Восстановление формата данных

А.И. Гетьман, Ю.В. Маркин, В.А. Падарян, Е.И. Щетинин
{thorin, ustas, vartan, schet}@ispras.ru

Аннотация. Одной из распространенных практических задач анализа бинарного кода является восстановление структуры полученного программой сетевого сообщения или считанного файла. В случае работы аналитика с защищенным бинарным кодом трудоемкость восстановления формата данных становится недопустимо большой. В статье предлагается метод автоматизированного восстановления формата данных, базирующийся на динамическом анализе бинарного кода. Метод позволяет восстановить иерархическую структуру изучаемых данных и выявлять определенную семантику полей. Помимо того, представлены прототипная версия ПО, поддерживающего данный метод, и результаты работы данного ПО на модельном примере.

1. Введение

В настоящее время задача восстановления протоколов и файлов является весьма актуальной в сфере обратной инженерии и информационной безопасности. Решение этой задачи вручную является весьма затратным по времени и ресурсам, поэтому в настоящее время разрабатывается большое количество программных средств, позволяющих автоматизировать этот процесс. Восстановленная спецификация может использоваться при предотвращении несанкционированного доступа (intrusion prevention), для реализации технологии глубокого инспектирования пакетов (DPI), а также при тестировании приложения по методу «чёрного ящика» (black-box fuzzing) и сравнительном тестировании разных реализаций серверов.

Рассматривается задача восстановления формата сообщений используемых программой в рамках некоторого протокола. Под протоколом понимается набор правил, по которым осуществляется обмен сообщениями (байтовыми буферами известного размера) между двумя процессами. В частности, одним из участников обмена может являться файловая система – файл, в этом случае, рассматривается как совокупность сообщений. Основными компонентами протокола являются автомат состояний протокола и набор форматов сообщений, входящих в протокол. Задача восстановления автомата состояний в данной работе не рассматривается, так как её решение напрямую зависит от точности и полноты решения основной задачи – восстановление формата сообщений.

Формат сообщения, это совокупность знаний о полях сообщения, которые хранят базовые типы данных, а также о группировке полей в структуры и последовательности. Кроме того, в понятие формата входит информация о семантике полей – их роли в структуре сообщения. Примером семантически нагруженного поля может служить поле длины – оно хранит значение, равное размеру некоторого другого поля. От этого значения, следовательно, зависит длина сообщения.

Спецификация сообщения содержит, таким образом, следующие данные:

- Границы отдельных полей в сообщении.
- Группировка полей в структуры и последовательности.
- Семантика полей.
- Информация о значениях полей, входящих в сообщение.

Предлагаемое в данной работе решение включает в себя систему автоматического восстановления формата сообщений на основе динамического анализа трассы программы-разборщика, систему взаимодействия с пользователем, которая помимо основного результата – формата сообщения выдаёт данные, на основе которых пользователь может вручную уточнить полученный формат, а также систему хранения восстановленный форматов на основе базы данных MySQL.

Особенностью предлагаемого решения является отказ от полностью автоматического подхода в пользу автоматизированной системы анализа. Причиной такого выбора является то что, как показывает анализ предлагаемых программных решений, автоматический анализ в ряде случаев не может полностью решить проблему, а ручной анализ в этом случае весьма затруднён. В данной работе делается попытка с одной стороны использовать все преимущества автоматического анализа, а с другой предоставить пользователю гибкий инструмент для уточнения и коррекции, полученных в ходе автоматического анализа результатов.

Статья организована следующим образом. Во втором разделе рассмотрены известные подходы к решению задачи восстановления формата данных, проанализированы их слабые и сильные стороны. Третий раздел описывает схему работы предлагаемого метода восстановления формата данных. Далее более подробно рассматриваются отдельные аспекты этого метода. В четвертом разделе показано как входной буфер разделяется на поля примитивных типов данных, которые затем группируются в иерархическую структуру. Пятый раздел посвящен выявлению семантики отдельных полей. В шестом разделе рассматриваются способы обобщения восстановленного формата, если аналитик располагает несколькими экземплярами полученных сообщений. Практические результаты работы предложенного метода и заключение даны в седьмом разделе.

2. Обзор работ

Существует несколько различных подходов к восстановлению формата сообщений. Основными являются:

1. Анализ потока данных (например, сетевого трафика) с выделение повторяющихся паттернов. Основной недостаток – сложность определения семантики полей. Этот подход рассматривается в работе [1]
2. Статический анализ кода одного из участников обмена. Основной недостаток – наличие защиты от отладки и применение обфускации в современных программных средствах. Этот подход рассматривается в работе [2]
3. Динамический анализ кода одного из участников обмена. Основной недостаток – для анализа доступна только выполнявшаяся часть кода, вследствие чего формат сообщения не всегда может быть восстановлен полностью. Этот подход рассматривается в работах [3,4,5,6].

Для реализации был выбран динамический подход, так как в случае защищённых приложений он способен дать наиболее точные результаты.

Восстановление формата сообщений состоит в восстановлении двух типов информации: структурной и дополнительной. Структурная информация состоит из двух типов данных – границ полей и семантической информации, которая необходима для правильного разбора сообщения: список полей, задающих длину (length fields), поля-разделители (delimiter fields), поля-указатели (pointer fields), поля содержащие «магические константы» протокола и ключевые значения (key fields). Кроме того, к структурной информации относятся вариации сообщений одного типа, например наличие или отсутствие некоторого «плавающего» (floating) поля и иерархическая структура сообщения (вложенность одних полей в другие). Дополнительная информация включает в себя знания о содержимом полей: ключевые слова, имена файлов, идентификаторы сессии. В рассматриваемых ниже работах предлагаются некоторые методики для восстановления как структурной, так и дополнительной информации.

В работе [2] предлагается методика восстановления формата данных генерируемых некоторой программой, с помощью статического анализа этой программы. Для работы анализа от аналитика требуется специфицировать функции вывода, использующиеся программой. На основании этих данных строится межпроцедурный граф потока управления, который затем проецируется на вызовы функций, специфицированных пользователем. На основании этого графа генерируется формат в виде регулярного выражения. Для группировки полей в структуры используется алгоритм ASI[7]. Основным недостатком этой работы является игнорирование семантической информации, такой как поля длины и разделители. Кроме того, требование

ручной спецификации функций вывода сильно ограничивает применение данного подхода.

В работах [3,4,5,6] используется динамический анализ трасс, основанный на dynamic taint analysis [8], позволяющий выделить подтрассу обработки сообщения. В работе [3] описывается инструмент Polyglot, в котором впервые реализован ряд методик анализа семантики полей, таких как поле длины и разделитель, которые, с незначительными изменениями используются в последующих работах. Областью применения данного инструмента следует считать преимущественно текстовые протоколы (HTTP). Это накладывает определённый отпечаток на реализованные алгоритмы. В частности, большинство эвристик, применяемых в инструменте, используют статистические данные, полученные на основе использования для указанного класса задач. Работа [4] снимает ограничение на область применимости – рассматриваются как текстовые, так и бинарные протоколы, однако перед началом анализа необходимо указать, к какой именно группе принадлежит анализируемый протокол. Кроме того, восстанавливается не только «плоский» формат сообщения (последовательность полей), но также иерархическая структура и зависимости между полями. Взаиморасположение двух элементов в структуре могут быть 3-х типов: последовательность, вложенность, параллельность. Пример формата приведён на рис. 1.

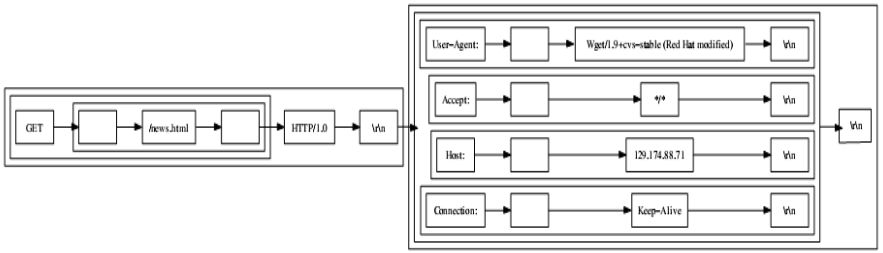


Рис.1. Пример структуры формата.

Применение подхода ограничено необфусцированными программами. Эвристика выделения параллельных полей основана на статистике по некоторому набору протоколов, что также ограничивает её применимость. Помимо прочего, в инструментах [3,4] отсутствует возможность уточнения формата на основе анализа программы на разных входных данных. Две различные методики, предоставляющих возможность обобщения формата сообщения на основе анализа нескольких трасс описываются в статьях [5,6]. Для обобщения формата требуется предварительно сопоставить элементы разных структур. Для решения этой задачи в работе [6] применяется анализ контекстов обработки этих элементов (наборов инструкций (их адресов), которые их обрабатывают). Преимуществом этого подхода является его

сравнительно высокая точность. Однако этот подход неприменим, во-первых, к обобщению на основании анализа разных реализаций обработчика (в том числе разных версий), во-вторых, при наличии навесной защиты сопоставление инструкций на основании адресов может быть некорректным. Методика, предложенная в работе [5], не имеет этих недостатков. Она основана на применении модифицированного алгоритма Нидлмана-Вунша для сравнения иерархических структур данных (Hierarchical Needleman-Wunsch).

В работе [6] описывается программный комплекс Tupni. Показывается эффективность этого инструмента при восстановлении формата на широком классе протоколов и файлов. В статье предложена оригинальная методика выделения последовательностей и структур в сообщении на основе анализа циклов в графе потока управления, которая в уточнённом виде использовалась при реализации описываемой системы. Восстановление семантики полей длины и разделителей осуществляется на основании анализа терминальных условий циклов. Среди ограничений подхода можно указать байтовую гранулярность при анализе данных – это не позволяет указывать поля, содержащие флаги. Помимо этого, для анализа циклов используется граф потока управления, полученный в ходе предварительного статического анализа программы. Такой анализ для защищённых приложений возможен далеко не всегда.

3. Описание системы восстановления формата

Решение поставленной задачи предполагает реализацию автоматизированной системы анализа, основными компонентами которой являются:

1. Набор алгоритмов анализа формата
2. Системы получения и отображения информации, на основе которой аналитик может оценить и проверить точность полученных результатов и принять решение о необходимости их ручной корректировки

Особенностью реализуемой системы, является предоставление пользователю как основных результатов анализа – спецификации формата сообщения, так и дополнительных данных, которые могут облегчить пользователю ручную коррекцию полученной спецификации. В качестве дополнительных данных выдаётся привязка выделенных элементов структуры к блокам шагов трассы, участвующим в их обработке. Кроме того, выдаётся полная подтрасса обработки сообщения, по которой может быть построен граф зависимостей, отражающий процесс преобразования каждого поля. Эта подтрасса также может служить входными данными для дальнейшего анализа, например, в случае зашифрованного протокола по этой трассе можно вычислить буферы, куда помещены расшифрованные данные, и рекурсивно повторить процедуру восстановления для этих буферов. Схема работы системы представлена на рис. 2.

Дополнительным источником информации о структуре сообщения могут быть спецификации некоторых функций, участвующих в обработке сообщения. Например, если некоторое поле передаётся как параметр размера в функцию malloc, то оно, вероятно, является размером некоторой последовательности. Другие примеры использования спецификаций есть в работах [2,6]. Предоставление такой информации, также отражено на схеме пунктиром.

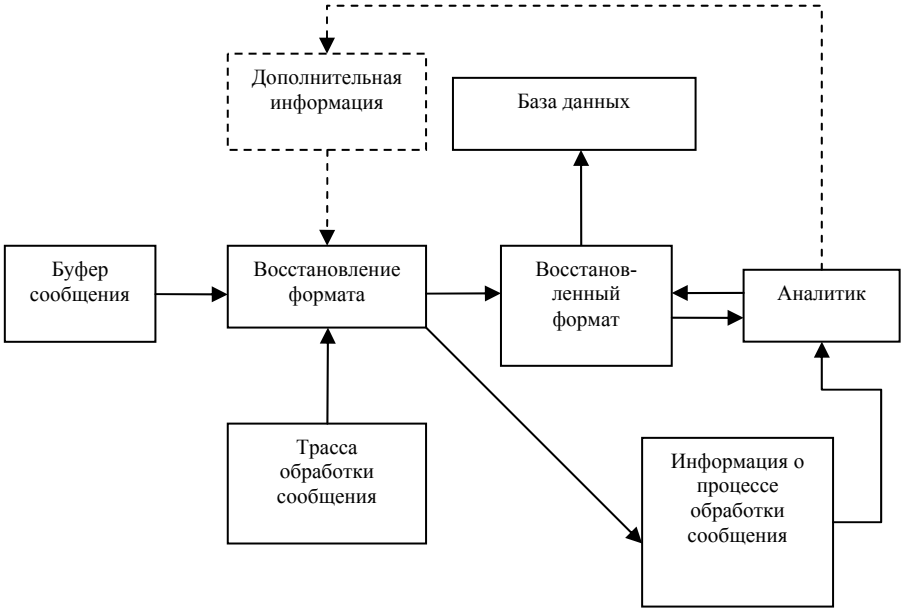


Рис. 2. Упрощённая схема работы системы восстановления..

3.1. Представление формата данных

Опишем сначала структуру формата, восстанавливаемую в ходе анализа.

Формат представляется в виде дерева, корневой вершиной которого является само сообщение, листовые вершины – поля, соответствующие базовым типам данных, а внутренние вершины подразделяются на несколько типов, определяющих группировку полей. Один из типов группировки – последовательность, аналог массива языка C, отличие заключается в том, что элементы последовательности могут иметь разную структуру и размер, вследствие чего доступ к ним осуществляется последовательно, а не в произвольном порядке. Кроме того, длина последовательности может быть неизвестна на момент начала её обработки, если она определяется полем разделителем (как в случае ограниченной нулем строки). Другой тип группировки – структура, аналог `struct` языка C. Поля, объединённые в структуру, логически связаны между собой и обрабатываются

локализованным кодом. Варианты, представляющие собой аналог `union` языка C, введены для группировки элементов последовательности, которые имеют разную структуру. Это сделано для того чтобы отразить факт, неоднородности последовательности. Всем вершинам сопоставлен размер соответствующих структурных единиц и их смещение относительно родительской вершины.

Семантическая информация хранится в виде атрибутов-меток, которые отражают некоторые функциональные свойства полей: разделитель, длина, указатель, функция, ключ, флаг. Пометка разделитель определяет поле, значение которого интерпретируется как конец последовательности. Пометка длина обозначает поле, определяющее количество идущих подряд полей или структур, составляющих последовательность. Пометка указатель обозначает поле, с помощью которого вычисляется указатель на некоторое другое поле сообщения. Пометка функция обозначает поле, имеющее произвольную функциональную зависимость от значений других полей, например, в случае вычисления контрольной суммы. Пометка ключ обозначает поле, в зависимости от значения которого определяет способ разбора некоторой части сообщения. Поля флаги интерпретируются, как совокупность отдельных битов, значение которых определяет поведение разборщика. Ещё один атрибут поля – признак виртуальности, этот атрибут означает, что последовательность байт, соответствующая этому полю, не обрабатывалась в анализируемой трассе программы. Следовательно, об этой последовательности байт на основании анализа текущей трассы ничего сказать нельзя.

3.2. Структура модуля восстановления формата

Перечислим основные компоненты анализа и опишем связи между ними.

1. Компонент выделения циклов в графе потока управления
2. Алгоритм проектирования выделенных циклов на трассу
3. Алгоритм выделения подтрассы обработки буфера сообщения
4. Алгоритм поиска границ полей сообщения
5. Компонент выделения последовательностей
6. Компонент выделения структур
7. Компонент построения дерева формата
8. Компонент обобщения формата
9. Компонент анализа семантики, который включает в себя следующие части:
    - поиск полей длины выделенных последовательностей
    - поиск полей-разделителей выделенных последовательностей
    - поиск полей-указателей
    - поиск полей, содержащих ключевые значения
    - поиск полей-флагов

Общая структура анализа может быть представлена в виде схемы на рис. 3.

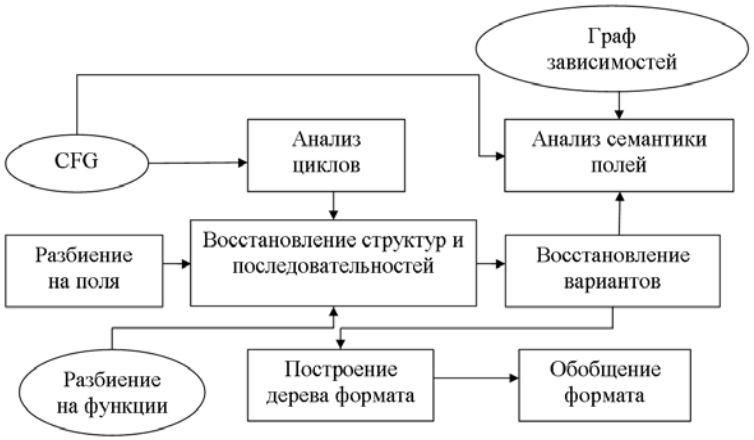


Рис. 3. Общая структура анализа..

Эллиптические вершины на рисунке обозначают данные, которые необходимы алгоритму для дальнейшего анализа. Прямоугольные вершины обозначают работу алгоритмов.

4. Анализ структуры сообщения

Источником информации о структуре сообщения служит трасса выполнявшихся инструкций процесса разбора этого сообщения. Для начала анализа требуется определить точку в трассе, в которой буфер сообщения уже заполнен, но разбор ещё не осуществлялся. Затем, методом прямого слайсинга, из трассы выделяется алгоритм обработки буфера. На основании размеров операндов инструкций доступа к буферу сообщения восстанавливаются размеры отдельных полей. По трассе частично восстанавливается статическое представление программы – граф потока управления (далее CFG). Следует отметить, что в данном случае его неполнота не сказывается на результатах восстановления. Анализ циклов, выделенных из CFG и спроецированных на трассу, позволяет сгруппировать поля в структуры и последовательности. Дополнительный семантический анализ условий выхода из циклов позволяет восстановить семантику отдельных полей. Полученная таким образом информация представляется в виде дерева формата. Затем производится обобщение этого дерева на случай произвольного сообщения данного типа. Это необходимо, для того чтобы можно было объединять структурную информацию, полученную при анализе разных сообщений одного типа. Основа методик, использующихся при анализе структуры сообщения, взяты из работы [6]. Они были усовершенствованы, что позволило с одной стороны получать больше информации на тех же примерах (флаговые поля), а с другой расширило

область применения методик (защищённый код, последовательности с полями указателями).

4.1. Восстановление полей

Под восстановлением полей имеется в виду задача выделения из последовательности байт сообщения таких непрерывных подпоследовательностей, каждая из которых интерпретируется обрабатывающим кодом как одно число.

Рассматриваются инструкции, обрабатывающие данные лежащие в буфере. При этом инструкции копирования не учитываются, так как они не осуществляют обработку данных. На основе размеров операндов (в битах) этих инструкций соответствующих данным буфера производится выделение полей. Последовательности байт буфера, к которым доступ не осуществлялся, помечаются как виртуальные поля.

Проблему представляет неоднозначная интерпретация сообщения в обрабатывающем его коде. Например, в основном цикле обработки сообщения, его поля могут восприниматься, как 4х байтовые целые, а в процессе подсчёта контрольной суммы, всё сообщение может восприниматься как массив однобайтовых величин. Для решения этой проблемы применяется эвристика, основанная на сопоставлении каждому выделенному полю веса, равного количеству инструкций, которые к нему обращались. В этом случае задача выделения полей сводится к определению покрытия буфера полями с максимальным весом. Вследствие высокой сложности алгоритма поиска оптимального решения применяется жадный алгоритм, который показал хорошие результаты в процессе применения к реальным задачам. Результатом работы алгоритма является разбиение буфера сообщений на поля, соответствующие базовым типам данных сообщения.

4.2. Восстановление последовательностей записей

В рамках данной задачи рассматривается взаимное положение разных полей. Вводится эвристическое предположение, что поля, обрабатывающиеся в цикле, образуют последовательность, причём на каждой итерации цикла обрабатывается одна запись – группа полей. Другой возможностью обработки последовательностей являются рекурсивные вызовы, однако ресурсоёмкость такого способа гораздо выше, а скорость ниже. На практике такой способ не встречался. Для каждой последовательности можно определить способ задания её длины. По аналогии с работой [6] выделяются следующие возможности:

- длина фиксирована протоколом
- длина явно задаётся значением поля (поле длины)
- длина неявно задаётся полем разделителем.

Алгоритм, определяющий способ задания последовательности является частью анализа семантики.

Примером последовательности записей может служить 0-терминированная строка, считывающаяся в цикле.

Для начала работы алгоритма требуется информация о статической структуре программы в виде графа потока управления. В данной реализации CFG предварительно строится на основе информации из трассы – не использую статический анализ бинарного кода. Это связано с тем, что в современных программах применяется большое количество средств защиты от анализа. К ним можно отнести применение программ-загрузчиков и шифрация кода. В результате применение статического анализа может быть значительно затруднено. Построение CFG на основе динамической информации из трассы позволяет преодолеть эти проблемы. Следует отметить, что, так как в трассе, как правило, реализуются далеко не все пути программы, то полученный CFG будет неполным – в нём будет отсутствовать информация о нереализованных ветвях исполнения. Однако его неполнота не сказывается на результатах восстановления. Поясним сказанное.

В рамках задачи восстановления последовательностей требуется найти циклы в CFG. Поиск циклов осуществляется с помощью алгоритма из книги [9]. Этот алгоритм основан на поиске обратных рёбер – для каждого обратного ребра может быть выделен цикл, которому оно соответствует. Если тело цикла было выполнено более одного раза, то обратное ребро попадёт в трассу и будет присутствовать в восстановленном CFG. Для случая вырожденного цикла, когда тело выполнялось всего один раз - обратное ребро в трассе отсутствует. Для того чтобы восстановить цикл в этом случае, применяется методика анализа условных переходов, которая на основании вычисления целевого адреса перехода позволяет восстановить ребро графа потока управления, если базовый блок, на который указывает переход присутствует в графе. Для циклов последнее условие всегда выполнено, поэтому все исполнявшиеся циклы будут присутствовать в восстановленном CFG.

Каждый выделенный на 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 каждой итерации строится множество релевантных инструкций. В него добавляются инструкции, которые осуществляют доступ к некоторому полю только на этой итерации – это позволяет отделить от полей последовательности поле длины, обращение к значению которого осуществляется более чем на одной итерации. Если то множество не пусто на каждой итерации, быть может, кроме последней, то такой цикл назовём итерационно-зависимым. Поля, обрабатывающиеся в релевантных инструкциях группируются в последовательность если:

- Поля образуют непрерывную байтовую последовательность на каждой итерации, если их дополнить виртуальными полями.

- Поля обрабатываются в прямом порядке, то есть адреса всех полей обрабатывающихся на итерации n меньше чем адреса полей, обрабатывающихся на всех итерациях k, где k > n.

Дополнительные сложности при выделении возникают при наличии внутри последовательности полей-указателей. При этом часто возникает ситуация, когда поля, доступ к которым осуществляется по этим указателям, также обрабатываются в цикле. В этом случае нарушается условие непрерывности обработки.

Для того чтобы восстанавливать последовательности и таких случаях была разработана дополнительная методика анализа доступа по полям-указателям. По результатам анализа – все инструкции доступа разделяются на две группы – доступ по полю указателю и доступ без указателя. Эти группы при анализе циклов обрабатываются отдельно, что позволяет восстанавливать последовательности точнее.

4.3. Восстановление структур

Под восстановлением структур понимается задача группировки последовательно расположенных полей, которые логически связаны друг с другом и обрабатываются кодом, относящимся к одному блоку шагов трассы. В качестве блоков шагов может выступать трасса отдельного вызова функции или трасса одной итерации некоторого цикла. В текущей реализации, в качестве блоков шагов рассматриваются только отдельные итерации циклов, так как анализ проведённый в статье [4] показывает, что применение вызовов функций для группировки полей ограничено текстовыми протоколами типа HTTP.

При восстановлении записей может применяться информация, полученная от пользователя – определения того, что пользователь считает блоками шагов. Для текстовых протоколов, пользователь может передать в частности границы вызовов функций. Это позволит уточнить полученные результаты.

Группируемые поля должны обладать свойством непрерывности – с учётом виртуальных полей они должны образовывать непрерывную последовательность байт.

Для того чтобы определить границы структур обрабатывающихся на разных итерациях цикла применяется следующая методика. Среди релевантных инструкций на первой итерации цикла выбираются те, которые осуществляют доступ по минимальному смещению. Это смещение считается началом первой структуры. Кроме того, предполагается, что эти же инструкции обращаются к началу каждой следующей структуры на последующих итерациях. Конец структуры на итерации n определяется началом структуры на итерации n + 1. Конец последней структуры определяется как максимальное смещение, по которому есть обращение на последней итерации.

Структуры могут быть вложенными друг в друга, то есть одна структура может являться частью другой. Эта ситуация может возникать, например, при

вложенных вызовах (внешний вызов обрабатывает охватывающую структуру, а вложенный - вложенную) или вложенных циклах. Таким образом, в процессе восстановления структур проявляется иерархическая структура сообщения.

5. Анализ семантики полей

Задача выявления семантики полей подразумевает определение роли некоторых полей (их значений) в формировании структуры сообщения. Рассмотрим подробнее предлагаемые алгоритмы.

5.1. Поиск полей длины

Значение поля длины тем или иным способом задаёт размер последовательности, которому оно соответствует. Оно может содержать, как размер последовательности в байтах, так и количество элементов в последовательности. В любом случае, это значение должно использоваться в условии выхода из цикла (которое проверяется на каждой итерации), при этом на каждой итерации это условие должно не выполнятся, а на последней должно быть выполнено. Именно на этих свойствах поля длины основан алгоритм его поиска. На первом шаге определяется поле, значение которого использовалось в условии выхода на последней итерации. Далее, на каждой итерации, проверяется, что в условии выхода используется значение этого поля и, при этом условие не выполняется ни на одной итерации.

5.2. Поиск полей-разделителей

Поле разделитель, это поле, которое следует сразу после некоторой последовательности, а его значение интерпретируется обработчиком, как терминальный символ, завершающий последовательность. По определению, его значение – это некоторая константа или набор констант (как в случае HTTP), значение которых известно обработчику. Для того чтобы определить конец последовательности обработчик сравнивает каждый элемент последовательности с терминальным символом (константой). Это используется в алгоритме поиска полей-разделителей.

Для цикла осуществляется поиск всех сравнений с константами, среди релевантных инструкций. Среди этих сравнений оставляют только те, которые выполнены на последней итерации и не выполнены на всех предыдущих. Поля, к которым осуществляется доступ в оставшихся инструкциях на последней итерации, объявляются разделителями. В текстовых протоколах таких полей может быть несколько (в HTTP как разделитель используется пара символов “\r\n”), в бинарных поле, как правило, одно.

5.3. Поиск полей-указателей

Основное свойство таких полей – использование в коде их значений для формирования указателя на некоторое другое поле сообщения. Их выявление основано на анализе способа формирования адреса для доступа к полям. В

случае если адрес формируется на основе значения некоторого поля, поле добавляется во множество указателей. Приведём пример:

```
MOV AX, dword ptr [BX + CX]
```

Если ячейка памяти является полем сообщения A, а значение BX или CX зависит от значения некоторого другого поля сообщения B, то поле B является полем-указателем. Стоит отметить, что в отдельных случаях значения полей длины также могут использоваться подобным образом, поэтому из множества полей-указателей должны быть удалены поля длины.

5.4. Поиск ключевых полей

Под ключевыми полями имеются в виду поля, которые могут содержать заранее известный для данного протокола набор значений. В процессе анализа сообщения, разборщик проверяет значения таких полей, сравнивая их с этим набором и, в зависимости от результата сравнений, осуществляет дальнейший разбор. Таким образом, значение данного поля определяет конкретный вид и интерпретацию некоторой части сообщения. Примером такого поля является поле «Тип ресурсной записи» протокола DNS, значение которого определяет, что содержит данная ресурсная запись.

Поиск ключевых полей определяется способом их обработки – требуется найти поля, значения которых сравнивается с некоторым набором констант, причём одно из сравнений истинно, и по результатам сравнения осуществляется условный переход в обработчике. Шаблон поиска:

```
CMP field, const
ConditionalJump target
```

Константы, фигурирующие во втором операнде, при анализе интерпретируются как ключевые значения протокола. Однако только константа, сравнение с которой дало истину действительно является ключевым сообщением. Другие же сравнения подходящие под шаблон могут являться как перебором разборщиком ключевых значений, так и результатом применения обфускации, вставляющей мёртвый код (сравнения, которые заведомо не выполнятся). В результате требуется принять решение – применять консервативный подход и учитывать только константы истинных сравнений или оптимистичный, предполагающий отсутствие такого рода обфускации, сохраняющий все константы. В данной работе принят промежуточный подход – сохраняются все константы, но при этом они разбиты на две группы: встречающиеся в истинных сравнениях и остальные. При этом константы первой группы точно являются ключевыми значениями протокола, а константы второй группы являются таковыми при условии отсутствия обфускации.

Следует отметить, что этому шаблону также соответствуют поля-разделители. Поэтому из множества найденных ключевых полей требуется удалить поля разделители.

5.5. Поиск полей флагов

Битовая гранулярность работы с памятью даёт возможность анализировать работу с отдельными битами сообщения. В архитектуре x86 существует набор операций (BT, BTC, BTR, BTS), позволяющий оперировать отельными битами регистров и ячеек памяти. Однако на практике при анализе флагов в полях сообщения чаще используется другой шаблон работы – применение логического «И» к значению некоторого поля, причём в качестве второго операнда выступает константа, значение которой определяет, какой бит (или группа битов) интересует программу. В архитектуре x86 такая операция представлена двумя инструкциями – AND и TEST, первая из которых разрушает исходное значение ячейки, а вторая только устанавливает флаги по результатам применения логической операции. По результатам проверки битового поля осуществляется условный переход. Соответственно, поиск битовых полей осуществляется по двум шаблонам –

```
BT field, bitNum
ConditionalJump target
```

и

```
“AND” field, const
ConditionalJump target
```

Границы битового поля в обоих шаблонах определяются вторым операндом первой инструкции.

5.6. Особенности применения шаблонов

Дополнительной трудностью при применении шаблонного анализа является наличие обфускации и оптимизированного кода. В результате между инструкциями сравнения (первой инструкцией шаблонов) и условным переходом (вторая инструкция шаблона) может присутствовать мёртвый код (результат применения обфускации) или инструкции, которые можно выполнить параллельно с инструкцией сравнения (результат оптимизаций). Это приводит к тому, что указанные шаблоны лучше применять не на последовательном потоке инструкций, а непосредственно на графе зависимостей, в котором между инструкцией сравнения и инструкцией условного перехода будет присутствовать зависимость по данным. Таким образом, шаблоны преобразуются к тривиальному виду, показанному на рис. 4.

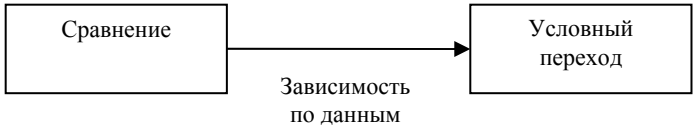


Рис. 4. Шаблон поиска на графе зависимостей.

6. Дерево формата

На основании данных о границах полей и их группировке в структуры и последовательности строится дерева формата. Вершинами дерева являются: само сообщение, последовательности, структуры, поля. Элемент A соответствующий диапазону адресов [x1, y1] сообщения, является предком элемента B [x2, y2] тогда и только тогда, когда x1 <= x2 и y1 >= y2.

6.1. Обобщение дерева формата

Одной из целей автоматического восстановления формата сообщения является автоматическая генерация парсера для всех сообщений данного типа. Следствием применения динамического анализа является то, что для анализа доступна только часть программы, участвующая в обработке некоторого конкретного сообщения. Для преодоления этого ограничения производится обобщение формата конкретного сообщения, таким образом, чтобы результирующий формат соответствовал более широкому классу сообщений данного типа. В качестве примера рассмотрим следующую возможность:

Пусть разобранное текстовое сообщение представляет собой ограниченную нулем строку ASCII-символов «Hello world!» длиной 12.

Восстановленный формат будет соответствовать классу сообщений, изображённому слева на рис. 5. Если предположить, что все сообщения данного типа содержат ровно 12 символов, то поле-разделитель является избыточным. Так как сетевые сообщения, как правило, не содержат избыточности, то это приводит к выводу, что на самом деле текстовое сообщение может быть любой длины. Именно такой формат и будет результатом применения операции обобщения к восстановленному формату – он показан на рис.5 справа. Такой формат позволит сгенерировать парсер для более широкого класса сообщений, чем исходный.

Обобщение включает в себя три основные операции:

1. Определение элементов, длина которых может быть неопределённой
2. Распространение неопределённого смещения и размера по дереву формата
3. Определение неоднородных последовательностей (последовательностей, элементы которых имеют разную структуру) и группировка таких полей с помощью элементов типа вариант

Неопределённую длину могут иметь два типа элементов – последовательности, длина которых задаётся с помощью поля разделителя или поля длины и варианты, если их дети имеют разную длину. Алгоритм, определяющий является ли некоторая последовательность однородной, а также описание типа вариант будет описано в следующем разделе. Далее приводится алгоритм распространения неопределённости в смещении и размере по дереву формата.

1. Пусть U – множество элементов, длина которых неопределенна.

2. Берём следующий элемент u из множества U, элемент p = parent(u) – его предок в дереве формата, если его размер имеет определённое значение, то делаем его неопределённым и добавляем p в множество U.
3. Для p, вычисляем множество childs(p), такое что, для каждого его элемента c выполнено c.offset > u.offset. Для всех элементов множества childs(p) выставляем неизвестное смещение.
4. Удаляем u из множества U.
5. Переходим на шаг 2.

На рис. 5 можно видеть результат применения операции 1 (элемент с неопределённой длиной – последовательность A) и 2 (смещение поля B в обобщённом формате не определено).

На рис. 6 приведён пример применения операции 3 к другому типу исходного сообщения, в котором фигурирует последовательность, чётные элементы которой имеют длину 1, а нечётные – длину 2. Так как информация о том, какие именно элементы имеют тот или иной размер анализатору недоступна, то алгоритм консервативно предполагает, что элемент последовательности с любым номером может иметь как размер 1, так и размер 2. При этом, так как не найден способ задания поля длины, предполагается что длина последовательности фиксирована и равна 12. В этих условиях, сколько именно полей будет содержать последовательность в конкретном сообщении, неизвестно – она может содержать от 6 (все поля длины 2), до 12 (все поля длины 1) элементов.

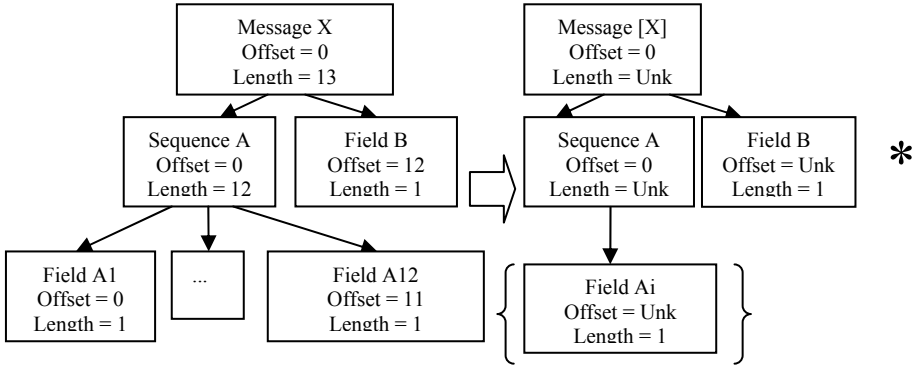


Рис. 5. Пример №1 обобщения формата.

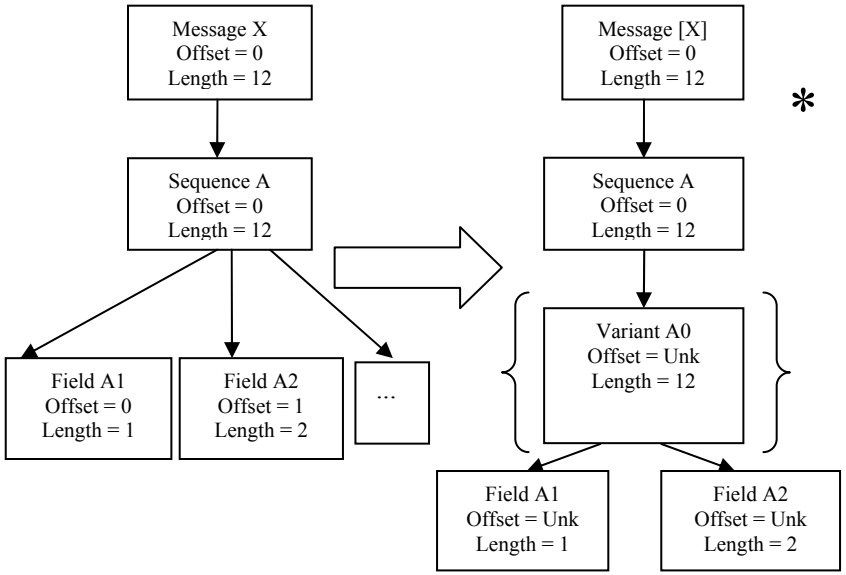


Рис. 6. Пример №2 обобщения формата.

6.2. Восстановление вариантов

Задача восстановления вариантов включает в себя анализ элементов некоторой последовательности с целью определения того, являются ли они объектами одного и того же класса или нет, а также генерацию элемента дерева типа вариант, дети которого соответствуют представителям всех обнаруженных классов. Элементу последовательности сопоставляется класс на основании следующих его характеристик:

- тип (поле, структура, последовательность);
- размер;
- семантика (в случае полей);
- сигнатура.

Сигнатура сопоставляется элементам структуры на основании особенностей их обработки в трассе. Алгоритм создания сигнатуры:

1. Если элемент – поле, то сигнатура это адрес инструкции первой обработки этого поля.
2. Если элемент – последовательность, то сигнатура это адрес инструкции входа в цикл обработки этой последовательности. Такой способ задания сигнатуры для последовательности позволяет выявить эквивалентность между последовательностями разной длины, и разного состава

элементов, которые, тем не менее, обрабатываются одним и тем же кодом.

3. Если поле – структура, то сигнатура это объединение сигнатур, полученных рекурсивным применением правил 1 и 2, для элементов, образующих структуру (её потомков в дереве формата).

Для каждого выделенного класса создаётся его представитель. Если выявлен ровно один класс, то его представитель включается в дерево, как единственный ребёнок последовательности. В случае если количество классов больше 1, создаётся элемент типа вариант, который интерпретируется, как представитель всех элементов последовательности. Он включается в дерево как единственный ребёнок последовательности, а в качестве его детей добавляются представители выделенных классов.

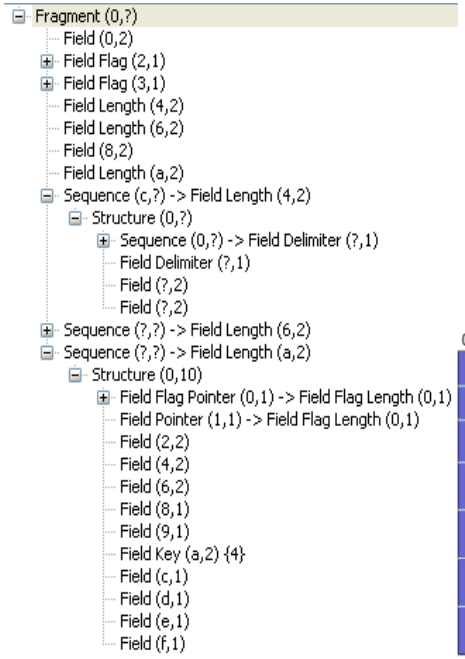
Результаты работы алгоритма восстановления вариантов показан на рис. 6.

7. Заключение

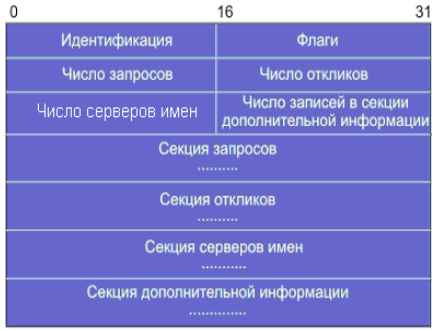
Предложенный в разделах 3-6 метод автоматизированного восстановления формата данных был поддержан прототипным программным средством, реализованным в рамках среды динамического анализа защищенного бинарного кода TrEx. Реализация представляет собой модуль-расширение с именем «ru.ispras.ctt.trex.algs.FormatRecovery»; модулю требуется для корректной работы наличие разметки в трассе вызовов функций и графы потока управления для каждой функции.

В качестве теста, для проверки точности восстановления был выбран протокол DNS. Его спецификация детально описана в документе RFC-1035. Протокол достаточно функционален и область его потенциального применения достаточно широка. При этом протокол содержит всего один тип сообщения, которое, однако, имеет сложный формат, содержащий все типы семантических полей – поля длины, разделители, указатели, флаги, ключевые поля. Тестировалась реализация разборщика DNS-сообщений в ОС Windows XP, nslookup. Для проверки работы алгоритма была снята трасса отправки запроса типа SOA (start of authority record — ресурсная запись DNS о сервере, хранящем эталонную информацию о домене) и получения соответствующего ответного сообщения. На рис. 7 слева приведена спецификация формата на основе RFC-1035, а справа спецификация, восстановленная системой. По результатам сравнения можно заметить следующие особенности – 2х байтовое поле флагов восстановилось как два отдельных флаговых поля – при проверке вручную, выяснилось, что данная реализация действительно обрабатывает эти 2 байта отдельно. Поле длины соответствующее числу серверов имён не восстановлено, так как при данном конкретном запросе секция серверов имён была пуста, и это поле не использовалось как поле длины. Таким образом, причина не полностью восстановленной семантики – принципиальное ограничение динамического подхода.

Для преодоления этого ограничения возможно использовать одну из методик объединения информации от анализа трасс разных сообщений. Две подобные методики рассмотрены в работах [5,6]. Для того чтобы объединить информацию о типах сообщений требуется сначала произвести операцию выравнивания, то есть сопоставить записи в одном сообщении соответствующим записям в другом, а затем произвести объединение форматов. Рис. 8 демонстрирует описанные операции. В работе [5] для реализации выравнивания используется модифицированный алгоритм Нидлмана-Вунша, впервые описанный в работе [10].



Спецификация DNS по RFC-1035



Фрагмент восстановленной спецификации

Рис. 7. Сравнение результатов восстановления формата DNS-сообщения с его спецификацией.

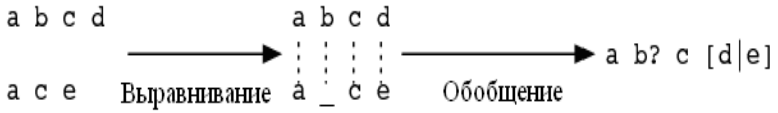


Рис. 8. Обобщение форматов двух сообщений. Буквами обозначены записи разных типов.

Ещё одна неточность восстановленной спецификации – не восстановленная последовательность байт содержащая IP-адрес сервера (последние 4 поля) и соответствующее ей поле длины (восстановленное как ключевое поле со значением 4). При ручном анализе соответствующего кода было выяснено, что он оптимизирован с применением алгоритма разворота цикла. Для преодоления этого типа оптимизаций требуется разработать соответствующий алгоритм шаблонного поиска развёрнутых циклов на графе потока управления.

Тем не менее, прототипная версия ПО уже используется на практике, что позволяет аналитику существенно сократить время, затрачиваемое на анализ соответствующего класса программ.

В дальнейшем предполагается продолжить работы по улучшению предложенной методики восстановления формата сообщений и развитию ее программной реализации.

Литература

[1] W. Cui, J. Kannan, H.J. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. // 16th USENIX Security Symposium, 2007. Pp. 199–212.

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

[3] J. Caballero, H. Yin, Z. Liang, D. Song. Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. // Proceedings of the 14th ACM conference on Computer and communications security, 2007. Pp. 317 – 329.

[4] Z. Lin, X. Jiang, D. Xu, X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. // Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[5] G. Wondracek, P. Milani Comparetti, C. Kruegel, E. Kirda. Automatic Network Protocol Analysis. // Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[6] 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.

[7] G. Ramalingam, J.Field, F. Tip. Aggregate Structure Identification and its Application to Program Analysis. // In Symp. on Principles of Programming Languages, 1999. Pp. 119 – 132.

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

[9] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы. Принципы, Технологии и Инструментарий. / Вильямс, 2008 г.

[10] 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):443-53.