления потоками сетевых пакетов. Для успешной классификации требуется уметь распознавать трафик, то есть определять тип протоколов, используемых в сетевых пакетах и привязывать его к конкретным приложениям. Ранее, для решения этой задачи было достаточно информации о предопределённых номерах портов, используемых конкретными приложениями. В настоящее время, большую часть сетевого трафика составляют пакеты, относящиеся к протоколам, которые не используют фиксированные номера портов. Для классификации такого трафика также требуется знание его формата.

Восстановление формата "вручную" является весьма сложной и ресурсоёмкой задачей, требующей высокой квалификации, а получаемые результаты могут быть не вполне точны. Таким образом, проблема автоматизации восстановления форматов сетевых сообщений и файлов является актуальной.

Целью диссертационной работы является исследование и разработка методов и соответствующих инструментальных средств, поддерживающих автоматизированное восстановление форматов сетевых сообщений и файлов по бинарным трассам программ. Разрабатываемые методы должны обеспечивать возможность дополнения и уточнения получаемых результатов за счёт анализа нескольких трасс исполнения.

Основные результаты.

1. Разработан метод автоматизированного восстановления форматов сетевых сообщений и файлов на основе информации, извлекаемой из бинарных трасс программ. Метод базируется на совместном использовании оригинального алгоритма анализа помеченных данных и структурного анализа графа потока управления.

2. Разработан метод объединения форматов данных, получаемых при анализе нескольких трасс, обеспечивающий повышение точности получаемых результатов, за счёт увеличения покрытия кода.

3. На основе предложенных автором методов, разработана и реализована подсистема восстановления форматов сетевых сообщений и файлов.

Практическая ценность работы состоит в том, что разработанная подсистема восстановления форматов данных среды анализа бинарного кода, разрабатываемой в ИСП РАН существенно расширяет класс решаемых задач. Эффективность подсистемы подтверждена на примере анализа сетевых сообщений и файлов с открытым форматом. Подсистема используется в различных научно-исследовательских и промышленных организациях.

Апробация работы. Основные результаты работы опубликованы в статьях [1-7] и обсуждались на следующих конференциях:

1. Международные конференции "РусКрипто" (Москва, 2009 и 2010 гг.)
2. Научно-технические конференции "Методы и технические средства обеспечения безопасности информации" (Санкт-Петербург, 2010 и 2011 гг.)

Структура и объём работы. Диссертация состоит из введения, пяти разделов, заключения и одного приложения. Работа изложена на 127 страницах. Список источников насчитывает 75 наименований. Диссертация содержит 4 таблицы и 33 рисунка.

Краткое содержание работы

Во введении обсуждается цель диссертационной работы, обосновывается ее актуальность. Формулируются основные результаты работы. Рассматриваются возможности практического применения предложенных в работе методов и реализованных инструментов, даётся краткий обзор работы.

В **главе 1** описывается постановка задачи и виды информации, включаемые в понятие *формат данных*, а также форма её представления. В разделе 1.1 описываются контекст решаемой задачи, даются понятия *сообщение*, *сессия*, *автомат протокола*, необходимые при постановке задачи.

Большинство сетевых протоколов уровня приложения используют концепцию *сессии*, то есть последовательности сообщений, которые передаются между участниками обмена в рамках некоторого *протокола*. Сетевые сообщения представляют собой байтовую последовательность фиксированного размера. Под *протоколом* понимается набор правил двух видов, специфицирующих какие данные и как могут передаваться по данному протоколу. Правила первого вида описывает допустимые последовательности сообщений, передаваемых в одной сессии. Второй вид правил описывает структуру всех типов сообщений, то есть какие данные передаются в сообщениях этого типа, и как именно они упаковываются в последовательность байт. Процесс считывания файла также можно рассматривать как процесс обмена данными между двумя участниками. При этом одним из участников является файловая система - файл рассматривается как одно сообщение, а понятие сессия используется для обозначения последовательности операций с этим файлом. Далее для обозначения объекта, формат которого восстанавливается, будет использоваться общий термин *сообщение*, обозначающий сетевое сообщение или файл.

В разделе 1.2 описывается понятие формат данных. Вводятся базовые элементы формата данных - *поля*, их основные характеристики - смещение от начала сообщения и размер. С точки зрения канала передачи данных сообщение представляет собой последовательность байт. Однако с точки зрения участников обмена, сообщения состоят из последовательности единиц данных, обрабатываемых программой. То есть, с позиции кода обработки, сообщения аналогичны классу *структура* языка Си, поэтому для обозначения обрабатываемых единиц данных, используется термин *поле*.

В процессе обработки, каждому полю сопоставляется переменная программы, в которую помещается значение поля. Поэтому, поле, как и переменная, имеет тип. По аналогии с работами, посвящёнными восстановлению типов данных по бинарному коду, можно использовать понятие простого типа данных, как совокупность трёх характеристик:

- базовый тип {целый, с плавающей точкой, указатель}
- знаковость {знаковый, беззнаковый}
- размер в битах {1..128}

Размер поля измеряется в битах, так как в некоторых архитектурах существуют команды работы с отдельными битами, такие как BT и BST в архитектуре x86-64. Границы размера (от 1 до 128 бит) связаны с существующими ограничениями процессорных архитектур: максимальный размер операндов равен 16 байтам и используется в командах SSE и MMX архитектуры x86-64. Задачи восстановления таких характеристик как базовый тип и знаковость решаются в работах по декомпиляции и в настоящей работе не рассматриваются. Кроме типа,

важной характеристикой поля является его размещение в сообщении, которое задаётся в виде смещения относительно начала сообщения.

В разделе также водится понятие *семантики* поля, которая описана в протоколе, и используется программой при получении и интерпретации значений этих полей. Для того чтобы поле могло быть корректно обработано, программа должна уметь получать и интерпретировать его значение. Для получения значения необходимо знать кодировку поля, то есть способ, по которому из значений байтов и битов, входящих в поле, можно получить значение поля. Также необходим способ его упаковки в виде последовательности байтов. Порядок следования байтов в многобайтовых типах данных является одним из примеров кодировки. С точки зрения использования, поля можно разделить на те, которые обеспечивают возможность передачи и хранения данных и те, что содержат непосредственно сами данные. В процессе передачи и хранения возникает ряд задач:

- обеспечение возможности передачи и хранения данных переменного размера
- обеспечение целостности сообщения
- обеспечение непоследовательного доступа к данным
- возможность задавать тип сообщения, или команду, которая в нём содержится

Для решения этих задач используются поля, обладающие специальной семантикой.

Для хранения и передачи данных переменного размера используются поля длины, содержащие размер этих данных, и поля разделители, содержащие специальный терминальный символ, обозначающий конец данных переменной длины. Примером полей второго вида являются нулевой символ на конце нуль-терминированной строки. Для обеспечения целостности используются поля с контрольными суммами, а для непоследовательного доступа к данным - поля указатели, содержащие смещение некоторого другого поля от начала сообщения. Тип сообщения и команды, которые оно передаёт, задаются с помощью специальных ключевых полей, которые могут содержать только значения из определённого набора констант, определяемых протоколом. Например, файл BMP начинается с ключевого слова "BM", задающего тип файла.

Значения полей, в процессе обработки, используются программой в качестве параметров различных функций, что накладывает на поля дополнительные ограничения. Например, вызов функций работы с файловой системой подразумевает, что поле содержит имя файла. А использования параметра в функции открытия сетевого соединения подразумевает, что поле содержит IP-адрес или номер порта.

Представление сообщения в виде "плоской" последовательности полей позволяет описать многие простые сетевые протоколы, такие как TFTP. Однако, во многих форматах, таких как файлы с потоковым видео, данные хранятся в виде последовательностей записей, каждая из которых включает в себя некоторый

набор полей. Таким образом, данные помимо "плоской" могут иметь иерархическую структуру, на основе вложенности полей в *записи*, а *записей* в *последовательности*.

Формат сообщения описывает представление этого сообщения в виде иерархической структуры, базовым элементом которой являются поля, которые сгруппированы в записи и последовательности по принципу вложенности. Для поля известен его размер и смещение относительно начала сообщения, а также набор атрибутов, определяющих его семантику. На рис. 1 показан пример формата GET-запрос протокола HTTP на загрузку файла index.html, который обычно используется для отображения страниц сайтов в браузере.

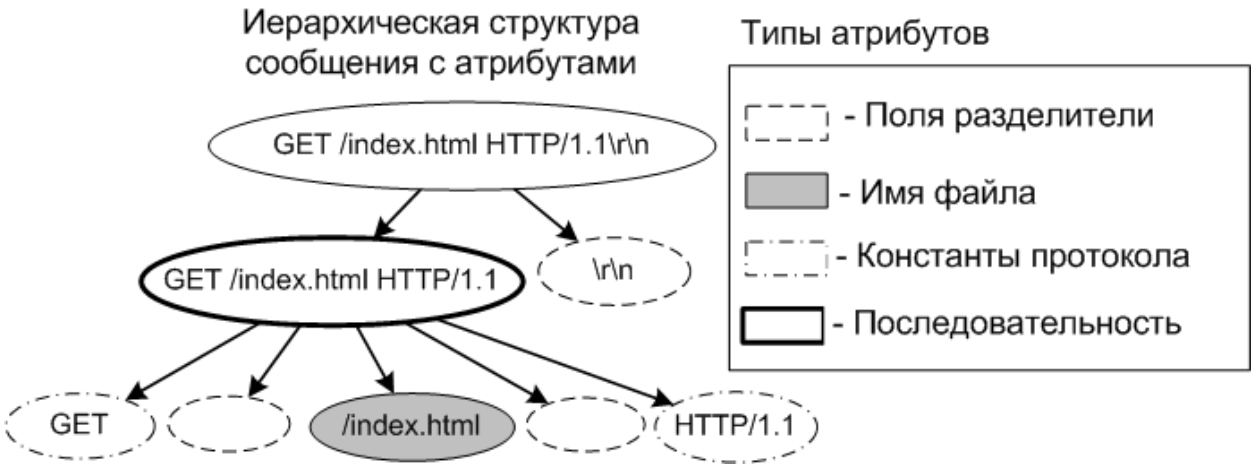

Рисунок 1. Пример описания иерархической структуры и атрибутов отдельных полей для GET-запроса протокола HTTP

В конце раздела осуществляется постановка промежуточной и основной задач:

- восстановление иерархического формата конкретного сообщения, соответствующего дереву синтаксического разбора по трассе обработки этого сообщения для того чтобы обеспечить
- восстановление возможно более полной грамматики, описывающей язык, словами которого являются все сообщения этого типа на основе обобщения и объединения деревьев разбора набора сообщений одного типа.

В **главе 2** описываются существующие подходы к решению задачи восстановления форматов и программные средства, которые их реализуют. При решении задачи, в качестве входных данных, доступен сетевой трафик, а также исполняемый файл клиентской стороны обмена. Код серверной стороны, как правило, не доступен. Исполняемый файл может анализироваться статически, либо анализу подвергается процесс его выполнения на некоторых входных данных. В соответствии с описанными входными данными выделяют три основных подхода к решению задачи восстановления форматов - анализ сетевого трафика, статический и динамический подходы.

Подход на основе анализа сетевого трафика описывается в разделе 2.1. Этот подход основан на статистических методах выделения повторяющихся последовательностей байт. Границы полей определяются в этом подходе с достаточно низкой точностью, за исключением ключевых полей, содержащих константы протокола. Иерархическая структура не восстанавливается. Восстановление семантической информации ограничивается полями с постоянными значениями - ключевыми полями и полями-разделителями. Этот подход используется в инструменте Discoverer. В более поздних инструментах, реализующих этот подход, основной акцент делается на восстановлении автомата протокола, а анализ формата отдельных сообщений выполняется в степени, необходимой для разделения на группы, соответствующие типам сообщений. Обычно в качестве признаков принадлежности к одной группе, используются общие подстроки или значения найденных ключевых слов. К таким системам анализа относятся PEXT и ReverX.

В разделе 2.2 рассматривается статический подход. Он позволяет определять возможные последовательности вызовов функций ввода/вывода и тем самым, косвенно определять иерархическую структуру сообщения. Основным ограничением является отсутствие точных значений переменных и параметров при статическом анализе и проблема алиасинга. Это приводит к тому, что размеры вводимых и выводимых данных могут быть не известны, что затрудняет определение границ отдельных полей. Этот подход реализован в программе FFE/x86.

В динамическом подходе, который описан в разделе 2.3, отслеживается процесс обработки буфера, содержащего сообщение, и на основе анализа доступа к данным делаются предположения о границах полей и их группировке. Основным ограничением динамического подхода является то, что для анализа доступен только код программы, выполнявшийся на конкретных входных данных. Это может приводить к тому, что восстанавливаемый формат будет не полон. Этот подход применяется во многих средствах анализа таких как Polyglot, AutoFormat и Tupni.

Раздел 2.4 описывает сходство и различие подходов к анализу входящих сетевых сообщений (считываемых файлов) и исходящих сетевых сообщений (записываемых файлов). Рассматриваются аспекты формата данных, которые могут быть восстановлены в каждом подходе. Основой анализа входящих сообщений является алгоритм распространения помеченных данных. Подобный алгоритм для анализа исходящих сообщений - алгоритм деконструкции буфера, был предложен в инструменте Dispatcher.

В разделе 2.5 рассматривается более общая, чем восстановление форматов данных задача - восстановление протокола, которая также включает задачу восстановление автомата протокола. Описываются возникающие при решении этой задачи проблемы, связанные с тем, что она является частным случаем задачи получения минимального языка по ограниченному набору его слов. В теореме Голда доказано, что эта задача неразрешима для регулярного языка. Также

показано, что в случае наличия также и набора слов, заведомо не относящихся к языку, задача его восстановления является NP-полной. Тем не менее, существует ряд эвристических подходов к решению этой задачи, которые можно разделить на три основные группы: статистические подходы, реализованные в инструментах PEXT и ReverX, подход на основе вывода грамматики (инструмент flowinfer) и подход на основе динамического анализа, реализованный в системе Prospex.

Раздел 2.6 посвящён рассмотрению специализированных средств представления результатов восстановления форматов данных и протоколов. Необходимость в таком представлении связана с тем, что разнообразие получаемой информации не укладывается в традиционные типы представления, такие как, например, расширенная БНФ форма грамматик. Кроме того, основной целью анализа формата сообщений является получение некоторого описания, на основе которого может быть легко получена программа-разборщик или генератор сообщений этого формата. Это приводит к тому, что разработанное представление должно предоставлять возможность автоматической генерации программы разборщика. Среди средств представления результатов восстановления формата рассматриваются специализированная система типов PacketTypes, система описания протоколов GAPA и генератор программ разборщиков для систем обнаружения и предотвращения вторжений binpac.

В разделе 2.7 на основе сделанного разбора делаются выводы и обосновывается решение о выборе для реализации динамического подхода и форме представления результатов на языке генератора binpac. Выбор динамического подхода обусловлен тем, что область его применения включает и защищённый код и протоколы с шифрацией трафика. Чтобы компенсировать ограничения этого подхода, вследствие неполного покрытия кода, используется механизм объединения форматов, полученных при анализе нескольких сообщений.

В **главе 3** осуществляется формальная постановка задачи восстановления формата в рамках динамического подхода. Описываются разработанные методы, и приводится детальное описание основных алгоритмов восстановления структурной и семантической информации.

Для начала анализа требуется получить специальное представление программы – трассу её работы на определённом сценарии. Требуется, чтобы в процессе работы программа считывала файл, формат которого исследуется, или получала соответствующие сетевые сообщения. Для получения трассы программа запускается внутри симулятора, так, что при выполнении каждой инструкции на процессоре, в трассу попадает код этой инструкции и состояние регистров процессора перед её выполнением.

Особенность современных вычислительных систем заключается в том, что программа может обрабатывать только данные, которые лежат в оперативной памяти и на регистрах. Поэтому, с точки зрения анализа трассы, любому высокоуровневому 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. Далее термин *буфер* используется в значении "хранилище данных изучаемого файла или сетевого сообщения".

Таким образом, для восстановления формата, требуется определить в какой точке трассы и в каком буфере памяти содержатся данные исследуемого объекта. В качестве такой точки можно использовать вызов библиотечной функции чтения файла или получения сетевого сообщения, после завершения которого, данные объекта содержатся в буфере, который задаётся параметрами вызова. После того, как буфер определён, схема анализа может быть представлена в виде последовательности следующих шагов.

1. Выделение алгоритма обработки данных, содержащихся в буфере.
2. Восстановление «плоского» формата сообщения - определение границ отдельных полей, на основании анализа размера операндов инструкций доступа.
3. Восстановление иерархического формата сообщения, путём группировки отдельных полей в записи и последовательности на основе структурного анализа графа потока управления выделенного алгоритма.
4. Восстановление семантической информации полей на основе поиска шаблонов определённого вида на графе потока управления, с использованием графа зависимостей по данным.

Общая схема восстановления формата и представление исследуемых данных на каждом этапе показаны на рисунке 2.

Рисунок 2. Схема работы алгоритмов восстановления формата и представление данных на каждом этапе

В разделе 3.1 даётся набор определений, используемых при описании алгоритмов анализа.

Раздел 3.2 содержит описание анализа помеченных данных, с помощью которого выполняется выделение алгоритма обработки данных буфера. Этот алгоритм реализует одну из разновидностей анализа потока данных. В ходе анализа каждый шаг трассы обработки, атрибутируется множеством ячеек входного буфера, значения которых используются на этом шаге. Дополнительно могут распространяться и другие атрибуты. Особенностью реализованного

подхода является использование битовой гранулярности, что позволяет повысить точность анализа и выделять поля-флаги, размер которых меньше одного байта. В качестве дополнительного атрибута используется признак постоянства значения ячейки исходного буфера в ходе распространения. Отслеживание этого признака требуется для успешного анализа семантики полей, так как при этом рассматриваются значения отслеживаемых ячеек и необходимо знать, соответствуют ли они значениям исходных ячеек. Процесс распространения «помеченных данных» показан на рисунке 3.

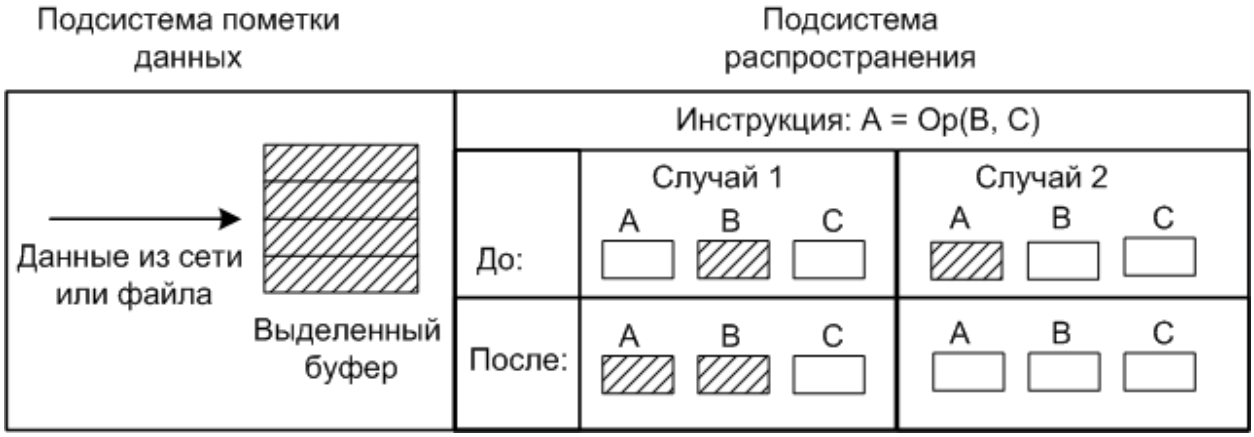

Рисунок 3. Схема работы алгоритма помеченных данных

Вначале в список помеченных данных заносятся все ячейки буфера, содержащего данные сетевого сообщения или файла. Далее осуществляется последовательный просмотр выполнявшихся инструкций, начиная с шага, на котором в буфер записываются соответствующие данные. Анализ останавливается, если достигнут конец трассы, или текущее множество помеченных ячеек пусто. При просмотре инструкции, выполняются следующие действия:

1. Если некоторые из операндов инструкции являются помеченными, то выходные данные инструкции, которые зависят от этих операндов, также заносятся в список помеченных.
2. Если выходной операнд есть в списке помеченных, а все входные операнды отсутствуют в этом списке, то выходные данные удаляются из списка помеченных.

В приведённом описании важно понятие зависимости между входными и выходными данными инструкции, которое влияет на точность и полноту получаемых результатов. Применение алгоритма к инструкциям процессора без учёта их семантики может приводить к проблеме лишних помеченных данных. Она выражается в том, что данные, не имеющие отношения к исходному буферу, могут оказаться помеченными. Эта ситуация может возникать вследствие следующих причин.

Лишние помеченные данные могут возникать из-за учёта лишних типов зависимостей, таких как косвенная (или адресная) зависимость. Такие

зависимости возникают между ячейкой, содержащей указатель и ячейкой, содержащей значение, адрес которой хранится в ячейке-указателе. В реализуемом подходе эти зависимости отслеживаются для поиска полей указателей, однако распространение помеченных данных по ним не выполняется. Это обусловлено тем, что предметом исследования является обработка значений, содержащихся в исходном буфере, а в случае косвенной зависимости, значение выходной ячейки не зависит от значения входной.

Другой причиной являются, так называемые, константные функции, которые могут быть реализованы с помощью различных процессорных инструкций. Они характеризуются тем, что значение выходного операнда функции не зависит от значений входных параметров и равно константе. Примерами таких инструкций могут служить `XOR AX, AX` и `SUB AX, AX`. Для того, чтобы избежать лишних помеченных данных в этом случае привлекается знание семантики для выделения класса инструкций, реализующих константные функции.

Ещё одной причиной неточности анализа может быть гранулярность помеченных данных и операндов. В случае, если только часть байт входного операнда является помеченной, и рассматриваемая инструкция является инструкцией копирования, то не все байты выходного операнда зависят от помеченных данных. То есть, требуется пометить не весь выходной операнд, а только те его байты, которые соответствуют помеченным байтам входного операнда. Чтобы повысить точность анализа выделяется класс инструкций, реализующих копирование данных.

Применение алгоритма помеченных данных на выходе даёт отображение шагов трассы в диапазоны битов исследуемого буфера, доступ к которым осуществляется в этих шагах. Шаги, присутствующие в этом отображении, представляют собой выделенный в виде подтрассы алгоритм обработки буфера, а диапазоны битов являются кандидатами для выделения отдельных полей в буфере сообщения. Сложность алгоритма можно оценить как $O(N \log M)$, где N — количество шагов в трассе, а M — размер статического представления программы (кода и данных). Это следует из того, что в худшем случае алгоритму потребуется обработать каждый шаг в трассе, при этом на каждом шаге обрабатываются все операнды инструкции количество которых O(1). При обработке шага осуществляется поиск операндов в множестве помеченных данных и его преобразование - $O(\log M)$.

В разделе 3.3 описывается алгоритм восстановления границ полей. Под восстановлением полей имеется в виду задача выделения из последовательности байт буфера таких непрерывных подпоследовательностей, каждая из которых интерпретируется обрабатывающим кодом как одно число.

Эта задача схожа с задачей выделения переменных, которая решается в декомпиляторах при восстановлении типов. Необходимость в решении этой задачи возникает из-за того, что в бинарном коде отсутствуют такие сущности, как *переменная*, *параметр*, *поле структуры*. В процессе компиляции эти данные отображаются на регистры и диапазоны ячеек памяти в стеке или куче. Поэтому,

прежде чем восстанавливать высокоуровневые типы данных, необходимо выделить участки памяти и регистры, на которые были отображены переменные и параметры, имевшие соответствующие типы в высокоуровневом коде.

Разница задачи восстановления полей с восстановлением переменных состоит в том, что переменные и параметры в большинстве языков высокого уровня явно описываются в разделе объявлений и компилятор использует эти знания при генерации кода. То есть, информация о типах явно отображается в набор используемых инструкций и вид их операндов. Поэтому знания об особенностях компилятора и целевой процессорной архитектуры могут использоваться при восстановлении переменных. В то же время, описание структуры сетевого сообщения или файла в исходном коде не содержится. Вследствие этих особенностей, для выделения отдельных полей используются размеры операндов инструкций доступа к данным, содержащимся в буфере.

На вход алгоритму подаётся отображение шагов трассы в диапазоны байт исходного буфера, которое было получено на выходе алгоритма помеченных данных. На каждом шаге рассматривается инструкция, выполнявшаяся на этом шаге и размеры её операндов, с помощью которых осуществлялся доступ к данным буфера. Например, пусть дана инструкция `ADD EAX, EBX`, причём в регистре `EAX` хранятся данные, взятые из исходного буфера по смещению 0x10. Так как размер операнда 4, то диапазон байт [0x10, 0x13] является кандидатом для объявления полем. После того, как все шаги алгоритма обработки рассмотрены, для всех обрабатываемых байт буфера известен диапазон, к которому он относится. Байты, не попавшие ни в один диапазон, в исследуемой трассе не обрабатываются. Поэтому, в рамках рассматриваемого подхода, о них ничего сказать нельзя. Такие части буфера, объявляются *виртуальными полями*.

Основной проблемой при таком подходе является неоднозначное отображение байт буфера в диапазоны. Это может происходить, если доступ к одним и тем же данным производится с различной гранулярностью. Такие ситуации обычно возникают при массовом доступе к памяти, например при копировании или вычислении контрольной суммы, когда побайтно считывается весь буфер, независимо от того, из каких полей он состоит. Эта ситуация также наблюдается при использовании оптимизированных функций обработки строк в формате ASCII и Unicode.

Для решения этой проблемы применяется эвристика, используемая в инструменте Tupni. Она основана на сопоставлении каждому выделенному полю веса, равного количеству инструкций, которые к нему обращались. При подсчёте веса игнорируются инструкции копирования, так как они не осуществляют обработку данных. В этом случае задача выделения полей сводится к определению покрытия буфера полями с максимальным весом (задача SCP). Эта задача является NP-полной, поэтому для поиска приближённого решения применяется жадный алгоритм, который показал хорошие результаты в процессе применения к реальным задачам. Результатом работы алгоритма является представление буфера в виде набора полей, каждое из которых хранит значение

одного из базовых типов данных. Сложность алгоритма можно оценить как $O((N+M^4)\log M)$, где N - количество шагов в трассе, а M — размер исследуемого сообщения (в байтах).

Для группировки полей в записи и последовательности в настоящей работе используется комбинация из структурного анализ графа потока управления (CFG) и анализа помеченных данных. Для его применения необходимо наличие максимально полного графа потока управления, информации о наличии циклов в этом графе и реализации (проекции) этих циклов в трассе. Таким образом, перед применением алгоритмов восстановления иерархической структуры необходимо восстановить граф потока управления, выделить в нём циклы и спроектировать их на трассу. Решение указанных задач рассматривается в разделе 3.4. Следствием восстановления графа потока управления по трассе является его неполнота - в граф попадают только реализовавшиеся рёбра. Для циклов, в которых отработала только одна итерация, обратное ребро в трассе нереализовалось, что препятствует выявлению этого цикла в CFG. Для преодоления этого ограничения был разработан алгоритм дополнения CFG на основе вычисления адресов нереализовавшихся переходов. Сложность данного алгоритма - $O(N)$, где N - количество базовых блоков в восстановленном CFG. Сложность алгоритма проектирования циклов на трассу - $O((N+M)\log M)$, где M - общее количество базовых блоков в CFG, N - количество шагов в трассе. Другой проблемой при выявлении циклов является применение оптимизирующих преобразований в современных компиляторах. Существует класс оптимизирующих преобразований, использование которых приводит к значительному искажению CFG программы, что снижает точность результатов восстановления. К таким преобразованиям, в первую очередь, относится оптимизация "развёртывание циклов". Пример подобной оптимизации на уровне исходного кода приведён на рисунке 4.

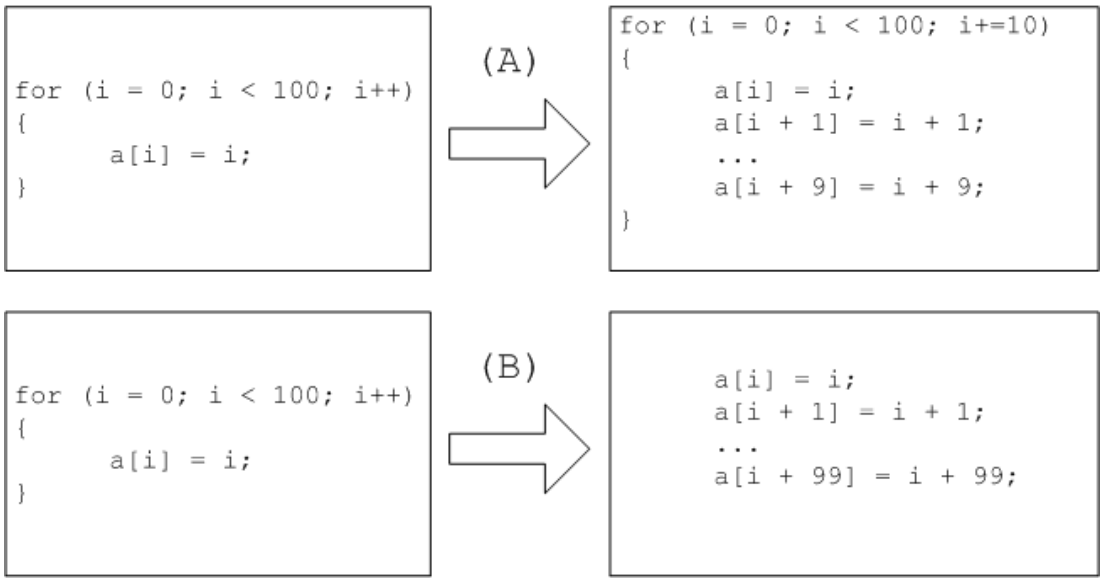

Рисунок 4. Пример частичного(A) и полного(B) развёртывания цикла.

Для уточнения результатов в условиях применения этой оптимизации был разработан алгоритм поиска развёрнутых циклов в графе потока управления. Этот алгоритм рассматривается в разделе 3.5. Сложность разработанного алгоритма - $O(NM^2)$, где N - количество базовых блоков в трассе, а M - максимальное количество инструкций в базовом блоке.

Описанию алгоритмов восстановления некоторых видов семантики отдельных полей посвящен раздел 3.6. В начале раздела описывается общий подход *вывода семантики* во многом схожий с алгоритмом вывода типов, который применяется в задачах декомпиляции. Вводится понятие *источник семантики*, то есть место в программе (в случае динамического анализа - в трассе), в котором семантика используемых данных точно известна. Такими источниками могут быть следующие.

- специальные инструкции, например инструкция запроса времени архитектуры x86 `RDTSC`, результат которой - число тактов, записывается в определённые регистры,
- специальные конструкции в CFG программы, например условные переходы, прерывающие выполнение цикла по условию. Если в цикле обрабатывается массив и значение каждого его элемента сравнивается с константой, то поле, содержащее константу, является полем разделителем, а сама константа - терминирующим символом,
- вызовы функций стандартной библиотеки, семантика параметров которых известна, например - функции для работы по сети оперируют IP-адресами, а функции работы с файловой системой - именами файлов.

Сложность данного алгоритма, с учётом наличия данных алгоритма распространения помеченных данных, и результатов вычисления ячеек с известной семантикой можно оценить как $O(N)$, где N - число источников семантики в трассе.

В настоящей работе предложены алгоритмы шаблонного поиска специальных конструкций в CFG программы с использованием графа зависимостей. Также используется подсистема описания функций, позволяющая описывать их параметры и, таким образом, задавать "источники семантики", соответствующие вызовам библиотечных функций.

После того, как список источников семантики известен, для выведения семантики нужно определить, значения каких именно полей используются в этих источниках. Эта задача решается в процессе анализа помеченных данных.

Как правило, большинство полей в сообщении содержат данные, специфичные для протокола и понять смысл значений этих данных можно только зная задачи, которые решает конкретный протокол. В то же время, существует ряд типов данных, которые используют многие протоколы, что позволяет разработать общие подходы к поиску полей, содержащие данные этих типов. К таким данным можно отнести, например, IP адреса, имена файлов, имена серверов в сети, номера портов, временные метки.

При передаче и обработке отдельного пакета возникает ряд задач, характерных для многих протоколов:

- передача данных переменного размера, например строк переменной длины,
- передача данных различных типов в одном контейнере,
- проверка отсутствия искажений, возникающих при передаче данных,
- ускорение обработки данных,
- уменьшение объёма передаваемых данных.

Для решения этих задач в самых различных протоколах используются общие подходы, решаемые с помощью специального вида полей. Для передачи данных переменного размера используют поля длины, задающие размер данных в конкретном пакете. В некоторых случаях, особенно при передаче строк используют поля разделители, содержащие специальные терминальные символы, означающие конец поля с данными переменного размера. Примером может служить нуль-терминированная строка. Для передачи данных различных типов в одном контейнере, используют ключевые поля, которые могут принимать ограниченный набор константных значений, однозначно задающий тип данных в контейнере. Для проверки отсутствия искажений в данных пакета, используются поля, содержащие контрольные суммы, значение которых зависит от значения других полей. Также зависимости могут возникать и в других случаях, при этом между значениями полей образуются функциональные зависимости, выявление которых важно для понимания схемы функционирования протокола. Одним из способов ускорения обработки данных является выравнивание отдельных частей пакета в памяти. Побочным эффектом является появление в пакете неиспользуемых байт, которые могут содержать произвольные значения, либо заполняются некоторой константой, например нулём. Для уменьшения объёма передаваемых данных используется несколько подходов. Можно выделить использование флагов - полей, размером в несколько бит, что позволяет в один байт поместить несколько переменных, в случае, если они могут принимать малое количество значений. В качестве другого метода сжатия используется замена повторяющихся значений в сообщении на специальные поля указатели, которые содержат смещение первого поля, содержащего это значение. Это имеет смысл, если значение занимает большее количество байт чем указатель. Такой метод сжатия используется в протоколе DNS.

То, что данные типы специальных полей используются во многих протоколах с одинаковыми целями, позволяет разработать алгоритмы поиска этих специальных полей, применимые к широкому классу протоколов.

В настоящей работе используются алгоритмы поиска следующих типов полей: поля длины, поля разделители, поля, содержащие флаги, поля, значения которых зависят от значений других полей, поля, содержащий константы протокола (ключевые поля), поля указатели.

Алгоритмы, позволяющие восстанавливать информацию о группировке полей на основе структурного анализа графа потока управления рассматриваются

в разделе 3.7. Используется эвристическое предположение, что поля, обрабатывающиеся в цикле, образуют *последовательность*, причём на каждой итерации цикла обрабатывается одна *запись*. Помимо циклов, *последовательности* могут обрабатываться рекурсивными вызовами, однако такой способ гораздо медленнее из-за накладных расходов и на практике не встречался. В основе предлагаемого структурного анализа лежит анализ циклов в графе потока управления и анализ реализаций этих циклов в трассе. Получение этой информации описано в разделе 3.4. Каждый выделенный в CFG цикл проектируется на трассу. Из полученных проекций выбираются только те, внутри которых есть обращение к данным буфера сообщения. Для этих циклов производится анализ отдельных итераций для того, чтобы выделить группы полей, к которым были обращения на каждой итерации. Для того, чтобы выделить циклы, соответствующие обработке *последовательностей*, используется их основное свойство – обращение к элементам *последовательности* должно происходить строго последовательно. Поэтому для итераций циклов проверяется, что доступ к полям на них происходит строго монотонно и непрерывно – на каждой следующей итерации наименьшее смещение поля должно быть на единицу больше, чем максимальное смещение доступа на предыдущей итерации. В случае, если свойства монотонности и непрерывности для заданного цикла выполнены – весь диапазон полей, к которым осуществлялся доступ в цикле группируется в *последовательность*. При этом поля, обрабатывающиеся на каждой отдельной итерации, группируются в *запись*.

Для каждой найденной последовательности требуется определить, каким образом задаётся её длина. Выделяется несколько основных случаев:

- длина фиксирована протоколом и является постоянной,
- длина явно задаётся значением некоторого поля (поле длины),
- длина неявно задаётся специальным терминальным символом, то есть последовательность заканчивается полем-разделителем.

Алгоритм, определяющий способ задания длины последовательности является частью анализа семантики, описанного в разделе 3.6. В случае, если ни поля длины, ни поля разделителя для исследуемой последовательности найти не удалось, считается, что её длина фиксирована протоколом.

Для построения множеств релевантных инструкций для всех проекций требуется их полный просмотр сложностью $O(N)$ и построение отображения кода соответствующих циклов сложностью $O(\log M)$, где N - количество шагов в трассе, а M - размер статического кода программы в байтах. Для выделения последовательностей требуется ещё один полный просмотр проекций, с учётом множества релевантных инструкций. Таким образом, сложность всего алгоритма может быть оценена как $O(N \log M)$.

В разделе 3.8 рассматриваются возможности применения описаний функций, получаемых от пользователя через существующую подсистему, для уточнения результатов анализа. Один из способов применения - в качестве источника семантики, был описан в разделе 3.6. Также описываются способы применения

этих описаний на этапах восстановления границ полей и группировки полей в *последовательности* и *записи*.

В разделе 3.9 описываются различия, возникающие при анализе сетевых сообщений, и файлов. Уточняется термин *сетевое сообщение* - поясняются его отличия от термина *сетевой пакет*. В целом, раздел посвящён описанию подхода к восстановлению формата данных получаемых по частям, с использованием подсистемы описания функций. Необходимость в решении этой задачи возникает если файл считывается с диска по частям, или сообщение при передаче по сети разбивается на несколько сетевых пакетов. Это приводит к тому, что может не существовать единого буфера, содержащего весь анализируемый объект в заданный момент времени. То есть, одному сетевому сообщению или файлу может соответствовать несколько буферов в программе в разные моменты времени, каждый из которых содержит разные части данных анализируемого объекта. Ситуация осложняется тем, что в случае файлов доступна операция позиционирования, что позволяет считывать файл не полностью и в произвольном порядке. Это усложняет задачу определения, какой части исходного объекта соответствует некоторый буфер в памяти программы. Для решения задачи вводится и формализуется понятие *виртуального буфера*, соответствующего всему исследуемому сетевому сообщению или файлу. Так как для выполнения операций с данными файлов и сетевых сообщений, таких как чтение, запись, позиционирование, используются стандартные библиотечные вызовы, то для решения описанной задачи используется подсистема описания функций. Эта подсистема позволяет описывать формальные параметры функций и вычислять значения фактических параметров для каждого вызова функции в трассе. Описание функций, отвечающих за чтение, запись и позиционирование, позволяет вычислить параметры виртуального буфера и определить, каким его частям соответствуют считанные или записанные данные.

Одним из основных ограничений подхода к восстановлению формата на основе анализа исключительно сетевых трасс, без использования анализа кода, является его неприменимость к шифрованному (или сжатому) трафику. В разделе 3.10 описывается обобщение анализа данных, поступающих по частям на случай шифрованного трафика. Для выполнения анализа шифрованных данных необходима дополнительная информация, а именно:

- список функций дешифрования,
- описание этих функций, содержащее параметры, в которых находятся шифрованное сообщение и результат его дешифрования.

В этом разделе также описываются известные подходы к автоматическому получению этих сведений на основе динамического анализа.

Раздел 3.11 описывает алгоритм автоматического выделения сессий обмена данными с внешними по отношению к программе процессами. Для этого вводится атрибут *идентификатор сессии*, который сопоставляется некоторому параметру в описании функций получения данных. Решение этой задачи необходимо в случае, если исследуемая программа работает одновременно с несколькими источниками

данных, например, читает несколько файлов. Это обусловлено тем, что требуется отделять вызовы функций работы с разными потоками данных. Также эта информация необходима при решении задачи восстановлении автомата протокола.

С учётом наличия информации о вызовах функции в трассе сложность получения списка вызовов - O(logM), где M - количество функций. Считается что на момент анализа, фактические параметры всех вызовов функций вычислены. Проход по списку вызовов требует O(N) операций, где N - количество вызовов. Поиск среди сессий по значению идентификатора требует O(logK) операций, где K - количество сессий в трассе. Общая сложность алгоритма - O(logM) + O(NlogK).

В **главе 4** описываются методы обобщения и объединения форматов, полученных при анализе нескольких трасс. Методы описанные в главе 3 описывают решение задачи восстановления иерархического формата одного конкретного сообщения или файла. В тоже время, одна из целей восстановления формата сетевых сообщений - автоматическая генерация программ разборщиков для всех сообщений этого типа. Подобные разборщики используются в системах разбора сетевого трафика, таких как WireShark и системах предотвращения вторжений, таких как Bro. Следствием применения динамического анализа является то, что для анализа доступна только часть программы, участвующая в обработке некоторого конкретного сообщения. Кроме того, даже это сообщение может обрабатываться не полностью на конкретных входных данных, что приводит к неполноте получаемого формата. Для преодоления первого ограничения производится обобщение формата конкретного сообщения, таким образом, чтобы результирующий формат соответствовал более широкому классу сообщений этого типа. Второе ограничение может быть снято путём разработки метода объединения форматов, получаемых при анализе нескольких трасс.

Раздел 4.1 посвящён описанию алгоритма обобщения формата, позволяющему расширить класс сообщений, описываемых этим форматом. При передаче данных по сети один и тот же тип сообщений может использоваться для передачи различных команд и соответствующих им параметров. В качестве примера, можно привести протокол DNS, который содержит единственный тип сообщения. Причём это сообщение используется как для прямых DNS-запросов, когда передаётся IP-адрес, с целью получить доменное имя сетевого ресурса, так и для обратных DNS-запросов, когда передаётся доменное имя с целью получить соответствующий ему IP- адрес. Очевидно, что типы данных "IP адрес" и "доменное имя" имеют различный формат. Для описания обобщённого формата вводится новый тип группировки полей - *вариант*, позволяющий описывать формат частей буфера, которые могут хранить различные типы данных, в зависимости, например, от значений ключевых полей конкретного сетевого сообщения. Другим случаем, когда необходим такой тип группировки являются *плавающие* (необязательные) поля - которые могут присутствовать, а могут и отсутствовать в конкретном сообщении или файле. Например, в файлах BMP

может присутствовать поле описания палитры, в этом случае цвета отдельных пикселей задаются индексами в палитре. В случае отсутствия этого поля цвета задаются в формате RGB. Если описать формат в нотации БНФ, то вершине типа *вариант* в иерархическом описании формата можно сопоставить символ '|'. В частности, необязательное поле A может быть описано выражением

<необязательное поле A> ::= <поле A> | <>

Сложность алгоритма обобщения можно оценить как O(N). где N - количество элементов формата, так как каждый из них будет посещён не более одного раза.

В разделе 4.2 описывается метод объединения форматов, полученных при анализе нескольких трасс. Для того, чтобы объединить информацию о типах сообщений, требуется сначала произвести операцию выравнивания, то есть сопоставить элементы формата в одном сообщении элементам в другом, а затем произвести операцию объединения сопоставленных записей. Для того, чтобы произвести сопоставление элементов между собой требуется ввести операцию сравнения между элементами. При этом, следует учитывать, что результаты восстановления могут содержать ошибки разного рода. Среди таких ошибок можно выделить:

- размер некоторого поля был определён неверно,
- семантика поля была не восстановлена,
- не была восстановлена последовательность,
- разбиение последовательности на отдельные элементы выполнено неверно.

С учётом возможности ошибок такого рода в обоих восстановленных форматах, операцию сравнения имеет смысл ввести в виде функции, оценивающей не точное равенство, а степень совпадения двух элементов формата. Также нужно учесть возможность наличия необязательных полей в сообщении, которые могут быть в одном сообщении, а в другом отсутствовать, при том, что остальная структура сообщений идентична. Для сравнения используются характеристики элемента, такие как:

- базовый тип (поле, последовательность, структура, вариант),
- семантика поля,
- размер,
- смещение.

Также в некоторых подходах используется сравнение подмножеств кода, обрабатывающих это поле. Обычно сравнивают не сам код, что может быть проблематично, так как требуется разделять одинаковые инструкции, находящиеся по разным адресам, а подмножества адресов, по которым он находится. Однако привлечение этого критерия автоматически означает, что сравнивать нужно форматы, восстановленной по одной и той же реализации программы, загруженной по одним и тем же адресам в памяти, что не всегда удобно.

Для решения задачи такого нечёткого сравнения, с учётом наличия необязательных полей, был выбран алгоритм Нидлмана-Вунша, решающий

схожую задачу, однако на других исходных данных. Оригинальная версия этого алгоритма используется для выравнивания двух нуклеотидных последовательностей. Эффективность применения этого алгоритма при анализе сетевого трафика была показана в проекте Protocol Informatics. В качестве примера применения этого алгоритма можно привести сравнение двух сообщений протокола HTTP "GET /index.html HTTP/1.0" и "GET / HTTP/1.0". Результатом выравнивания будет:

```
1. GET /index.html HTTP/1.0
2. GET /___________ HTTP/1.0
```

Алгоритм Нидлмана-Вунша является алгоритмом динамического программирования на матрицах и сравнивает две символьных последовательности. Он состоит из трёх стадий: вычисление схожести, суммирование и обратный проход. Алгоритм работает на двух матрицах `S` (матрица совпадений) и `M` (матрица сравнения) размера $(m+1)\times(n+1)$, где m - длина первой последовательности, а n - второй.

Отличием решаемой задачи является, во-первых, то, что сравниваются не последовательности (линейные объекты), а деревья формата (двумерные объекты). Во-вторых, что элементы сравниваемых объектов имеют много характеристик, в отличие от элементов символьных последовательностей, и их сравнение также нетривиальная задача, особенно, с учётом возможных неточностей. Совокупность этих факторов потребовала существенной доработки алгоритма Нидлмана-Вунша. Оригинальный и доработанный алгоритмы подробно описаны в соответствующих разделах.

Сложность доработанного алгоритма можно оценить как сложность алгоритма Нидлмана-Вунша, применённого с учётом полного разворачивания иерархической структуры в линейную - $O(mn)$, где m и n - размеры матрицы M.

В **главе 5** описывается реализация описанных алгоритмов в виде системы восстановления форматов данных в среде анализа бинарного кода (раздел 5.1), а также результаты проверки реализованной системе на примере анализа сетевого протокола и файла с открытым форматом (раздел 5.2).

Среда анализа бинарного кода, разрабатываемая в ИСП РАН, представляет собой расширяемую среду анализа бинарных трасс программ, предназначенную для решения широкого класса задач обратной инженерии и анализа кода. С точки зрения разработанной системы, эта среда обладает следующими преимуществами.

- Имеет развитое SDK и поддерживает разработку модулей-расширений, предоставляя этим модулям всю информацию полученную различными алгоритмами при помощи системы интерфейсов.
- Система является архитектурно-независимой, то есть внутреннее представление с которым работают модули расширения не зависит от деталей реализации конкретной процессорной архитектуры. Добавление поддержки новой архитектуры происходит прозрачно для модулей - все алгоритмы работают с новой архитектурой без изменений. В настоящее

время, среда анализа бинарного кода поддерживает архитектуры x86-64, MIPS и ARM, PowerPC.

- В среде анализа бинарного кода уже имеются модули расширения, отвечающие за построение таких представлений программы, как граф потока управления и граф зависимостей по данным, а также подсистема описания функций. Эти данные необходимы для работы системы восстановления формата.

Общая архитектура реализованной системы приведена на рис. 5.

В общем случае, анализируемому объекту соответствует не буфер памяти, а виртуальный буфер. В качестве дополнительной информации, которая может использоваться системой для уточнения формата используется описание функций, участвующих в обработке сообщения. Эти данные могут вводиться пользователем с использованием подсистемы описания функций, реализованной в среде анализа бинарного кода. Список функций, описание которых позволит уточнить результат, может быть получен из алгоритма обработки сообщения.

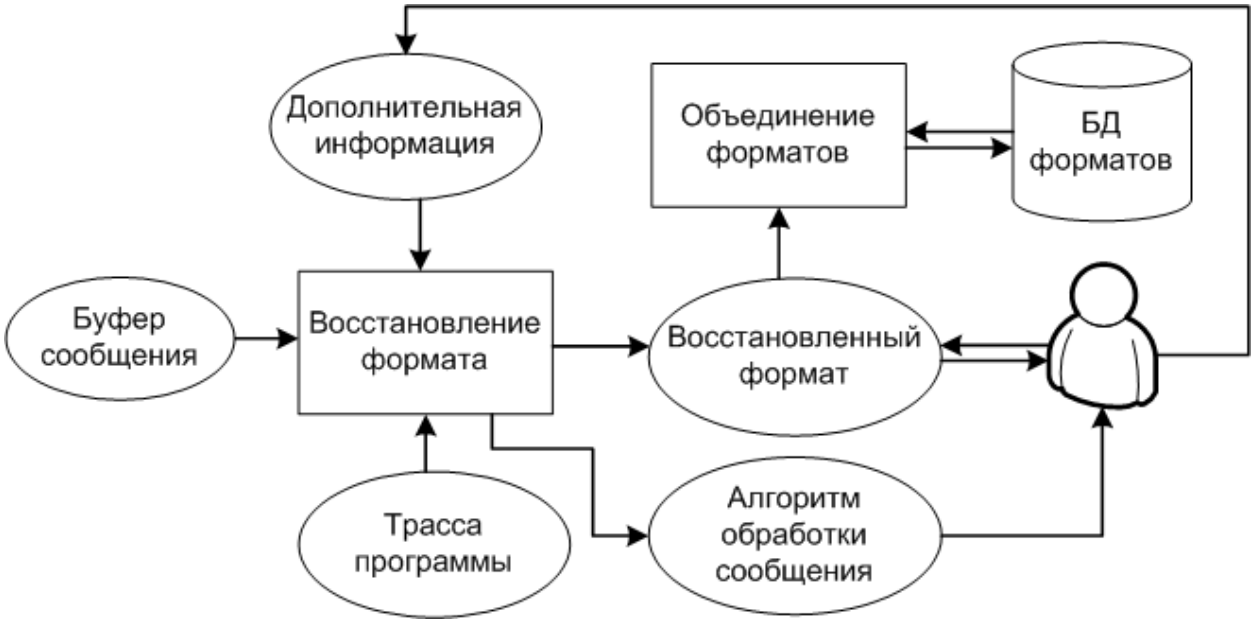

Рисунок 5. Архитектура реализованной системы восстановления формата данных

В разделе 5.2 приводятся результаты проверки реализованной системы на примере анализа открытых протоколов и форматов файлов. Среди сетевых протоколов рассматривается протокол DNS, среди файлов - BMP. Проверка состояла из трёх этапов. На первом этапе проверялась точность анализа формата одного сообщения, который описывается в главе 3. На втором этапе проверялась точность алгоритма обобщения формата. Последний этап соответствовал проверке алгоритма объединения форматов, полученных при анализе нескольких сообщений. На первом и последнем этапе проверки, получаемые форматы вручную сравнивались со спецификациями. Проверка на втором этапе осуществлялась путём автоматической генерации программы-разборщика и её

запуска на наборе сообщений, по которым восстановление формата не проводилось.

Анализ протокола DNS был выполнен по программе `nslookup`, входящей в стандартную поставку Windows 2000 на стандартном прямом запросе. Анализ формата BMP был выполнен по конвертеру форматов `pvw32con`, входящего в программу PictView. Результаты первого этапа проверки приведены в таблице 1 и позволяют говорить о возможности практического применения разработанных методов.

Таблица 1. Результаты проверки реализованного подхода

Формат данных	Лишние поля	Не выделены поля	Не выделены последовательности	Не восстановлена семантика
DNS	2	0	0	3
BMP	1	6	1	2

В таблице приведены ошибки алгоритма автоматического восстановления. Лишние поля возникли в результате особенностей кода обработки - код обрабатывал многобайтовые поля как несколько отдельных байт. Некоторые поля не были выделены, так как конкретная программа на заданном сценарии работы не использовала их значения. Последовательность байт в формате BMP не была выделена, так как обрабатывалась в обратном порядке. Семантика некоторых полей не была восстановлена, так как данные поля, на заданном сценарии не использовались по прямому назначению.

Для проверки точности алгоритма обобщения был разработан текстовый язык описания форматов в который могут быть транслированы получаемые форматы. Формат, представленный в таком виде, может быть отредактирован пользователем, а также уточнён путём анализа нескольких трасс, а затем автоматически оттранслирован в программу-разборщик на языке `binpac`, который является составной частью системы предотвращения вторжений Bro. По полученной грамматике формата DNS была автоматически сгенерирована программа разборщик прямых DNS-запросов. На вход программе-разборщику был подан набор сообщений, отличных от исходного. Все они были полностью и без ошибок разобраны, то есть каждому выделенному полю было сопоставлено его значение в разбираемом сообщении.

Для проверки алгоритма объединения был применён следующий подход. Анализировался общий формат DNS-сообщений на примере нескольких видов DNS-запросов (прямых, обратных и информационных), которые имели различную иерархическую структуру. Полученные форматы обобщались, а затем происходило их объединение. Результаты объединения сравнивались со спецификацией. В результате обобщения было выявлено 4 поля-варианта, и 1 плавающее поле. Одно поле-вариант является следствием неточности алгоритма, остальные результаты соответствуют спецификации.

В **заключении** перечисляются основные особенности реализованной системы, отличающие её от существующих аналогов. Описываются возможные

применения реализованной системы и получаемых с помощью неё результатов, а также планы развития разработанных методов. Среди основных направлений дальнейших работ по этой тематике можно выделить два наиболее важных.

- Разработка специализированных алгоритмов анализа формата исходящих сетевых сообщений (записываемых файлов).
- Разработка подходов к решению второй части задачи восстановления протоколов, а именно - автоматическое восстановление автомата протокола.

Основные результаты диссертационной работы.

1. Разработан метод автоматизированного восстановления форматов сетевых сообщений и файлов на основе информации, извлекаемой из бинарных трасс программ. Метод базируется на совместном использовании оригинального алгоритма анализа помеченных данных и структурного анализа графа потока управления.

2. Разработан метод объединения форматов данных, получаемых при анализе нескольких трасс, обеспечивающий повышение точности получаемых результатов, за счёт увеличения покрытия кода.

3. На основе предложенных автором методов, разработана и реализована подсистема восстановления форматов сетевых сообщений и файлов.

Публикации

1. А.И. Аветисян, А.И. Гетьман. Восстановление структуры бинарных данных по трассам программ. Труды Института системного программирования, том 22, 2012, с.95-118.
2. А.И. Гетьман, Ю.В. Маркин, В.А. Падарян и др. Восстановление формата данных. // Труды Института системного программирования, том 19, 2010, с. 195-214.
3. В.А. Падарян, А.И. Гетьман, М.А. Соловьёв. Программная среда для динамического анализа бинарного кода. // Труды Института системного программирования, том 16, 2009, с. 51-72.
4. В.А. Падарян, А.И. Гетьман. Автоматизация динамического анализа бинарного кода. // Материалы конференции "РусКрипто'2009". Москва, 2-5 апреля 2009
5. Гетьман А.И., Падарян В.А. Методика восстановления формата данных. Материалы конференции "РусКрипто'2010". Москва, 2-5 апреля 2010
6. А.И. Аветисян, В.А. Падарян, А.И. Гетьман и др. О некоторых методах повышения уровня представления при анализе защищённого бинарного кода. // Материалы XIX Общероссийской научно-технической конференции "Методы и технические средства обеспечения безопасности информации". Санкт-Петербург, 2010, с.97-98
7. А.И. Аветисян, В.А. Падарян, А.И. Гетьман и др. Возможности среды анализа бинарного кода ТРАЛ и актуальные направления её развития. //

Материалы юбилейной XX научно-технической конференции "Методы и технические средства обеспечения безопасности информации". Санкт-Петербург, 2011, с.120-123