Восстановление структуры бинарных данных по трассам программ

А.И. Аветисян, А.И. Гетьман
{arut, thorin}@ispras.ru

Аннотация. В данной работе рассматривается задача восстановления форматов бинарных данных. Предложены подходы к решению данной задачи, а также описывается реализация этих подходов в рамках системы восстановления форматов, разрабатываемой в ИСП РАН. Особенности одного подхода заключаются в методах восстановления сетевых сообщений и файлов, входящих и исходящих потоков бинарных данных. Помимо того, в работе предложен подход, основанный на механизме описания моделей функций, позволяющий решать задачу восстановления формата с гораздо большей точностью за счёт привлечения знаний пользователя об особенностях работы конкретной программной среды.

Ключевые слова: динамический анализ; бинарные трассы; восстановление форматов; анализ протоколов

1. Введение

Использование трасс, как источника данных о программе, подразумевает динамический подход к её анализу. Особенностями данного подхода являются, с одной стороны высокая точность, так как известна вся последовательность выполнявшихся инструкций, и значения всех регистров в процессе их исполнения, что, в свою очередь, даёт возможность вычислить все буферы данных (их адреса и размеры), с которыми работала программа. Ограничение данного подхода кроется во фразе «с которыми работала программа», так как для анализа в качестве источника данных доступна только та часть программы (её графа потока управления), которая выполнялась на тех данных, поданных на вход программе в процессе снятия с неё трассы. То же самое касается и обрабатываемых программой буферов памяти – известны только те буферы, с которыми программа работала на фиксированных входных данных. Ниже в данной работе будут приведены конкретные примеры, как эти ограничения влияют на точность анализа, с точки зрения восстановления структуры бинарных данных. Указанные ограничения (во всяком случае, полнота покрытия графа потока управления) могут быть частично сняты с привлечением некоторых аспектов статического анализа, однако при этом обычно резко падает точность анализа. Проблема баланса между этими подходами при их комбинировании остаётся открытой.

Статья организована следующим образом. В разделе 2 приводится краткий обзор работ по тематике восстановления формата и проводится их сравнительный анализ. В разделе 3 описываются возможные источники данных для анализа и ограничения накладываемые особенностями этих источников на применяемый подход. В разделе 4 приводится методика, используемая в системе, разрабатываемой в ИСП РАН, а также особенности её реализации. Раздел 5 содержит общее описание планируемых работ и направлений развития в рамках решения общей задачи восстановления протоколов.

2. Обзор работ

На данный момент существует большое количество работ по восстановлению форматов данных [1,2,3,4,5,6]. Обзор применяемых подходов и их анализ приведён в работе [7], которая описывает инструмент анализа формата, разрабатываемый в ИСП РАН. Используемое далее понятие "формат сообщения" подробно описывается в этой работе. Система восстановления формата реализована в виде модуля расширения к системе динамического анализа TrEx[8], так же разрабатываемой в ИСП РАН. Динамический подход, используемый в этой системе, основан на анализе «помеченных данных» (taint analysis)[9].

Подход	Входные данные	Точность	Получаемые данные
Анализ перехваченного сетевого трафика	Перехваченный сетевой трафик исследуемого протокола	Низкая. Зависит от точности анализа паттернов	Границы и возможные значения полей
Анализ статического представления программы	Программа (исполнимый файл), реализующая работу в рамках исследуемого протокола	Средняя. Сложность определения точных границ буферов сообщений	Границы (неточные) полей, все возможные варианты структур и последовательностей полей, ограничения на значения полей
Динамический анализ программы-клиента или сервера	Трасса исследуемой программы в процессе получения/отправки и обработки/генерации сетевого пакета	Высокая. Границы буферов точно известны.	Границы полей, структуры и последовательности полей для конкретного сообщения, ограничения на значения полей. Семантика полей.

Табл. 1. Существующие подходы к задаче восстановления форматов и их основные характеристики

Как видно по представленному в [7] обзору, существует три основных подхода к решению данной задачи, различающиеся по входным данным, точности и количество возможной информации о формате, которую можно получить при использовании данного подхода. Существующие подходы и их характеристики приведены в табл. 1.

Анализ перехваченного сетевого трафика, без анализа кода сетевого приложения, основан на выделение повторяющихся паттернов. Основным ограничением является невозможность восстановления семантики полей, так как полностью отсутствует информация о том, как именно используются значения указанных полей. Этот подход применяется в работе[6].

Анализ статического представления программы, реализующей работу в рамках исследуемого протокола (или работу с файлом исследуемого формата), основан на анализе возможных последовательностей вызовов функций чтения/записи в файл (получения/отправки пакета по сети). Для точного анализа необходимы знания о параметрах, содержащих собственно читаемый/записываемый буфер. Ограничение подхода связано с тем, что для восстановления формата необходимо уметь достаточно точно восстанавливать фактические значения этих параметров (размеры буферов), чтобы верно определять границы отдельных полей и структур. При чистом статическом анализе это представляет определённые сложности. В качестве варианта решения этой проблемы может быть использован алгоритм VSA (Value Set Analysis) описанный в работе[10]. Отдельную проблему представляет собой возможность того, что исследуемая программа может работать с несколькими файлами различных форматов, так как для каждой функции чтения/записи требуется уметь определять, с каким именно файлом идёт работа, что также непросто при статическом подходе. Статический подход рассматривается в работе [1].

Различные вариации динамического анализа программы-клиента или сервера в процессе получения/отправки и обработки/генерации сетевого пакета описаны в работах [2,3,4,5,7]. Причём работы [5, 7] основаны на развитии и обобщении идей, изложенных в работах [2,3,4]. При использовании этого подхода основным преимуществом является его точность - использование анализа "помеченных данных" [9], позволяет точно определять, с данными каких участков исходного буфера, содержащего сообщение, оперирует каждая конкретная инструкция. Основной недостаток подхода вытекает из его динамической природы - восстанавливается только та часть возможной структуры сообщения, которая была обработана при выполнении на фиксированных входных данных (конкретном сетевом сообщении или файле). Для частичной компенсации этого недостатка предлагается несколько различных методов по уточнению формата, за счёт объединения данных восстановленных по разным сообщениям (файлам) одного типа. Эти методы описаны в работах [4,5]. Методы отличаются областью применимости и точностью. Объединение информации о форматах подразумевает решение

двух задач: выравнивание и обобщение. Схема, поясняющая эти задачи, приведена на рис.1.

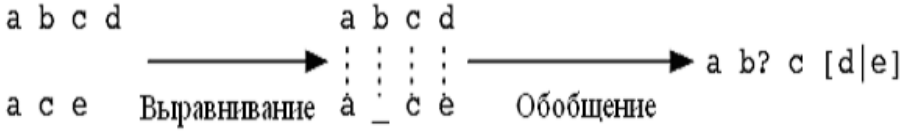

Рис.1. Объединение форматов двух сообщений. Буквами обозначены поля разных типов.

Подход, описанный в [4] для типизации полей (необходимой для решения задачи выравнивания) использует данные об адресах инструкций, что подразумевает, что при анализе формата использовалась одна и та же программа (одна её версия). Дополнительное требование состоит в том, что программа должна быть загружена в память по одним и тем же адресам (может не соответствовать действительности для ОС Windows, начиная с версии 7). При этом данный подход обеспечивает достаточно высокую точность. Подход работы [5] основан на генетическом алгоритме Нидлмана-Вунша[11], который используется для сравнения двух последовательностей нуклеотидов. Этот метод имеет более эвристическую природу и сравнительно меньшую точность, однако позволяет объединять информацию, полученную при анализе различных реализаций одного протокола и никак не привязан к особенностям используемой ОС. Вариант этого подхода также реализован в системе [7].

Основной тенденцией в области восстановления форматов сообщений является использование динамического анализа. Это можно пояснить тем, что основным требованием, в рамках данной задачи, является точность определения границ полей, так как именно на результатах этого этапа базируется дальнейший анализ структуры сообщения. Требуемую точность крайне трудно обеспечить в рамках статического подхода. Что касается анализа сетевого трафика, то, как уже было сказано, он неспособен предоставить информацию о семантике значений отдельных полей, что также сильно ограничивает область его применимости.

3. Источники данных для анализа

Источниками бинарных данных могут служить два типа объектов – сетевые соединения и локальные файлы. С точки зрения анализа структуры файлы являются более простым объектом, так как, во-первых, в них содержится только та информация, структура которой интересна исследователю – в отличие от сетевых соединений, где в сетевых пакетах передаётся большое количество вспомогательной информации, обеспечивающей собственно:

- возможность её передачи (например, координаты отправителя и адресата),
- надёжности доставки (например, номера пакетов для проверки полноты доставки),
- отсутствия возникновения искажения при передаче (например, контрольные суммы).

Другой особенностью файла является то, что его данные доступны целиком и однородны, в отличие от сетевых соединений, в которых данные передаются отдельными, как правило, небольшими порциями (пакетами). Пакеты, в свою очередь могут иметь различную структуру, описываемую в рамках протокола, по которому происходит конкретная коммуникационная сессия в рамках данного сетевого соединения. В зависимости от сложности и многообразия задач, для решения которых разрабатывался протокол, количество типов и сложность структуры сетевых сообщений, входящих в него, могут сильно варьироваться. Под сообщением здесь понимаются данные передаваемые сети на уровне приложения модели OSI. Вообще говоря, восстанавливать структуру данных можно на любом, произвольно выбранном уровне этой модели, однако более низкие уровни, как правило, хорошо стандартизированы, проверены и имеют известную структуру, а также, что немаловажно, относительно редко меняются. В то же время, количество протоколов уровня приложения растёт очень быстро, они в меньшей степени стандартизированы, многие из них являются проприетарными и закрытыми. Описанную тенденцию легко видеть на примере наиболее известной open-source системы разбора сетевых пакетов Wireshark[12]. Эта система имеет развитое API и SDK, что позволяет с минимальными трудозатратами добавлять в неё поддержку новых форматов сетевых пакетов в виде модулей расширений на языке C. Количество доступных на данный момент расширений исчисляется сотнями, и большинство из них относятся к протоколам пользовательского уровня.

3.1. Уточнение понятия "сообщение"

Следует отметить, что понятие "сообщение", в том виде как оно определено выше, не соответствует понятию "часть сетевого пакета". Между этими понятиями нет эквивалентности. Более того, ни одно из этих понятий не включает другое. Сказанное можно пояснить на следующих примерах. С одной стороны, протокол может позволять размещение в одном сетевом пакете более одного сообщения, что приводит к необходимости перед выполнением собственно анализа структуры выделять отдельные сообщения в рамках одного пакета. С другой стороны, большие сообщения могут быть разбиты на большое число сетевых пакетов, что соответствует парадигме сети с «коммутацией пакетов».

3.2. Особенности анализа пакетов разного уровня

Сложности анализа протоколов нижележащего уровня связана с одной стороны с используемым подходом, который подразумевает наличие точки в трассе, в которой существует единый буфер, содержащий всё "сообщение", а с другой техническими особенностями реализации подсистемы обработки сетевых сообщений ОС Windows NDIS (Network Driver Interface Specification)[13]. Эта подсистема сильно оптимизирована для максимально быстрой обработки нижних уровней стека протоколов. В результате этих оптимизаций единого буфера содержащего всё "сообщение" может не существовать. Таким образом, чтобы применять указанный подход требуется его предварительная модификация под существующие реализационные ограничения. В разделе 4.3 будет описан один из возможных подходов к решению данной проблемы.

3.3. Типы сообщений, с точки зрения их структуры

Обычно сообщения можно разделить на два класса – управляющие и информационные. Управляющие переводят программу-адресата (её автомат состояния протокола) в состояние, соответствующее приёму тех данных, которые будут переданы далее, либо в состояние соответствующе началу или концу сессии. Информационные сообщения содержат непосредственно данные, которые необходимо передать, информацию об их типе и структуре, данные для контроля целостности и т.д. Такая схема организации протокола соответствует протоколам, подразумевающим потенциально длительный обмен данными. Обычно их называют протоколами «с установлением соединения». Примером такого протокола может служить протокол TFTP, описанный в RFC1350[14]. Часто возникают другие задачи, в которых обмен данными организован по принципу "однократный запрос-ответ". В этом случае в рамках протокола может существовать всего один тип сообщения, но его структура может быть достаточно сложна, так как в него нужно инкапсулировать как управляющую, так и информационную составляющие. То есть оно должно содержать исчерпывающую информацию о типе запроса, который оно содержит и полное описание данных, которые в рамках него передаются. Если количество различных запросов относительно велико, а структура данных, передаваемых в рамках этих запросов, может сильно разниться, то сложность структуры сетевого сообщения может быть весьма высокой. Примером такого протокола (и сообщения) может служить протокол DNS, описанный в RFC1035[15] и RFC3425[16].

С точки зрения задачи восстановления формата первый тип протоколов гораздо проще для анализа, так как вся сложность, связанная с организацией взаимодействия удалённых узлов преобразуется в количество разных сообщений, в то время, как структура каждого отдельного сообщения довольно проста. В таких сообщениях, обычно, не встречаются последовательности структур и вложенные последовательности.

Управляющие сообщения, как правило, предельно просты и лаконичны и не содержат ни последовательностей, ни структур, а только относительно небольшой набор полей, задающих команду, содержащуюся в данном сообщении и необходимые параметры. Второй тип протоколов, напротив, представляет большую сложность и по тому, какой результат различные инструменты анализа показывают на сообщениях из этого класса, можно судить об уровне их развития. Сообщения из этого класса имеют высокую сложность организации структуры, содержат вложенные последовательности, поля указатели и т.д. Это является отражением того факта, что все возможные действия в рамках данного протокола должны быть переданы с помощью единственного сообщения. То, что разные команды протокола могут иметь сильно отличающиеся параметры, приводит к высокой вариативности структуры сообщения - для разных команд структура сообщения будет существенно различаться. В терминах грамматики восстанавливаемого формата это будет выглядеть, как большое количество дизъюнкций в правых частях правил вывода. В терминах статьи [7] это будет выглядеть, как большое количество полей-вариантов в дереве обобщённого формата. Для случаев таких сообщений, особенно актуальна задача объединения информации о формате, получаемой при анализе различных сообщений одного типа в рамках динамического подхода. Несколько вариантов решения этой задачи было рассмотрено выше.

3.4. Задача анализа протокола

Следует сказать, что задача анализа формата данных являясь с одной стороны самостоятельной и важной задачей обратной инженерии, в то же время, в контексте анализа сетевого трафика, является частью более общей задачи анализа протокола. Наряду с ней, частью задачи анализа протокола является задача восстановления автомата состояний протокола. Собственно наличие протокола, то есть списка правил обмена информацией также является свойством исключительно сетевых соединений (в отличие от файлов), что сильно усложняет их анализ. Автомат состояний протокола определяет, в каком состоянии находится разборщик протокола, то есть, фактически, какой набор сетевых сообщений он может принимать, а также возможные переходы между этими состояниями в зависимости от типов получаемых сообщений и их содержимого. Отдельный тип переходов соответствует переходам по таймеру, по истечению некоторого, обычно фиксированного в протоколе, времени, в случае, если за этот промежуток времени не произошло других событий (не было получено сообщений). Наиболее простой формой представления автомата состояний является граф, в котором вершинам соответствуют состояния, а рёбрам - переходы между ними, причём рёбра помечены условием соответствующего перехода (тип сообщения, его содержимое или истечение таймера). Пример графа автомата состояний протокола FTP (File Transfer Protocol) приведён на рис. 2.

Необходимым условием, для решения задачи восстановления автомата протокола является решение задачи восстановления формата - требуется восстановить форматы всех сообщений протокола и классифицировать их, то есть понять, какие сообщения относятся к разным типам. Это необходимо, чтобы понять, какой набор вершин (состояний) и рёбер (переходов) в принципе возможен в соответствующем графе автомата состояний.

До недавнего времени большинство усилий было сосредоточено на решение первоочередной задачи - автоматическом, либо максимально автоматизированном, восстановлении формата сетевых сообщений (и файлов). Однако в последнее время стали появляться статьи с новыми подходами к решению второй составляющей общей проблемы восстановления протоколов - задаче восстановления автомата состояния.

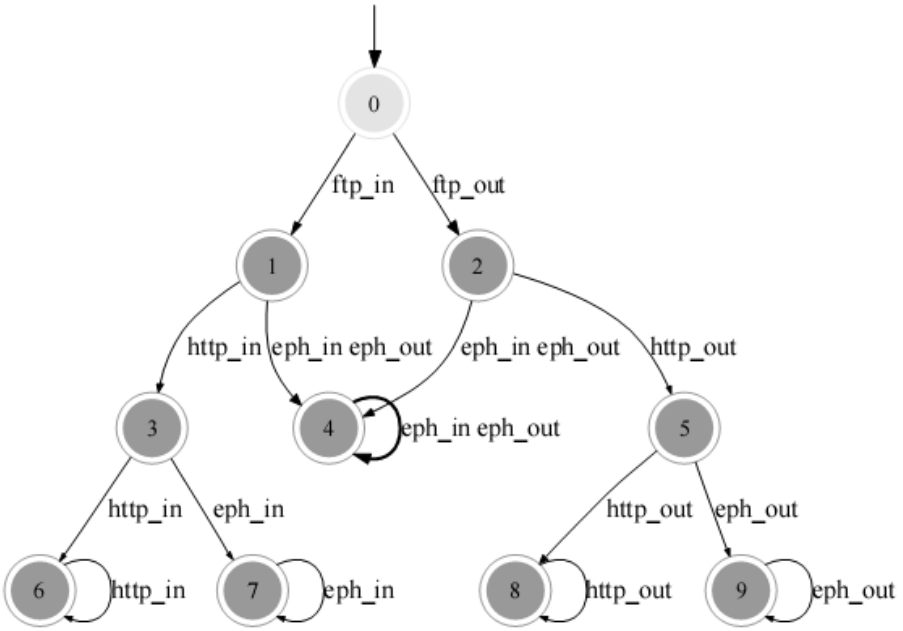

Рис.2. Представление автомата состояния протокола FTP в виде графа

4. Особенности реализованной системы

Работа [7] описывает систему восстановления форматов данных, разрабатываемую в ИСП РАН и реализованную в виде модуля расширения к среде динамического анализа TrEx. При разработке этой системы были обобщены и дополнены существующие методы. По сравнению с предыдущими работами можно выделить следующие нововведения.

- Использовалась битовая гранулярность представления памяти, что позволило анализировать поля-флаги.
- Реализована возможность анализировать совокупность вызовов функций получения пакетов (чтения из файлов), что позволило восстанавливать формат сообщений разбитых на несколько пакетов с одной стороны, и файлы с другой стороны, так как обычно файлы считываются именно по частям.
- Введено понятия "идентификатор источника", параметра, который идентифицирует объект, с которым работает конкретный вызов функции (например, сокет в случае сетевых пакетов). Это позволило автоматически разделять потоки сообщений, относящихся к разным соединениям.
- Усовершенствован алгоритм анализа циклов, что позволило восстанавливать последовательности с полями указателями. Пример таких последовательностей имеет место в DNS сообщениях, использующих сжатие строк.
- Для анализа циклов используется граф потока управления, восстановленный по трассе, что позволяет анализировать приложения, защищённые программами упаковщиками, для которых этот граф не может быть получен чисто статическими методами. Кроме того, использование такого графа потока управления и динамического подхода в целом, позволяет эффективно анализировать программы, защищённые от дизассемблирования, а также, содержащие обфускации вида "мёртвый код", "неисполняемый код" и т.п.
- Введено понятие "виртуальный буфер", соответствующее объекту, структура которого восстанавливается (всему сообщению или файлу). Необходимость этого понятия обусловлена рядом причин. В случае сетевых сообщений это вызвано тем, что может не существовать единого буфера, содержащего на некотором фиксированном шаге всё сообщения (например, в случае разбиения сообщения на отдельные пакеты). В случае файлов это обусловлено тем, что файл, как правило, считывается по частям и, возможно, непоследовательно, в результате чего простая "склейка" считанных данных может не соответствовать ни всему файлу, ни его части.

4.1. Особенности анализа файлов

Анализ файлов имеет ряд особенностей, являющихся следствием того, что данные файла сразу целиком доступны и доступ к ним может осуществлять в произвольном порядке. В случае сообщений это может быть не так - часто, для разбора некоторой части сообщения необходимо предварительно разобрать предшествующую ей часть. Результатом этого является ряд сложностей и ограничений возможности восстановления структуры файла с помощью динамического подхода. Одной из таких особенностей является

наличие функциональности позиционирования в данных файла (например, функция SetFilePointer API Windows), которая приводит к тому, что данные файла могут считываться и обрабатываться непоследовательно, не в прямом порядке и, более того, не полностью. То есть файл, с точки зрения модели доступа к его данным, представляет собой RAM (Random Access Memory). Это приводит, во-первых, к необходимости отслеживать, обработка какой части файла происходит в данный момент, во-вторых, следствием этого является то, что структура значительной части файла может быть не восстановлена в рамках динамического подхода, так как она не считывалась и не обрабатывалась. Эта особенность хорошо видна на примере файлов имеющих заголовок, в котором есть таблица смещений отдельных таблиц данных в этом файле. В качестве конкретного примера можно взять формат исполнимого файла PE32[17]. Основная часть файла состоит из отдельных секций (кода, данных и др.), смещения которых в файле описываются в таблице секций (section table), которая находится в заголовке файла. Другой пример будет приведён в разделе 4.6. Таким образом, возникает подзадача контроля позиции в файле, с одной стороны, за счёт суммирования размеров считанных (записанных) буферов, а с другой стороны, за счёт отслеживания вызовов функции (функций) позиционирования.

Одной из причин того, что файл читается не целиком, может быть избыточность содержащейся в нём информации (часть информации тем или иным способом дублируется), порождённая, с одной стороны, развитием формата и появлением в нём новых полей данных, а с другой необходимостью поддержки обратной совместимости. Примером подобной избыточности формата файла может служить формат для хранения растровой графики BMP[18], разработанный фирмой Microsoft. В начале файла идёт структура BITMAPFILEHEADER, которая в частности содержит поля, хранящие размер файла и смещение собственно данных изображения от начала файла.

Далее, в зависимости от версии файла, идёт одна из следующих структур:

- BITMAPINFOHEADER
- BITMAPV4HEADER
- BITMAPV5HEADER

Данные в начале этих структур совпадают и содержат информацию о:

- Размере самой структуры
- Ширине и длине изображения в пикселях
- Количестве битов, описывающих каждый пиксель

Очевидно, что на основе этих данных можно вычислить размер файла, и таким образом, поле структуры BITMAPFILEHEADER, хранящее размер файла становится избыточным. Это приводит к тому, что различные приложения могут использовать вычисления на основе значений различных полей или их комбинаций для достижения одного и того же результата – получения размера файла. Кроме того, одни приложения могут проверять, что скоррелированность значений в различных полях структур, а другие могут

просто игнорировать избыточные с их точки зрения поля. Кроме того, структура BITMAPFILEHEADER, содержит несколько зарезервированных полей, которые, согласно формату, должны быть заполнены нулями, что как показывает практика, проверяется далеко не каждым приложением. Пример восстановления формата BMP файла будет приведён ниже.

Следует отметить, что некоторые из особенностей, описанные выше для файла и связанные с возможностью позиционирования, могут быть ограниченно применимы и к сетевым сообщениям. Особенно это касается сложных сообщений типа DNS. В рамках сообщений таких протоколов считывание данных отдельных пакетов (но не пакетов в сетевом потоке) может происходить непоследовательно и не полностью. С точки зрения реализации это организуется с помощью полей-указателей – ячеек памяти пакета, содержащих смещение других полей относительно известных точек (например, начала сообщения). Неполнота обработки, как и в случае файлов, может быть связана с избыточностью сообщения, которая, в свою очередь, может быть порождена развитием протокола во времени и необходимостью поддержки обратной совместимости с программными продуктами, которые были разработаны для более ранних версий протоколов.

4.2. Особенности анализа входящих и исходящих данных

Следует отметить существенное различие подходов при восстановлении протокола по входящим сообщениям (считываемым файлам) и по сообщениям, которые формируются на выходе исследуемой программы (записываемые файлы). В случае входящих сообщений, проще определить семантику различных полей, таких как поля длины, контрольные суммы, разделители, так как они используются по назначению в коде программы. Например, поля длины используются как максимальные значения счётчиков итераций в цикле обработки последовательности. В случае анализа формата исходящего сообщения (выходного файла) часть этой семантики можно восстановить по способу формирования значения поля (контрольная сумма зависит от значений других полей), а часть семантики восстанавливается только в самом общем виде («магические константы» протокола тяжело отличить от разделителей). С другой стороны, границы полей и сообщений в выходном потоке, как правило, определять проще, так как они часто уже известны в момент вывода. Другой важный аспект - явное тяготение разработчиков к анализу форматов входящих данных в рамках динамического подхода. Этот вид анализа описывается в работах [2,3,4,5]. Это, по-видимому объясняется тем, что базовый структурный анализ сообщения во всех перечисленных работах, в том или ином виде, основывается на хорошо разработанном методе анализа "помеченных данных"[9]. В тоже время, до недавнего времени не было известно базового подхода, позволяющего эффективно анализировать структуру формирующегося пакета. Только в работе [19] был предложен первый оригинальная методика "деконструкции буфера" (buffer deconstruction) к восстановлению структуры входящих данных

на основе динамического анализа. Идея методики заключается в том, что программа сохраняет поля, структуры и последовательности в отдельных буферах памяти и создаёт сообщения путём комбинации этих буферов и копировании их в буфер сообщения. Соответственно, алгоритм осуществляет итеративный поиск соответствующих буферов назад по трассе от точки отправки сообщения (в этот момент его буфер полностью сформирован). Другой важной особенностью работы является чёткое разделение алгоритмов восстановления как структуры входящих и исходящих данных, так и подробный анализ того, какая именно семантика и как именно, может быть с достаточной точностью восстановлена для входящих и исходящих данных. Разница в восстановлении структуры хорошо просматривается на примере базового алгоритма, лежащего в основе большинства подходов к восстановлению на основе динамического анализа: анализ "помеченных" данных в случае входящего потока данных и "деконструкция буфера" в случае исходящего потока. Анализ семантики состоит из двух несвязанных подходов - структурный анализ графа потока управления и выводы о типах на основе его особенностей (так восстанавливают поля длины и поля-разделители) и, так называемый, алгоритм "выведения типов" (type inference), который позволяет делать выводы о семантике содержимого полей на основе того, как это содержимое используется в точках где семантика известна (например, в качестве параметров функций известного вида, типа обработки строк). В частности, в работе приведена табл.2, показывающая, какая именно семантика и для какого типа данных (входящих или исходящих) может быть эффективно и с достаточной точностью восстановлена.

Также в этой работе предлагался метод выделения из общей бинарной трассы сетевого диалога отдельных подтрасс, соответствующих получению и обработке (или формированию и отправке) отдельных сообщений. В рамках метода была предложена качественная эвристика, позволяющая определять, какие пакеты соответствуют одному сообщению, в случае, если сообщение было разбито на несколько пакетов. Эвристика сводится к определению наличия или отсутствия зависимости по данным между максимальным размером текущего получаемого пакета (параметр функции получения) и данными какого либо из предшествовавших пакетов.

Связь между подходами к анализу структуры входящих и исходящих бинарных данных во многом аналогична связи между прямым и обратным слайсингом, хотя в случае слайсинга, первым появился именно обратный анализ[20], эффективно применявшийся для целей отладки, и, лишь впоследствии, идеи были обобщены и разработан прямой анализ[21]

Field Semantics	Received	Sent
Cookies	yes	yes
IP addresses	yes	yes
Error codes	no	yes
File data	no	yes
File information	no	yes
Filenames	yes	yes
Hash / Checksum	yes	yes
Hostnames	yes	yes
Host information	no	yes
Keyboard input	no	yes
Keywords	yes	yes
Length	yes	yes
Padding	yes	no
Ports	yes	yes
Registry data	no	yes
Sleep timers	yes	no
Stored data	yes	no
Timestamps	no	yes

Табл. 2. Виды семантики полей, которые могут быть восстановлены для входящего и исходящего потока данных

4.3. Система описания моделей функций

Во многих подзадачах, возникающих в процессе решения задачи восстановления форматов и более общей задачи восстановления протоколов, требуется информация о функциях и их параметрах. Как правило, требуется иметь возможность задавать набор описаний некоторой группы функций и их параметров. Ярким примером может служить процесс автоматизации выделения отдельных потоков сообщений в полной трассе, содержащей сетевой диалог, а также отдельных алгоритмов получения и обработки отдельных сообщений в каждом потоке. Один из вариантов такого алгоритма был реализован в системе TrEx. Для работы алгоритма необходимо задать функции получения и отправки сообщений. В случае работы с файлом -

функции чтения/записи и позиционирования. После этого нужно описать параметры этих функций:

- Идентификатор объекта, с которым идёт работа: сокет в случае сетевого соединения, идентификатор файла, в случае файла
- Буфер, содержащий отправляемые/получаемые данные
- Смещение в файле для функции позиционирования

Собственно семантика параметров описывается с помощью системы описания семантики, которая будет рассмотрена в разделе 4.4. В данном разделе речь идёт не о задании конкретной семантики параметра, а об определении его места в памяти, то есть способе получения ячейки памяти (или регистра), содержащей значение фактического параметра для каждого конкретного вызова.

Приведём формальное определение того, что мы называем моделью функции. Модель представляет собой набор следующих атрибутов:

- Имя модели (функции может соответствовать несколько моделей)
- Границы модели
  - Адрес входа в функцию
  - Набор адресов возврата
- Список описание параметров (входных и выходных)
- Список зависимостей между входными и выходными параметрами

Приведём несколько характерных примеров, когда функции может соответствовать несколько моделей. Во-первых, многие функции ведут себя существенно различным образом (генерируют различные выходные параметры), в зависимости от того, произошла в ходе их выполнения ошибка или нет. Например, если функция получения сетевого пакета recv завершилась успешно, она возвращает размер полученных данных. В случае если произошла ошибка, она возвращает код ошибки. Другим примером являются функции-обработчики прерываний, которые в качестве одного из параметров принимают код ситуации, которая имела место и в зависимости от этого осуществляют различные виды обработки. Подобное поведение также характерно для функций-диспетчеров.

Описание параметра содержит имя параметра, его тип и описание способа получения ячейки памяти, содержащей значение фактического параметра для каждого конкретного вызова. По типу параметры делятся на стековые, регистровые и параметры-выражения.

Стековые параметры передаются вызову через стек, соответственно описание получения содержит смещение и размер параметра в стеке. Регистровые параметры передаются (или возвращаются) через регистры. Описание параметра содержит имя регистра. Параметры-выражения соответствуют данным, передаваемым через буферы памяти, адрес и размер которых определяются другими параметрами. Описание параметра выражения содержит следующие части:

- Опциональный сегментный регистр (для платформы x86)
- Арифметическое выражение, результат вычисления которого даёт адрес буфера. В качестве переменных в выражении могут фигурировать имена ранее описанных параметров.
- Арифметическое выражение (которое может быть константой), результат вычисления которого даёт размер буфера (или его верхнюю границу - см. ниже)
- Опциональный тип содержимого буфера - ASCII или Unicode строка

Указание сегментного регистра наиболее актуально для 16-разрядного кода. Если сегментный регистр не указан, то арифметическое выражение задаёт линейный адрес буфера. В противном случае, оно задаёт логический адрес, который транслируется в линейный с учётом значения соответствующего сегментного дескриптора.

В случае если задан тип содержимого буфера, значение выражения для размера интерпретируется как верхняя граница размера буфера, а его фактический размер определяется на основе его содержимого. Например, для ASCII строки её фактический размер это позиция первого символа «0» в этой строке.

Список зависимостей между параметрами состоит из пар входной-выходной параметр, которые задают зависимости по данным между этими параметрами. Наличие зависимости означает, что значение заданного входного параметра влияет на значение заданного выходного параметра и, наоборот, значение выходного параметра формируется на основе значения заданного входного параметра. Нужно отметить, что в данном случае не рассматривается вопрос о конкретном функциональном виде зависимости между параметрами.

Значения фактических входных параметров некоторого вызова определяются, как значение соответствующей параметру ячейки памяти или регистра в точке вызова функции, а значения выходных параметров - как значения соответствующих ячеек в точке выхода из функции.

4.4. Система описания семантики функций и их параметров

После того, как некоторая функция описана, часто требуется задать некоторые её свойства и свойства её параметров, для того чтобы соответствующие алгоритмы могли выделить эту функцию из общего набора и определить важные для них параметры. Один из примеров, когда такая информация о функциях может быть необходима, был приведён выше. Другой пример, о котором пойдёт речь в разделе 5.1 - анализ шифрованных сообщений, для проведения которого необходимы знания о функциях шифрации/дешифрации. Существует несколько подходов, позволяющих выделять такие функции автоматически (ни будут рассмотрены в соответствующем разделе). Однако в некоторых случаях нужно предоставить пользователю возможность задать такие функции явно. Реализованная система конструктивно состоит из двух подсистем:

- Подсистема описания семантики функций и параметров, без привязки к конкретным функциям и их параметрам
- Подсистема сопоставления описанной семантики конкретным моделям функций и их параметров

Обе подсистемы хранят свои данные в БД MySQL. Формально семантика функции описывается набором следующих атрибутов:

*Имя семантики
*Уникальный идентификатор семантики
*Текстовое описание семантики
*Набор связанных семантик параметров

Семантика параметра это:

*Имя семантики параметра
*Глобально уникальный (среди идентификаторов семантик параметров всех функций) идентификатор
*Текстовое описание семантики параметра

Эти атрибуты задаются при описании новой семантики в рамках первой подсистемы. Имена и текстовое описание необходимы для организации взаимодействия с пользователем в рамках графического интерфейса. За это взаимодействие отвечает вторая подсистема.

4.5. Общая схема подхода

Исходной точкой анализа формата бинарных данных является получение трассы – этот процесс подробно описан в статье [7]. Однако дальнейшая последовательность действий, по сравнению с описанной в этой статье, претерпела значительные изменения, связанные, в первую очередь, с повышением уровня автоматизации и расширением области применения подхода. После того, как трасса получена, последовательность действий аналитика следующая:

1. Описать модели функций получения/отправки пакета, либо чтения/записи/позиционирования файла – в зависимости от вида исследуемого потока бинарных данных с помощью системы описания моделей. Эти данные могут быть получены на основе:
   - Проведённого исследования бинарного кода (в случае анализа неизвестной программной системы)
   - Подключённой символьной информации
   - Знаний, полученных на основании анализа этой же программной системы в других проектах
2. Описать соответствующие виды семантики функций и их параметров в подсистеме описания семантики системы работы с семантикой функций
3. Задать семантику каждой из описанных функций и семантику её параметров с помощью подсистемы привязки семантики функций и параметров к описанным ранее моделям

4. В случае, если в трассе присутствует несколько потоков бинарных данных – задать значение идентификатора исследуемого потока
5. Указать идентификатор исследуемого бинарного потока, в случае, если таких потоков несколько

Получив всю необходимую информацию, система автоматически найдёт все вызовы интересующих функций, построит «виртуальный буфер», определит, к какой его части относится каждый вызов функции чтения или записи, восстановит содержимое соответствующих частей, а затем и структуру всего буфера.

4.6. Проверка реализованной системы

Для проведения экспериментов использовалась рабочая станция со следующими характеристиками: 2 ЦПУ Intel Xeon E5540 2.53 GHz, 48 GB RAM, 64-разрядная ОС Windows 7 SP1. Трассировка выполнялась в симуляторе AMDSymNow v4.5.1, в виртуальной машине была установлена ОС Windows XP SP2. Из библиотек ОС виртуальной машины были предварительно извлечены экспортируемые символы, которые затем использовались для разметки трассы.

Требовалось восстановить формат неизвестного файла, который подавался на вход сторонней программы, доступной в виде бинарного файла. Впоследствии было выяснено, что исследуемый файл содержит исполняемый код, который интерпретируется данной программой. Исследуемая программа принимает на вход бинарный файл testfile.bin размером 2100 KB и выводит некоторые данные на консоль. Так как вывод был достаточно объёмен, а обновление экрана в виртуальной машине при снятии трассы сильно замедлено - вывод был перенаправлен в файл.

Для применения алгоритма восстановления требовалось найти точку (или точки) ввода данных из файла. Первоначальный поиск функции ReadFile не дал результатов. Вторым вариантом получения данных из файла является его прямое отображение в память и чтение этой памяти без использования функции ReadFile. Изучение символьной информации позволило установить, что для отображения использовалась функция ZwMapViewOfSection. Восстановление её параметров позволило выделить область памяти, в которую был отображён файл. Восстановление формата осуществлялось от точки возврата данной функции до конца трассы. По результатам восстановления буфера, содержащего указанный файл, был сделан вывод о том, что фактически читалась незначительная часть файла, причём файл читался непоследовательно - с разрывами. Содержимое считанных областей было восстановлено. В результате автоматического анализа формат считанной части файла был восстановлен. Результаты восстановления выявили следующие аспекты:

- файл содержал несколько полей-указателей, хранящих смещения данных в файле (что соответствует формату исполняемого PE-файла)

- восстановлено некоторое количество ключевых полей со значением 0x66 (по-видимому, части префиксов машинных команд)
- восстановлены последовательности, содержащие двухбайтные префиксы
- восстановлены флаговые поля, содержащие от двух до трёх групп флагов.

Основная масса данных интерпретировалась как отдельные байты, что также соответствует исполняемому файлу.

5. Дальнейшие работы

5.1. Анализ шифрованного трафика

Следующим шагом в области восстановления форматов сообщений явилась разработка подходов для расширения области применимости существующих методов на шифрованный трафик. Необходимость развития в этом направлении возникла прежде всего вследствие распространения вирусов, организовывающих ботнеты из заражённых компьютеров. Для контроля ботнетов и их обезвреживания потребовались инструменты для проведения в краткие сроки анализа протоколов обмена сообщениями в рамках этих сетей.

Подробное описание модели функционирования ботнетов, а также методы их контроля и обезвреживания содержатся в работе [19]. Так как цель злоумышленников - максимально затруднить анализ как кода, так и командного протокола и, таким образом, продлить существование ботнета, то все сообщения передаются в зашифрованном виде. Одним из примеров такого рода вирусов является agobot, осуществляющий шифрацию своих сообщений с помощью собственной реализации SSL. Одной из первых работ в этом направлении является описание системы ReFormat[22], тестирование которой производилось как раз на примере анализа протокола agobot.

Идея подхода заключается в том, чтобы автоматически найти в трассе обработки сообщения точки, где сообщение уже полностью прошло через стадию дешифрации (или, в случае отправки сообщения, уже полностью сформировано, но ещё не зашифровано) и определить буферы, в которых в этот момент содержатся данные сообщения. Фактически, эти точки соответствуют вызовам функций шифрации и возвратам из функций дешифрации. Их поиск, таким образом, сводится к выделению функций шифрации/дешифрации. В качестве основного признака была выбрана повышенная концентрация арифметико-логический операций в этих функциях. Эмпирически была установлен порог процента арифметических операций, позволяющий с высокой долей уверенности идентифицировать функции шифрации/дешифрации.

Проведённые эксперименты над рядом как открытых, так и закрытых шифрованных протоколов (IRC, HTTPS, MIME, agobot) показали, что этот

порог может быть достаточно произвольно взят в пределах [0.25, 0.8]. На практике был взято значение 0.5. Ограничением описанного подхода является, во-первых, то, что в нём рассматриваются трассы содержащие получение и обработку одного сетевого сообщения и не описывается инструмента выделения таких трасс из трассы содержащей полноценный сетевой диалог. Кроме того, явно предполагается наличие ровно двух отдельных фаз расшифровки и обработки сообщения, причём вторая должна строго следовать за первой.

Эта модель может быть неприменима в случае, если только часть сообщения шифруется, причём шифрованная и незашифрованная части чередуются. В этом случае количество фаз обработки и дешифрации увеличивается, причём эти фазы чередуются в трассе. Это ведёт к тому, что отсутствует единая граница между этими двумя фазами (поиск которой, собственно, осуществляет алгоритм системы ReFormat). После того, как граница между фазами найдена, требуется определить буфер (или буферы) содержащий дешифрованные данные сообщения в граничной точке. Для этого применяется следующий критерий: буферы содержащие сообщение должны обладать рядом свойств. А именно, их данные должны:

- быть "помеченными" (зависеть от данных полученного сообщения)
- записываться на фазе дешифрации и читаться на фазе разбора

То есть, эти буферы можно получить в три этапа:

1. Провести анализ "помеченных данных"
2. Восстановить записываемое множество (write set) фазы дешифрации и читаемое множество (read set) фазы разбора
3. Пересечь эти три множества

Отдельная проблема возникает, если таких буферов оказывается несколько - необходимо понять в каком порядке и с какими смещениями содержат эти буферы расшифрованные данные сообщения. В работе эта проблема решается следующим эвристическим методом:

1. Буферы выстраиваются в том порядке, в котором осуществляется их чтение на фазе разбора
2. Их содержимое рассматривается, как единый "виртуальный буфер", содержащий всё расшифрованное сообщение

Следует отметить, что такой метод применим не всегда - если некоторая часть сообщения была дешифрована дважды (или была сделана копия дешифрованных данных), то применение данного метода сильно исказит структуру сообщения, так как в "виртуальном буфере" будет содержаться несколько одинаковых частей сообщения.

Реализация этого метода выполнена в виде отдельного независимого модуля, которые может применяться в комплексе с произвольной системой восстановления форматов, расширяя её возможности. Для тестирования использовалась система восстановления [4].

Развитие и обобщение этого подхода является предметом уже упоминавшейся работы [19]. В этой работе содержится описание инструмента Dispatcher, проверка которого осуществляется на примере анализа командного протокола (C&C protocol) достаточно широко известного вируса MegaD. Сообщения командного протокола этого вируса обладают указанными выше свойствами (делающими невозможным прямое применение подхода из работы [22]) : они содержат незашифрованную часть - размер пакета (Msg_len), который обрабатывается вначале, за которым идёт шифрованная часть, содержащая основные данные пакета (Payload). Структура пакета MegaD приведена на рис. 3.

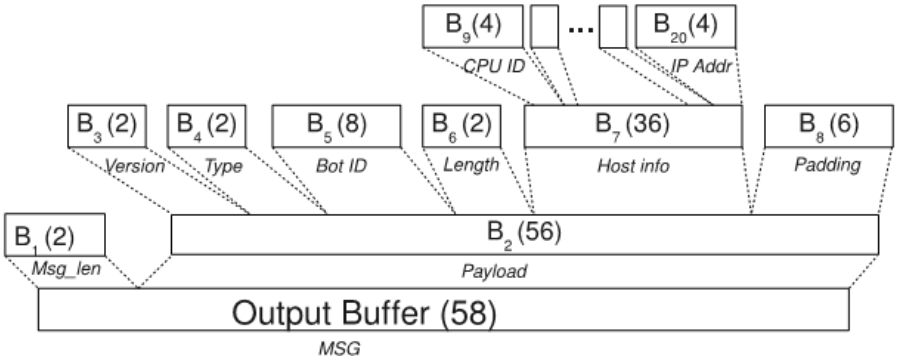

Рис. 3. Структура пакета управляющего пакета вируса MegaD.

Поход из работы [22] был развит в двух направлениях. Во-первых, было снято ограничение на то, что должна существовать единственная граница между фазами дешифрации и разбора. Вместо этого на основании тех же эвристик производилась идентификация вызовов (и, соответственно, функций) шифрации/дешифрации, после чего все вызовы этих функций, параметры которых оказывались "помечены" (зависели от потока входных данных) считаются границами между этими фазами. Во-вторых, методика была обобщена на случай шифрации исходящих пакетов (ReFormat обрабатывал только входящий трафик). Для идентификации буферов, содержащих данные сообщения непосредственно перед вызовом функции шифрации, применяется следующий алгоритм:

1. Для всех вызовов функции дешифрации вычисляется читаемое множество (read set)
2. Восстанавливаются значения всех этих буферов (фактические параметры вызовов)
3. Для каждого вызова выбираются те буферы, содержимое которых меняется от вызова к вызову. Это обусловлено тем, что читаемые данные содержат два типа информации - собственно данные для

шифрации и таблицы шифрования, которые не должны менять своё содержимое.

На данный момент, в рамках среды TrEx разрабатывается алгоритм автоматического поиска функций шифрации/дешифрации, на основе комбинации подходов ReFormat и Dispatcher. Результаты работы этого алгоритма представляют собой два вида информации:

- Модели функций шифрации/дешифрации, добавляемые в систему описания моделей
- Семантика функций и их параметров-буферов, добавляемая в систему описания семантики

5.2. Другие приложения результатов восстановления формата

При рассмотрении формата BMP приводились примеры возможных различий при реализации разборщика этого формата. Это может служить примером отличия формата, описанного в документе, от формата, поддержка которого реализуется в приложении. Так, например, приложения не проверяющие, что зарезервированные поля заголовка BITMAPFILEHEADER содержат нули, по сути, поддерживают более широкий класс сообщений, чем тот, что описан в документе. Такие особенности реализации могут использоваться для идентификации конкретных приложений обработчиков (или их классов). Подход, занимающийся изучением таких приложений, называется deviation detection (поиск отклонений). Его результаты, в свою очередь могут использоваться для поиска ошибок (error detection) и создания «отпечатков» (fingerprint generation) конкретных приложений, использующихся для их однозначной идентификации. Анализ отклонений при поиске ошибок обусловлен тем, что указанные особенности реализации часто могут приводить к ошибкам. Указанный подход подробно рассмотрен в работе [25].

6. Заключение

Большое количество работ по теме восстановления протоколов и форматов файлов от различных коллективов по всему миру, говорит о том, что это направление - бурно развивающаяся область. Получено множество важных результатов, в том числе и для зашифрованных протоколов, которые позволяют говорить о том, что в ближайшие годы, по-видимому, процесс решения данной задачи будет автоматизирован настолько, что трудозатраты на анализ каждого нового протокола будут сведены до разумных пределов. Это позволит анализировать протоколы гораздо менее подготовленным специалистам. В свою очередь, это повлечёт за собой бурный рост количества анализируемых протоколов (качество анализа также возрастёт), что позволит, с одной стороны, в сжатые строки устранять ошибки во вновь разрабатываемых протоколах, а с другой, обнаружить и ликвидировать "дыры" в тех, что широко используются на сегодняшний день.

Кроме того, уже сейчас в связи с высоким уровнем разработанности задачи восстановления форматов появляется большое количество потенциальных приложений результатов её решения:

- разработка open-source ПО на основе закрытых протоколов. Примером в области форматов файлов может служить пакет OpenOffice, поддерживающий закрытый формат doc фирмы Microsoft, а в области сетевых протоколов - мультипротокольный интернет-пейджер qutIM, поддерживающий, в частности, закрытый протокол ICQ OSCAR[26] (спецификация этого протокола была позже открыта и опубликована)
- использование форматов в антивирусных средствах и ПО для обнаружения и предотвращения вторжений (IDS и IPS соответственно), развитие подходов на основе сигнатурной фильтрации (signature-based filtering)
- автоматическая идентификация приложения по генерируемому и принимаемому им сетевому трафику (application fingerprinting). В частности это может использоваться в системах автоматического анализа и фильтрации трафика (DPI системы)
- автоматический анализ протоколов зловредного ПО (malware), что особенно актуально для программ, создающих ботнеты. Такой анализ позволяет реагировать на появление нового ботнета в режиме реального времени, контролировать его размер и уничтожать, не допуская чрезмерного разрастания
- автоматическая генерация сообщений протокола с целью его тестирования, например, по технологии фаззинга (fuzzing)
- улучшенный анализ сетевого трафика на ключевых узлах обмена с целью получить точную статистику и принять необходимые административные и технические решения для обеспечения качества связи

Список литературы

[1] J.Lim, T. Reps B. Liblit. Extracting Output Formats from Executables. // Proceedings of the 13th Working Conference on Reverse Engineering, 2006, pp. 167—178.

[2] J. Caballero , H. Yin ,Z. Liang ,D. Song . Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. // In Proceedings of the 14th ACM Conference on Computer and and Communications Security, 2007, pp. 317—329.

[3] Z. Lin, X. Jiang, D. Xu, X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. // In Network and Distributed System Security, 2008.

[4] G. Wondracek, P. Milani Comparetti, C. Kruegel, E. Kirda. Automatic Network Protocol Analysis. // In 15th Symposium on Network and Distributed System Security, 2008.

[5] W. Cui, M. Peinado, K. Chen, H. J. Wang, L. Irun-Briz. Tupni: Automatic Reverse Engineering of Input Formats. // Proceedings of the 15th ACM conference on Computer and communications security, 2008, pp. 391—402.

[6] W.Cui, J. Kannan, H. J. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. // Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, 2007, pp. 14:1—14:14
[7] А.И. Гетьман, Ю.В. Маркин, В.А. Падарян, Е.И. Щетинин. Восстановление формата данных. // Труды Института системного программирования РАН, том 19, 2010, стр. 195-214
[8] В.А. Падарян, А.И. Гетьман, М.А. Соловьев. Программная среда для динамического анализа бинарного кода. // Труды Института системного программирования РАН, том 16, 2009, стр. 51-72
[9] J. Newsome and 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), 2005
[10] G. Balakrishnan and T. Reps. Analyzing Memory Accesses in X86 Executables. // In 13th International Conference on Compiler Construction, 2004. pp. 5—23
[11] S. Needleman, C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. // Journal of molecular biology. 1970 Mar;48(3), pp. 443-453.
[12] Wireshark. http://www.wireshark.org/ дата обращения 22 апреля 2012
[13] Network Driver Interface Specification 5.1 http://msdn.microsoft.com/en-us/library/ff556916.aspx дата обращения 22 апреля 2012
[14] Trivial File Transfer Protocol (revision 2) http://www.ietf.org/rfc/rfc1350.txt дата обращения 22 апреля 2012
[15] DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION http://www.ietf.org/rfc/rfc1035.txt November 1987
[16] Domain Name System. Obsoleting IQUERY http://www.ietf.org/rfc/rfc3425.txt November 2002
[17] Portable Executable and Object File Format Specification http://download.microsoft.com/download/e/b/a/eba1050f-a31d-436b-9281-92cdfeae4b45/pecoff.doc дата обращения 22 апреля 2012
[18] Bitmap Storage http://msdn.microsoft.com/en-us/library/dd183391(VS.85).aspx дата обращения 22 апреля 2012
[19] J. Caballero, P. Poosankam, C. Kreibich, D. Song. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. // Proceedings of the 16th ACM conference on Computer and communications security (CCS '09), 2009. pp. 621—634
[20] Weiser M. Program slicing // Proceedings of the 5th International Conference on Software Engineering. — IEEE Computer Society Press, 1981. pp. 439—449
[21] G. A. Venkatesh: The Semantic Approach to Program Slicing. // Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI '91). pp. 26— 28.
[22] Z. Wang, X. Jiang , W. Cui, X. Wang, M. Grace. ReFormat: Automatic Reverse Engineering of Encrypted Messages. // Proceedings of the 14th European conference on Research in computer security (ESORICS'09). Springer-Verlag Berlin, Heidelberg, 2009. pp. 200—215
[23] J. Caballero. Grammar and Model Extraction for Security Applications using Dynamic Program Binary Analysis. / PhD thesis in Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, September 2010
[24] OSCAR "Open System for Communication in Realtime" http://iserverd1.khstu.ru/oscar/ Обновлено 07.02.2005