Федеральное государственное бюджетное учреждение науки
Институт системного программирования РАН

на правах рукописи

Маркин Юрий Витальевич

Методы и средства углубленного анализа сетевого трафика

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

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
к. ф.-м. н. Падарян В.А.

Москва, 2017

Оглавление

Введение

Активное развитие информационных технологий сделало их неотъемлемой частью быта, производства, сферы услуг. Информационные системы сегодня эксплуатируются как в коммерческих, так и в государственных организациях. Взаимодействие между такими системами осуществляется посредством глобальной сети. Наблюдается стремительный рост объема сетевого трафика, усложняется его структура. Анализ трафика становится все более востребованным в областях контроля и управления, оптимизации, защиты от вредоносных воздействий.

Актуальными являются исследования в области обратной инженерии закрытых сетевых протоколов [1], используемых коммерческими приложениями. Исходные коды таких приложений не публикуются в открытом доступе: разработчик предоставляет конечному пользователю программу в виде набора исполняемых файлов. В таких случаях необходимо проведение комплексного аудита безопасности кода на предмет поиска ошибок и уязвимостей, в том числе в реализации используемого протокола. В случае сетевого приложения вместе с бинарным кодом анализируется сетевой трафик компьютера (сетевого интерфейса), на котором это приложение запущено.

Важной практической задачей является разработка методов защиты внутренних сетей предприятий. Один из методов состоит в накоплении базы знаний о структуре проводимых атак и характерных сигнатурах передаваемых данных. Для случаев, когда факт проведения сетевой атаки установлен, необходимо провести расследование: выяснить, как развивался инцидент, какие данные были повреждены или считаны, какие взаимодействия с другими компьютерами происходили. Детальный анализ подобных инцидентов в перспективе позволяет совершенствовать средства защиты (как программные, так и аппаратные) таким образом, чтобы блокировать определенные виды сетевых атак в режиме реального времени. Атаки, основанные на перекрытии IP-фрагментов [2, 3] или TCP-сегментов [4, 5], не могут быть обнаружены путем анализа содержимого сетевых

пакетов по отдельности. Для детектирования этих атак необходимо провести IP-дефрагментацию и восстановить TCP-потоки. Восстановление потоков данных, передаваемых между сетевыми приложениями, позволяет выявлять случаи распространения вредоносного ПО, а также запрещенного контента. Известно, что сетевые пакеты могут быть получены принимающей стороной в порядке, отличном от того, в котором они были отправлены [6]. Для восстановления потока данных в таких случаях должен быть восстановлен порядок отправки пакетов.

По статистике доля зашифрованного трафика существенно увеличилась за последние несколько лет: крупные сайты начали использовать протокол HTTPS по умолчанию [7, 8]. В связи с этим актуальной практической задачей является анализ трафика на предмет использования серверами уязвимых криптографических алгоритмов, а также недоверенных сертификатов безопасности.

Широкое распространение также получили туннельные протоколы: в частности, они используются для организации VPN [9]. Особенность туннельных протоколов состоит в том, что потенциально, возможно построение туннеля (т.е. стека используемых протоколов) произвольной конфигурации. Анализ трафика вложенных туннелей позволит судить о безопасности такого рода соединений. Отметим, что при организации сетевого туннеля в общем случае может использоваться произвольный стек протоколов. В условиях отсутствия информации о стеке протоколов разбор трафика туннельных соединений возможен при наличии функциональности по распознаванию протоколов, формирующих стек.

Спектр инструментов, применяемых для решения практических задач, связанных с анализом трафика, очень широк – при этом каждый использует собственные разборщики трафика и оперирует над своим внутренним представлением разобранных сетевых пакетов. Как правило, инструменты не используются по отдельности, а объединяются в рамках некоторого комплекса. Например, для обеспечения безопасности внутренней сети могут использоваться межсетевые

экраны [10], системы обнаружения и предотвращения вторжений [11], системы защиты от DDoS-атак [12] и др. Чтобы добавить поддержку некоторого протокола (т.е. возможность его анализировать) в организованный из отдельных инструментов комплекс, потребуется создать разборщик пакетов этого протокола для каждого анализатора в соответствии с его внутренним представлением. Объем необходимой работы существенно сократится, если все инструменты комплекса будут опираться на единое внутреннее представление.

Задачи обеспечения безопасности сети, а также контроля качества связи решаются в *online-режиме*, тогда как расследование инцидентов нарушения информационной безопасности, обратная инженерия и отладка протоколов в *offline-режиме*. Под online-анализом (или режимом) понимается анализ трафика сетевого интерфейса (или канала связи). Offline-анализ (или режим) предполагает анализ предварительно сохраненного в файл трафика. Для проведения online-анализа инструмент должен работать непрерывно с производительностью, достаточной для разбора поступающего трафика: требуется обеспечение возможности обработки потенциально бесконечного входного потока данных. В случае offline-анализа инструмент получает входные данные (конечного размера) из файла. Поэтому может проводиться более детальный анализ по сравнению с online-анализом на аналогичном трафике. Основная масса существующих инструментов и применяемых в них методов фактически предназначена для проведения анализа только в одном из режимов. В то же время, наличие универсального для двух режимов инструмента позволило бы упростить расширение функциональности. Применение в online-анализе разборщика, созданного в offline-режиме, фактически реализует методологию повторного использования кода (code reuse). С течением времени разборщики будут дорабатываться в offline-режиме и сразу же применяться для анализа в режиме online.

Таким образом, необходима унификация компонентов инструментов, отвечающих за проведение разбора сетевых пакетов. При этом одни и те же разборщики должны применяться при проведении анализа в online и offline режимах.

Целью диссертационной работы является исследование и разработка методов и программных средств углубленного анализа сетевого трафика, позволяющих автоматизировать расширение их функциональности. Методы должны обеспечивать устойчивость к потере отдельных сетевых пакетов и переупорядочиванию пакетов при передаче.

Для достижения поставленной в работе цели необходимо решить следующие **основные задачи**:

1. Разработать модель представления разобранных сетевых пакетов. Учесть в модели следующие особенности передачи данных по сети:
   - потеря/переупорядочивание отдельных пакетов;
   - сжатие и шифрование данных;
   - вложенное туннелирование.
2. Разработать алгоритм восстановления потоков данных для протоколов произвольного уровня, в том числе прикладного, устойчивый к потере отдельных сетевых пакетов, а также их переупорядочиванию.
3. Разработать архитектуру системы углубленного анализа сетевого трафика, позволяющую разрабатывать и отлаживать модули поддержки протоколов на предварительно сохраненном трафике и впоследствии использовать эти модули для разбора пакетов в режиме online.
4. Для получения экспериментальных оценок качества разбора сетевого трафика разработать и реализовать программные инструменты, базирующиеся на предложенных автором модели, алгоритме и архитектуре.

Научной новизной обладают следующие полученные результаты:

1. Модель представления данных позволяет единообразно описывать разбор заголовков произвольного стека сетевых протоколов.
2. Алгоритм восстановления потоков данных устойчив к переупорядочиванию и потере отдельных сетевых пакетов.

3. Архитектура системы позволяет использовать одни и те же разборщики заголовков сетевых протоколов при проведении анализа в online и offline режимах.

Теоретическая и практическая ценность работы. Разработаны модель представления данных и алгоритм восстановления потоков данных сетевых протоколов. Модель позволяет разделять и выполнять независимо фазы распознавания и разбора данных для произвольного стека сетевых протоколов.
Модель и алгоритм реализованы в виде динамической библиотеки и интерфейса для ее использования в составе программной системы анализа сетевого трафика. На базе предложенной модели разработаны и реализованы инструменты для проведения анализа сетевого трафика в online и offline режимах.
Инструмент для проведения анализа в online-режиме рассчитан на использование в сторонних системах. Инструмент для проведения offline-анализа используется при решении задач, связанных с обратной инженерией или отладкой сетевых протоколов, в области научных исследований и учебных курсах ВМК МГУ и ФУМП МФТИ.

Апробация работы. Результаты работы обсуждались на следующих конференциях:

- XXIV Общероссийская научно-техническая конференция "Методы и технические средства обеспечения безопасности информации", Санкт-Петербург, 29 июня – 2 июля 2015 г.
- 58 Научная конференция МФТИ, Москва–Долгопрудный–Жуковский, 23 – 28 ноября 2015 г.
- Научно-практическая Открытая конференция ИСП РАН, Москва, 1 – 2 декабря 2016 г.

Краткое содержание. Работа состоит из введения, пяти глав, заключения и списка литературы. Диссертация изложена на 80 страницах. Список литературы насчитывает 53 наименования. Диссертация содержит 10 таблиц и 31 рисунок.

В **главе 1** перечисляются особенности процесса передачи данных по сети с коммутацией пакетов, формулируются требования к разбору сетевого трафика, проводится обзор и сравнение существующих подходов.

В **главе 2** описывается модель представления данных, удовлетворяющая перечисленным требованиям. Вводятся определения контекста, ключевой группы, потока, понятие распознавателя. Приводится алгоритм восстановления потоков данных.

В **главе 3** представлена архитектура системы анализа, показано назначение отдельных компонентов.

В **главе 4** рассматриваются инструменты для проведения анализа трафика в online и offline режимах. Детально описываются возможности графического интерфейса, посредством которых представляются результаты разбора сетевых пакетов.

В **главе 5** приводятся примеры практического применения разработанных инструментов. Для offline-анализатора демонстрируется применение разработанного алгоритма восстановления потоков: восстанавливается TCP-поток, пакеты которого при передаче были переупорядочены. Дальнейший разбор этого потока позволяет извлечь HTML-страницу из данных HTTP-пакетов. Второй пример использования offline-инструмента демонстрирует возможность анализа зашифрованных данных при наличии информации, необходимой для проведения их расшифровки: записи протокола SSL (TLS) расшифровываются с помощью закрытого ключа SSL-сервера. Для online-анализатора показана возможность извлечения из трафика высокоуровневых данных – файлов в формате PNG. Наиболее важной демонстрацией возможностей разработанных инструментов является применение их совокупности для решения задачи обратной инженерии закрытого командного протокола ботнета rbot.

1. Обзор существующих сетевых анализаторов

Анализ сетевого трафика приобретает все большую актуальность в связи с развитием сетевых технологий, увеличением объема данных, передаваемых по сети, внедрением большого количества новых сетевых протоколов (в том числе закрытых). В качестве основных областей практического применения можно выделить следующие:

- выявление проблем в работе сети [13];
- тестирование (отладка) сетевых протоколов [14, 15];
- предотвращение сетевых атак [16];
- классификация трафика [17].

Здесь и далее рассматриваются сети с коммутацией пакетов. Решение практических задач анализа главным образом опирается на разбор заголовков сетевых протоколов в пакетах и восстановление потоков передаваемых данных. В соответствии с количеством разбираемых заголовков протоколов, принадлежащих разным уровням модели OSI [18], выделяют три основных класса проводимого разбора (рис. 1): поверхностный (SPI – Shallow Packet Inspection), средний (MPI – Medium Packet Inspection) и углубленный (DPI – Deep Packet Inspection) [19].

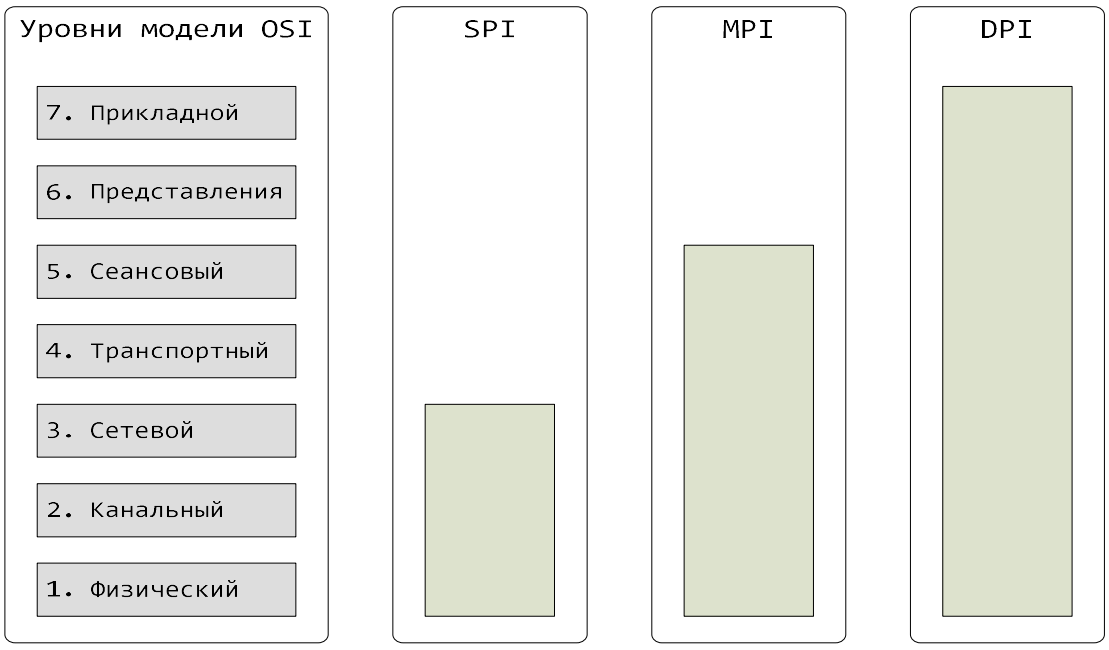

Рис. 1 – Классы проводимого разбора сетевых пакетов.

К анализаторам «поверхностного» уровня главным образом относятся простейшие межсетевые экраны: решение о блокировании того или иного пакета обычно принимается в соответствии со списком запрещенных IP-адресов и номеров портов.

Инструменты, относящиеся к «среднему» уровню, позволяют проводить фильтрацию трафика с использованием информации о формате передаваемых данных, а также более полной (по сравнению с отдельным IP-адресом) локализации отправителя. Как правило, такие инструменты выступают в роли посредника (application proxy) между провайдером доступа к Интернет и внутренней сетью.

Системы DPI прежде всего предназначены для идентификации приложений, участвующих в сетевых взаимодействиях. Поэтому «углубленный» разбор предполагает анализ содержимого сетевых пакетов на всех уровнях. Для более точной идентификации DPI-инструменты дополнительно могут использовать косвенные признаки, присущие определенным сетевым приложениям и/или протоколам. Для этого, в частности, применяются методы статистического анализа.

1.1 Особенности передачи сетевого трафика

Каждый сетевой пакет состоит из управляющей информации и полезной нагрузки. Здесь и далее термин «пакет» применяется в качестве универсального для обобщения таких понятий, как фрейм, дейтаграмма, сегмент соответствующих сетевых протоколов. В процессе разбора в пакете выделяются заголовки протоколов, анализируются значения полей в них. Структура заголовка определяется спецификацией, тогда как полезная нагрузка может содержать произвольным образом организованные данные, хотя обычно представляет собой пакет протокола следующего, более высокого уровня: для продолжения разбора необходимо определять, какой это протокол (рис. 2).

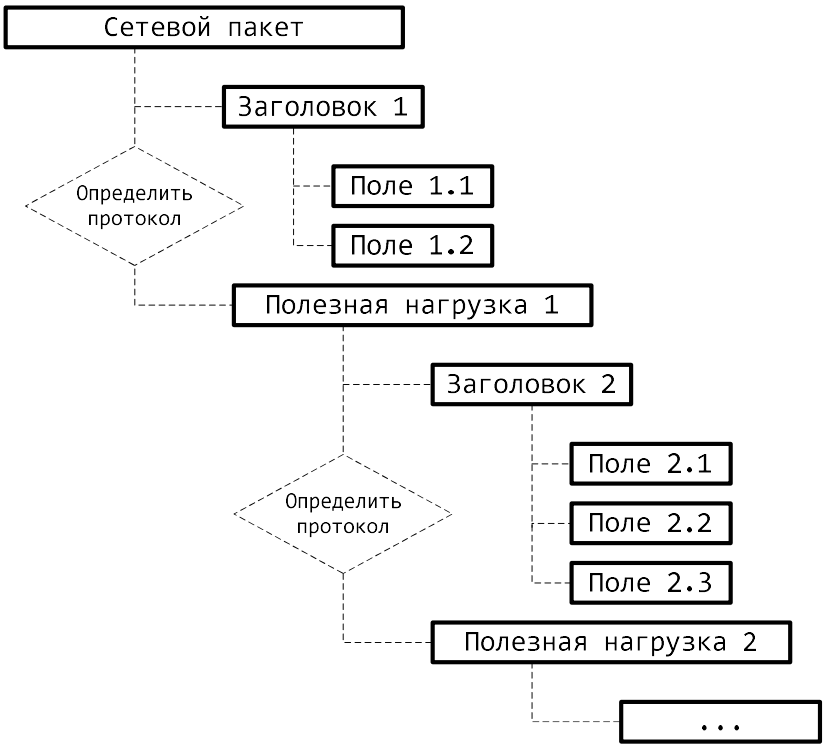

Рис. 2 – Выделение и разбор заголовков протоколов в пакете.

В соответствии с моделью OSI заголовки сетевых протоколов пакета образуют стек и, как правило, следуют друг за другом в естественном порядке – от низкого уровня к высокому. Однако при организации туннельных соединений этот порядок может быть нарушен – например, при передаче IPv4-пакетов (сетевой уровень) в рамках пакетов протокола UDP [20] (транспортный уровень). Туннельные протоколы в настоящее время получили широкое распространение: в частности, они используются при организации виртуальных частных сетей. В общем случае возможно построение туннеля произвольной конфигурации: в частности, один туннель может быть вложен в другой. Разбор туннельного трафика должен поддерживаться сетевым анализатором.

Выделяют протоколы с сохранением и без сохранения состояния. Обязательной частью спецификации протокола с сохранением состояния является соответствующий автомат состояний (Protocol State Machine). При проведении анализа в режиме реального времени количество соединений, для которых необходимо сохранение характеристик текущего состояния, может неограниченно

расти. Поэтому анализатор должен гибко управлять распределением доступных ему ресурсов.

Важной характеристикой сетевого протокола является MTU (Maximum Transmission Unit) – максимальный размер данных, которые могут быть переданы в рамках одного пакета. Для протокола IPv4 значение MTU составляет 65535. Поскольку на практике IPv4-пакеты обычно инкапсулируются в Ethernet-фреймы, результирующее значение MTU определяется в соответствии с конкретной версией стандарта Ethernet [21], поддерживаемого сетевым оборудованием. Для блоков данных с размером, превышающим MTU, проводится фрагментация: отправитель разбивает блок на порции допустимого размера, после чего каждая порция передается в рамках отдельного пакета. Получатель, таким образом, должен выполнить дефрагментацию: восстановить исходный блок из полученных по отдельности порций. Для протокола IPv4 последний фрагмент определяется сброшенным флагом MF (More Fragments): при этом в нем не содержатся (рис. 3) данные следующей PDU (Protocol Data Unit – единица передачи).

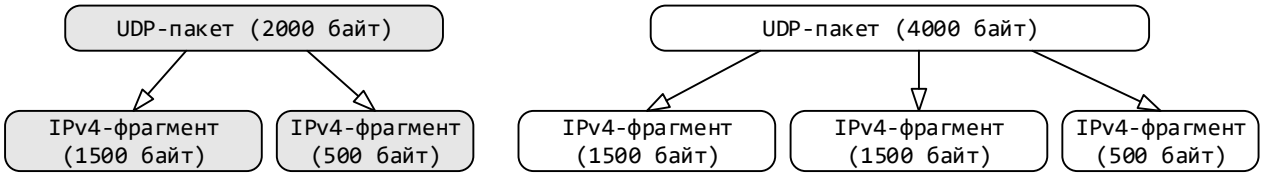

Рис. 3 – Пример фрагментации IPv4.

В случае протокола TCP (рис. 4) неформальным признаком «последнего» для заданной PDU сегмента являет PSH-флаг, однако этот сегмент в общем случае содержит данные следующей единицы передачи – возникает задача определения границ.

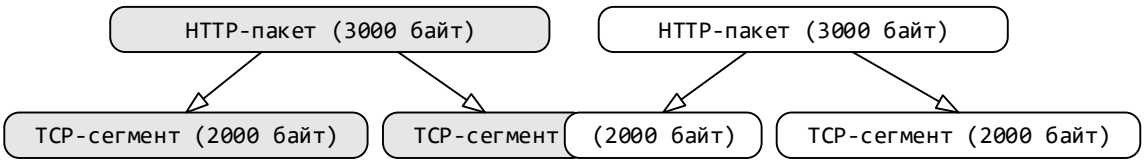

Рис. 4 – Пример сегментации TCP.

Для проведения глубокого анализа, таким образом, необходимо:

- восстанавливать исходный порядок следования пакетов;
- определять границы вышележащих PDU.

Вопросы, связанные с переупорядочиванием пакетов протоколов IP и TCP, в частности, рассматриваются в статьях [22, 23].

Для обеспечения безопасности соединений некоторые протоколы предполагают передачу данных в зашифрованном виде (например, семейство протоколов TLS [24, 25]). Чтобы проанализировать зашифрованные данные, необходимо их предварительно расшифровать, использовав предоставленный пользователем ключ (рис. 5): анализатор должен обладать интерфейсом для добавления недостающей для проведения разбора информации.

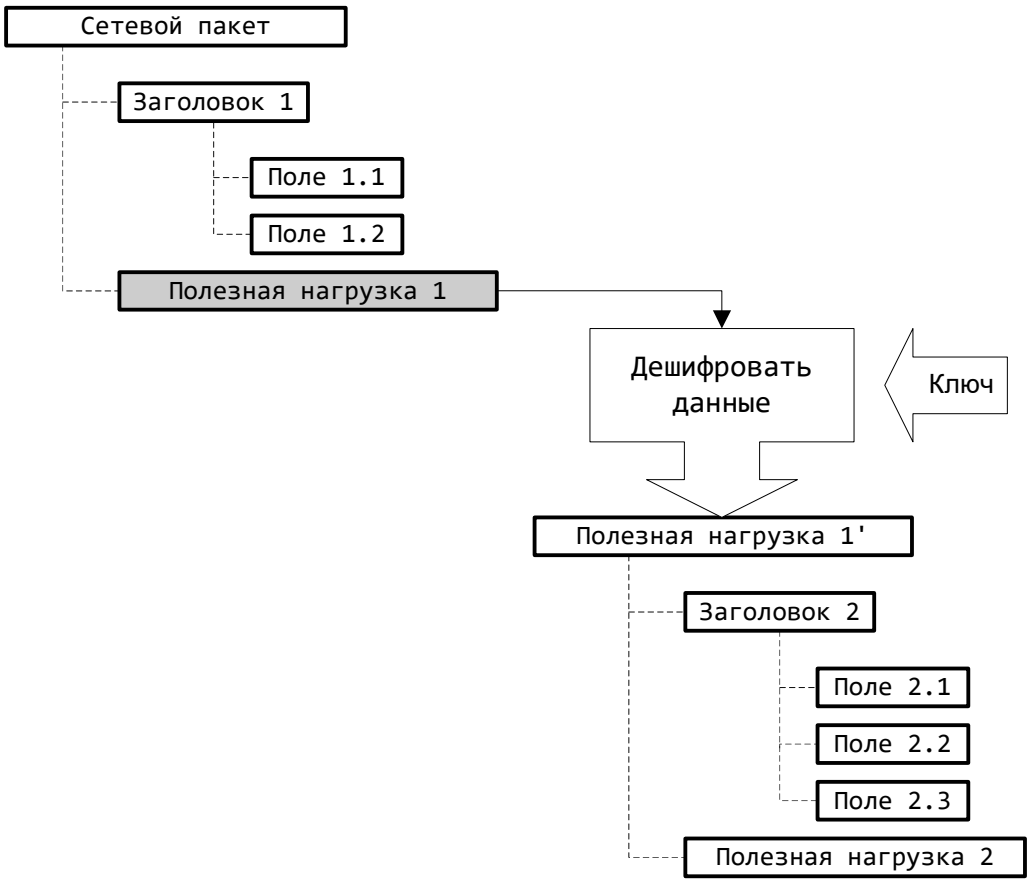

Рис. 5 – Расшифровка данных с использованием внешнего ключа.

При анализе трафика неизбежно возникают ошибки разбора. Под *ошибкой разбора* понимается несоответствие между спецификацией протокола (кодом, осуществляющим разбор) и данными, разбор которых проводится согласно этой спецификации. Причины возникновения ошибок разбора различны:

- недокументированные возможности протокола;
- искажения данных при передаче по сети;
- ошибки в коде анализатора.

Ошибки разбора должны быть легко локализуемы и воспроизводимы. Если возникшая ошибка не является критической, анализ должен продолжаться.

1.2 Требования к разбору сетевого трафика

Анализаторы сетевого трафика, как правило, имеют модульную архитектуру [26]: со временем появляются новые протоколы, и их необходимо поддерживать. Расширять систему, в которой функции разбора данных всех протоколов сосредоточены в одном функциональном модуле, затруднительно. В случае модульной архитектуры для каждого протокола создается отдельный модуль, в котором определяются методы и структуры данных для работы с этим протоколом. Возникает дополнительный вопрос о зависимостях: при добавлении нового модуля необходимо «сообщить» о его существовании остальным. Вносить изменения в код существующих модулей неэффективно: может быть нарушена логика их работы или внесены ошибки, отладка которых затруднительна. К тому же потребуется повторная сборка измененных модулей. Поэтому необходимо минимизировать количество вносимых в существующие разборщики изменений, необходимых для добавления поддержки нового протокола.

Отметим, что некоторые практические задачи решаются посредством анализа файла с сохраненным трафиком (будем называть такой файл *сетевой трассой*):

- воспроизведение ошибок разбора;
- разработка (отладка) разборщиков;
- расследование инцидентов нарушения информационной безопасности.

Поэтому крайне важно обеспечить возможность использования «результатов» offline-анализа для работы в режиме online, т.е. переносимость модулей разбора между инструментами offline- и online-анализа.

Необходимость в проведении разбора заголовков сетевых пакетов, как уже было отмечено, возникает при решении многих практических задач. Важно понимать,

что специалист в области сетевой безопасности в общем случае может не обладать высокими навыками в программировании. Поэтому необходимо предоставить высокоуровневый интерфейс (API), позволяющий пользователю поддерживать в рамках системы новые (в частности закрытые) сетевые протоколы.

С учетом особенностей процесса передачи данных по сети можно сформулировать функциональные требования к анализаторам трафика:

1. Восстановление и последующий разбор потоков данных с учетом возможного переупорядочивания и/или потери отдельных пакетов.
2. Анализ вложенных туннелей.
3. Анализ зашифрованных (модифицированных) данных.
4. Автоматическое определение вышележащего протокола.
5. Локализация и воспроизведение ошибок разбора.
6. Добавление поддержки протоколов без внесения изменений в уже существующие модули разбора.
7. Переносимость модулей разбора между режимами offline- и online-анализа.

Требования 1-3 условно можно отнести к способу представления разобранных сетевых пакетов, требования 4-7 – к архитектурным особенностям подсистемы разбора и инструмента в целом. В качестве дополнительного требования следует упомянуть о наличии графического интерфейса.

1.3 Способ оценки современных методов разбора

Проверка на соответствие требованию №1 проводится путем тестирования инструментов на наборе сетевых трасс №1 (табл. 1).

Табл. 1 – Набор сетевых трасс №1.

Сетевая трасса	**Особенности**
Trace11.pcap	• Переупорядочивание TCP-сегментов • TCP-соединения, установленные до начала записи трассы
Trace12.pcap	• Переупорядочивание TCP-сегментов

	• Повторная передача TCP-сегментов
Trace13.pcap	• Повторная передача TCP-сегментов
Trace14.pcap	• Переупорядочивание TCP-сегментов

Анализ вложенных туннелей с произвольной конфигурацией стека (требование №2) главным образом опирается на возможность автоматического определения протоколов (требование №4). Поэтому для проверки соответствующих требований проводится исследование механизмов распознавания протоколов в рассматриваемых анализаторах, а также способы организации внутреннего представления разобранных пакетов (изучается исходный код). Дополнительно на наборе сетевых трасс №2 тестируется поддержка известных туннельных протоколов (табл. 2).

Табл. 2 – Набор сетевых трасс №2.

Сетевая трасса	**Стек протоколов**
Trace21.pcap	ETH-IPv4-IPv4-ICMP
Trace22.pcap	ETH-IPv4-GRE-IPv4-ICMP
Trace23.pcap	ETH-IPv4-UDP-Teredo-IPv6-ICMPv6
Trace24.pcap	ETH-VLAN-IPv6-IPv4-GRE-PPP-IPv4-UDP-DNS

Для инструментов, поддерживающих анализ зашифрованных (в общем случае модифицированных) данных, осуществляется проверка требования №3 на наборе сетевых трасс №3 (табл. 3).

Табл. 3 – Набор сетевых трасс №3.

Сетевая трасса	**Особенности**
Trace31.pcap	Зашифрованное SSL-соединение
Trace32.pcap	Передача сжатых данных посредством HTTP

Соответствие требованиям 5-7 устанавливается непосредственно путем изучения исходного кода (с выявлением архитектурных особенностей) и сопроводительной документации инструментов.

1.4 Системы обнаружения и предотвращения вторжений

Обнаружение вторжений – это процесс мониторинга и последующего анализа событий в компьютерной системе или сети с целью выявления среди них инцидентов, нарушающих установленные политики и/или стандарты безопасности. Система обнаружения вторжений (СОВ) – это программное или аппаратное средство, автоматизирующее обнаружение вторжений.

По способу обнаружения системы разделяют на *сигнатурные* (signature-based) и *аномальные* (anomaly-based) [27]. Первые выявляют сетевые атаки посредством поиска предварительно подготовленных шаблонов (паттернов) в трафике, поступающем в подсеть или на отдельный компьютер. Сигнатурные системы, таким образом, способны обнаруживать лишь известные виды атак, тогда как не встречавшимся ранее атакам они ничего не могут противопоставить. Аномальные системы напротив ориентированы на выявление новых атак: общая идея подхода состоит в применении методов машинного обучения для построения модели *безопасного поведения* и последующего сравнения этой модели с наблюдаемым поведением. Аномальные системы обычно характеризуются большим числом ложноположительных срабатываний.

1.4.1 Snort

Работа над Snort [28] началась в 1998 году. Инструмент представляет собой сигнатурную СОВ и предназначен для предотвращения сетевых атак. Основные компоненты системы – это набор препроцессоров, предназначенных для разбора сетевых пакетов, и модули обнаружения, реализованные посредством задания правил вида *Action-Connections[-Options]*. Правила применяются к сетевым пакетам и делятся на статические (выполняются всегда) и динамические (выполняются в случае срабатывания какого-либо другого правила). *Action* определяет действие, выполняемое в случае срабатывания правила: сохранить или игнорировать сетевой пакет, активировать динамическое правило и др. *Connections* задает множество TCP или UDP соединений, к которым следует применять данное правило. В *Options* могут быть определены сигнатура и область

данных сетевого пакета для ее поиска, текстовое сообщение для записи в лог-файл, идентификатор динамического правила, активируемого данным. Например, правило на рис. 6 состоит в следующем: если в полезной нагрузке TCP-пакета, поступившего на TCP-порт 111 машины из подсети 192.168.1.0/24, будет обнаружена последовательность байт «00 01 86 a5», будет сгенерировано предупреждение с сообщением «mountd access».

```
alert tcp any any -> 192.168.1.0/24 111 \
        (content:"|00 01 86 a5|"; msg:"mountd access";)
```

Рис. 6 – Пример правила в Snort.

1.4.2 The Bro Network Security Monitor

Работа над инструментом Bro [29] началась в 1995 году. В настоящий момент система позволяет анализировать сетевой трафик на предмет выявления вторжений, с целью сбора и анализа статистики, для измерения пропускной способности сети и детектирования проблем в ее работе. Инструмент представляет собой гибридную СОВ: предусмотрены интерфейсы как для сигнатурного поиска, так и для применения алгоритмов машинного обучения.

В системе условно можно выделить два главных компонента (рис. 7): генератор событий (ядро системы) и интерпретатор скриптов, формально описывающих политику безопасности (какие действия следует предпринимать в случае генерации того или иного события). Событие может возникнуть в результате разбора одного или нескольких сетевых пакетов, а также вследствие обработки другого события (поддерживаются цепочки событий).

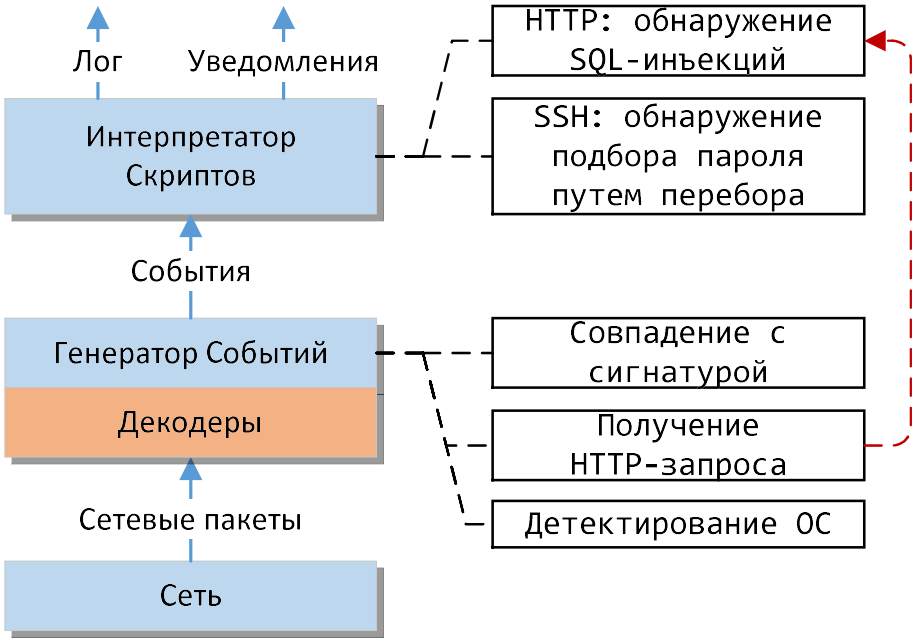

Рис. 7 – Архитектура анализатора Bro.

1.5 Универсальные анализаторы трафика

1.5.1 Wireshark

Работа над инструментом началась в 1998 году. Wireshark [30] предоставляет функциональность, схожую с возможностями программы tcpdump [31] по отладке сетевых приложений и сетевой конфигурации в целом, однако имеет графический интерфейс и позволяет выполнять фильтрацию сетевых пакетов при захвате или отображении по большому числу критериев. В отличие от Snort и Bro Wireshark сохраняет пакеты в файл, после чего проводит их разбор. Это позволяет просматривать данные всех полей всех выделенных заголовков во всех разобранных пакетах. На сегодняшний день осуществляется поддержка около 2000 протоколов; при этом количество полей, которые можно использовать для фильтрации, составляет порядка 206000 [32]. Wireshark является наиболее популярным анализатором трафика среди свободно распространяемых инструментов.

1.6 Результаты оценки

Будем называть сетевой протокол *низкоуровневым*, если он относится к физическому, канальному, сетевому или транспортному уровню модели OSI. Все

остальные протоколы (сеансовый, представительский, прикладной уровни) будем называть *высокоуровневыми*.

1.6.1 Восстановление потоков данных

В табл. 4 приведены TCP-потоки, при восстановлении которых между инструментами возникли различия: знак «+» означает, что поток данных восстановлен верно, «–» говорит о наличии неточностей.

Табл. 4 – Различия при восстановлении TCP-потоков (набор трасс №1).

##	SRC (IP:PORT)	DST (IP:PORT)	Корректное восстановление потока		
			Wireshark v.2.2.3	Snort v.2.9.7.0	Bro v.2.5
Trace11.pcap					
1	68.142.205.139:80	192.168.0.105:25168	-	-	+
2	68.142.205.139:80	192.168.0.105:25175	-	-	+
3	68.142.205.139:80	192.168.0.105:25180	+	-	+
Trace12.pcap					
4	74.125.19.113:443	192.168.0.105:24044	+	-	+
5	68.142.205.139:80	192.168.0.105:24053	-	-	+
6	68.142.205.139:80	192.168.0.105:24060	+	-	+
7	68.142.205.139:80	192.168.0.105:24089	+	-	+
Trace13.pcap					
8	74.125.224.96:443	24.6.173.220:61960	+	-	+
Trace14.pcap					
9	174.46.74.87:80	192.168.2.7:43542	-	-	+

Содержимое восстановленных потоков иногда различается. Анализ результатов показал следующее:

- Инструменты начинают восстанавливать данные TCP-соединений с момента обнаружения первого сегмента.
- Каждое из TCP-соединений 1-3, 5-7 и 9 имеет по несколько сегментов, попавших в трассу в неправильном порядке. Bro добавил данные этих сегментов правильно во всех соединениях. Snort добавил данные в том порядке, в котором они пришли. Wireshark восстановил правильный порядок сегментов для соединений 3, 6, 7, и не сделал этого для остальных.

Это произошло потому, что в соединениях 3, 6, 7 размер восстанавливаемой PDU удалось определить до того, как был разобран первый сегмент, попавший в трассу в неправильном порядке: для этого, в частности, использовалось значение поля Content-Length заголовка HTTP. В соединениях 1, 2, 5, 9 применяется порционная передача «chunked», в результате чего размер PDU определяется только при получении последней порции данных. Значит, при наличии хотя бы одной инверсии в порядке следования сегментов восстановление PDU невозможно.

- TCP-соединения 4 и 8 имеют сегмент повторной передачи. Данные этих сегментов были верно отброшены Wireshark и Bro, но ошибочно добавлены Snort.

1.6.2 Анализ вложенных туннелей

В процессе туннелирования принимают участие следующие типы протоколов:

- транспортируемый – протокол объединяемых сетей;
- несущий – протокол транзитной сети;
- инкапсулирующий – помещает пакеты транспортируемого протокола в поле данных пакетов несущего протокола.

При анализе вложенных туннелей критическими для системы являются возможность автоматического определения вышележащего протокола (требование №4), а также поддержка стека протоколов произвольной глубины.

В Wireshark предусмотрено три способа для активации разборщика заголовка вышележащего протокола:

1. Прямой вызов разборщика: непосредственный вызов заданной функции разбора в коде.
2. Обратный вызов разборщика с привязкой по значению: вызываемый разборщик определяется значением, полученным при разборе заголовка нижележащего протокола. Так, например, используются поля *EtherType* и *Protocol* в заголовках Ethernet и IP соответственно.

3. Вызов разборщика посредством эвристической привязки. Эвристики применяются, когда в заголовке нижележащего протокола отсутствуют признаки для точного определения следующего протокола. В частности, номера портов TCP и UDP не задают вышележащий протокол точно: данные, передаваемые по порту TCP 80 как правило представляют собой HTTP-трафик, однако в общем случае это может быть не так.

Разобранный сетевой пакет в Wireshark представляется в виде дерева, вершины которого описывают заголовки протоколов и выделенные в них поля. Такой подход позволяет анализировать вложенные туннельные протоколы: возможность добавления в дерево нового уровня была предусмотрена разработчиком.

Большинство препроцессоров Snort жестко связаны с номерами портов TCP/UDP: соответствие устанавливается посредством конфигурационного файла. Таким образом, сетевые пакеты протокола P, поступающие на порт X, будут разбираться только в случае, если X входит в множество отслеживаемых портов декодера P. Только препроцессоры для SSH [33], DCE/RPC [34] и SMB [35] позволяют автоматически определять соответствующий протокол. Вероятно, разработчик отказался от динамического распознавания протоколов в большинстве препроцессоров для увеличения скорости анализа.

Сетевой пакет в Snort описывается с помощью структуры с фиксированным набором полей. В частности, зафиксировано максимальное количество IP-заголовков в пакете, равное двум. При этом, значения полей структуры не могут быть перезаписаны в процессе анализа. Поэтому, если в трафике встретится сетевой пакет с тремя IP-заголовками, то его разбор прекратится на третьем IP-заголовке.

В анализаторе Bro фактически представлены те же механизмы активации разборщиков, что и в Wireshark: предусмотрена привязка к номерам портов TCP/UDP, а также к предварительно заданным сигнатурам (аналог эвристических разборщиков). На рис. 8 приведены сигнатуры для выявления HTTP-соединений в трафике. В отличие от Wireshark, где эвристики кодируются непосредственно на языке Си, Bro предоставляет язык описания сигнатур [36], что существенно

упрощает разработку. Разборщики протоколов канального, сетевого и транспортного уровней вызываются непосредственно.

```
signature dpd_http_client {
    ip-proto == tcp
    payload /^ *(GET|HEAD|POST) */
    tcp-state originator
  }

signature dpd_http_server {
    ip-proto == tcp
    payload /^HTTP\/[0-9]/
    tcp-state responder
    requires-reverse-signature dpd_http_client
    enable "http"
  }
```

Рис. 8 – Сигнатуры для выявления HTTP-соединений в Bro.

Структура для описания сетевого пакета в Bro содержит сведения только о канальном уровне. Информация о сетевом и транспортном уровнях хранится в структуре, описывающей *соединение*. Каждое такое соединение определяется пятеркой: два IP-адреса, два порта и протокол транспортного уровня. Для данного соединения хранится стек туннельных соединений, в рамках которых оно было организовано. Каждый элемент такого стека представляет собой либо пару IP-адресов (для описания инкапсуляции вида IP-in-IP), либо пятерку, описывающую соединение несущего протокола. Предусмотрено определение максимально возможной вложенности туннеля: если количество элементов в стеке туннельных соединений превысит пороговое значение, анализ вышележащих заголовков проводиться не будет.

Тестирование показало (табл. 5), что на всех трассах набора №2 Wireshark и Bro верно определили и разобрали все вложенные заголовки протоколов. У Snort возникли затруднения при анализе Trace24.pcap: заголовки ETH-VLAN-IPv6-IPv4-GRE [37, 38] были разобраны, заголовки PPP-IPv4-UDP-DNS [39, 40] – нет. Такой результат Snort полностью согласуется с документацией [41].

Табл. 5 – Поддержка туннельных протоколов (набор трасс №2).

Сетевая трасса	Разбор всех заголовков стека		
	Wireshark v.2.2.3	Snort v.2.9.7.0	Bro v.2.5

Trace21.pcap	+	+	+
Trace22.pcap	+	+	+
Trace23.pcap	+	+	+
Trace24.pcap	+	-	+

Таким образом, анализ вложенных туннелей возможен в Wireshark и Bro, тогда как Snort изначально не был на него рассчитан.

1.6.3 Анализ модифицированных данных

Wireshark поддерживает возможность анализа модифицированных данных: для разбора зашифрованных пакетов в рамках заданного протокола следует указать путь к файлу с ключами для дешифрования, распаковка сжатых данных осуществляется автоматически. Snort не предполагает проведение дешифрования по соображениям производительности. Для препроцессоров протоколов, поддерживающих шифрование, предусмотрена специальная опция «игнорировать зашифрованный трафик» с тем, чтобы уменьшить число ложноположительных срабатываний [42]. В Bro также не предполагается возможность расшифровки. Декомпрессия данных поддерживается как в Snort, так и в Bro. В табл. 6 представлены результаты тестирования инструментов на наборе сетевых трасс №3.

Табл. 6 – Результаты тестирования на наборе трасс №3.

Сетевая трасса	Анализ модифицированных данных		
	Wireshark v.2.2.3	Snort v.2.9.7.0	Bro v.2.5
Trace31.pcap	+	-	-
Trace32.pcap	+	+	+

1.6.4 Локализация ошибок разбора

Wireshark позволяет регистрировать ошибки, возникающие в результате несоответствия разбираемого пакета формату, определяемому разборщиком соответствующего протокола: в графическом интерфейсе предусмотрена компонента отображения всех таких ошибок, которая также позволяет переключаться к «нужному» сетевому пакету. Snort и Bro не предполагают сохранение всего проанализированного трафика, однако с помощью правил и

событий может быть реализована запись в файл пакетов, разбор которых приводит к возникновению ошибки. В дальнейшем потребуется использовать программу просмотра сохраненных пакетов (например, hex-editor) для того, чтобы выяснить какие именно байты в них не соответствуют спецификации, что затруднительно.

1.6.5 Переносимость разборщиков между offline- и online-режимами

Wireshark, Snort и Bro позволяют анализировать трафик, поступающий на сетевой интерфейс, а также предварительно сохраненный в файле. При этом используются одни и те же разборщики заголовков.

1.6.6 Добавление поддержки новых протоколов

Поддержка нового протокола без модификации существующего кода в Wireshark может быть реализована посредством привязки по значению или эвристической привязки. Модули разбора протоколов всех уровней в Wireshark реализованы единообразно, чего нельзя сказать о Snort и Bro: если разборщики протоколов канального, сетевого и транспортного уровней жестко связаны посредством прямых вызовов, то анализ высокоуровневых протоколов выполняется с учетом возможностей динамического распознавания (Bro) или задания диапазонов портов (Snort). Таким образом, для Bro и Snort поддержка нового низкоуровневого протокола потребует изменить существующий код.

Модули разбора в Wireshark не являются независимыми: SSL-разборщик использует информацию о восстановлении PDU из нескольких TCP-сегментов, HTTP-разборщику требуются функции, применяемые при анализе SSL-трафика и др. Подобная организация кода увеличивает его связность и затрудняет дальнейшую разработку: в случае изменения кода разборщика данного протокола необходимо проконтролировать корректность кода во всех зависимых разборщиках. Высокоуровневые декодеры Snort и Bro также зависят от низкоуровневых, но в значительно меньшей степени, нежели в Wireshark.

1.6.7 Выводы

Рассмотренные инструменты в полной мере не соответствуют предъявленным требованиям. Модули разбора в них сильно связаны на уровне исходного кода, что предполагает проведение большого объема работы в случае внесения в код изменений. Только Bro правильно восстанавливает потоки данных, Snort и Wireshark не справляются с этой задачей. Несмотря на возможность применения одних и тех же разборщиков при проведении online и offline анализа, каждый инструмент всегда использует одну и ту же стратегию обработки пакетов. Snort и Bro удаляют (drop) пакет сразу после проведения разбора, что делает затруднительным их применение для отладки разборщиков (offline). При проведении online-анализа Wireshark сохраняет все сетевые пакеты в физической памяти, что, ввиду конечности объема памяти, не позволяет проводить разбор в течение продолжительного времени.

Для достижения поставленной цели необходимо разработать модель представления данных, а также систему анализа трафика на ее основе.

2. Модель представления разбора сетевого трафика

Будем рассматривать сетевые взаимодействия между приложениями, не фиксируя какой-либо конкретный стек протоколов (TCP/IP [43], IPX/SPX [44]). Будем считать, что сетевой обмен осуществляется посредством произвольно стека протоколов, насчитывающего N уровней (рис. 9). Сетевое взаимодействие рассматривается как упорядоченный набор *логических соединений*. В рамках каждого соединения передается последовательность пакетов между логическими объектами одного уровня.

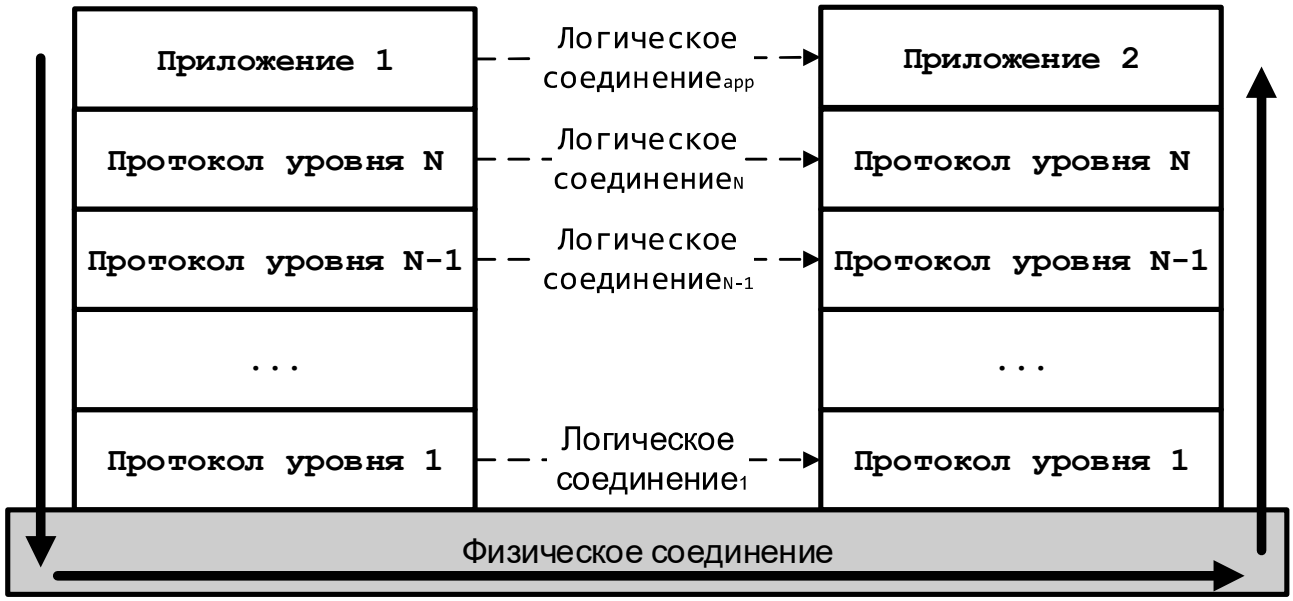

Рис. 9 – Представление сетевого взаимодействия между двумя приложениями.

Пусть логическое соединение на l-м уровне ($l = \overline{1,N}$) реализуется посредством протокола $Protocol^l$ и содержит K_l пакетов. Каждый протокол задает множество допустимых форматов пакетов $\{F_j\}$. Под форматом будем понимать разделение последовательности байт пакета на именованные поля и ограничения на значения некоторых полей. Как правило, полезная нагрузка – данные протокола верхнего уровня – помещена в одно поле, остальные поля выполняют служебные функции: указывают тип формата сообщения, несут в себе адресную информацию, задают размеры других полей и т.п. Упрощенно пакет протокола $Protocol^l$ будем

рассматривать как кортеж, состоящий из совокупности полей с управляющей информацией и полезной нагрузки: $Packet^l$=<$Control^l$, $Payload^l$>, $l=\overline{1,N}$.

Все сетевые протоколы можно разделить на два непересекающихся множества: допускающие и не допускающие фрагментацию данных. Фрагментация, в свою очередь, может быть организована различными способами. Будем называть *простой* фрагментацию, при которой пакет протокола $Protocol^l$ может содержать фрагмент не более одного пакета протокола $Protocol^{l+1}$ (в частности, IPv4 относится к протоколам с простой фрагментацией). *Сложная* фрагментация предполагает, что полезная нагрузка пакета протокола $Protocol^l$ может содержать фрагменты произвольного числа пакетов протокола $Protocol^{l+1}$ (примером может служить процесс передачи SSL-записей посредством TCP-сегментов).

При наблюдении состояния сетевого соединения фиксируется прохождение по нему последовательности пакетов протокола нижнего уровня. Пакеты относятся к произвольному числу приложений, попарно взаимодействующих или выполняющих широковещательную рассылку данных. Полное воспроизведение функциональности сетевого протокола при разборе пакетов не требуется, необходимо разделять поток байт на отдельные пакеты (сообщения), выбирая необходимый формат, а затем интерпретировать управляющую информацию $Control^l$ с целью группировки пакетов и извлечения полезной нагрузки.

Углубленный анализ трафика подразумевает восстановление и группировку последовательности пакетов для всех логических соединений всех уровней. Необходимо различать пакеты протокола $Protocol^l$, относящиеся к разным логическим соединениям – управляющая информация i-го пакета протокола $Protocol^l$ должна позволять идентифицировать логическое соединение, и тем самым, ассоциировать с ним этот пакет: $id_i^l = key^l\left(Control_i^l\right), i = \overline{1,K_l}$, где key^l – функция получения идентификатора логического соединения в рамках протокола уровня l. В рамках одного логического соединения уровня l может быть организовано несколько логических соединений уровня $l+1$. Для обеспечения

уникальности идентификатора логического соединения необходимо дополнить его идентификаторами всех «нижележащих» логических соединений: $uid_i^l = \{id_i^j\}$, $j=\overline{1,l}$. В случае произвольного стека протоколов потребуется дополнительно хранить идентификаторы протоколов: $uid_i^{P^l} = \left\{< P_i^j, id_i^{P^j} >\right\}$, $j=\overline{1,l}$.

Порядок получения пакетов может отличаться от порядка их отправления, поэтому для протоколов, допускающих фрагментацию, на основе управляющей информации пакета должны строиться *предикаты предшествования* (определение предыдущего фрагмента) и *следования* (определение следующего фрагмента), с помощью которых будет проводиться дефрагментация. При переходе к анализу пакетов вышележащего логического соединения пакеты данного логического соединения должны быть переупорядочены в соответствии со значениями введенных предикатов.

2.1 Сущности модели

Далее описывается модель представления разобранных сетевых пакетов, удовлетворяющая перечисленным в подразделе 1.2 требованиям и позволяющая рассматривать сетевые взаимодействия в виде последовательности логических соединений.

2.1.1 Сетевой пакет

На рис. 10 изображена диаграмма классов, с помощью которых представляется разобранный сетевой пакет. Для восстановления последовательности пакетов логического соединения требуется извлечение полезной нагрузки пакетов нижележащего логического соединения. Извлечение данных в разработанной модели сопровождается их копированием только при необходимости проведения дефрагментации. Благодаря такой стратегии повышается производительность по сравнению с подходами, требующими копировать полезную нагрузку при извлечении в общем случае.

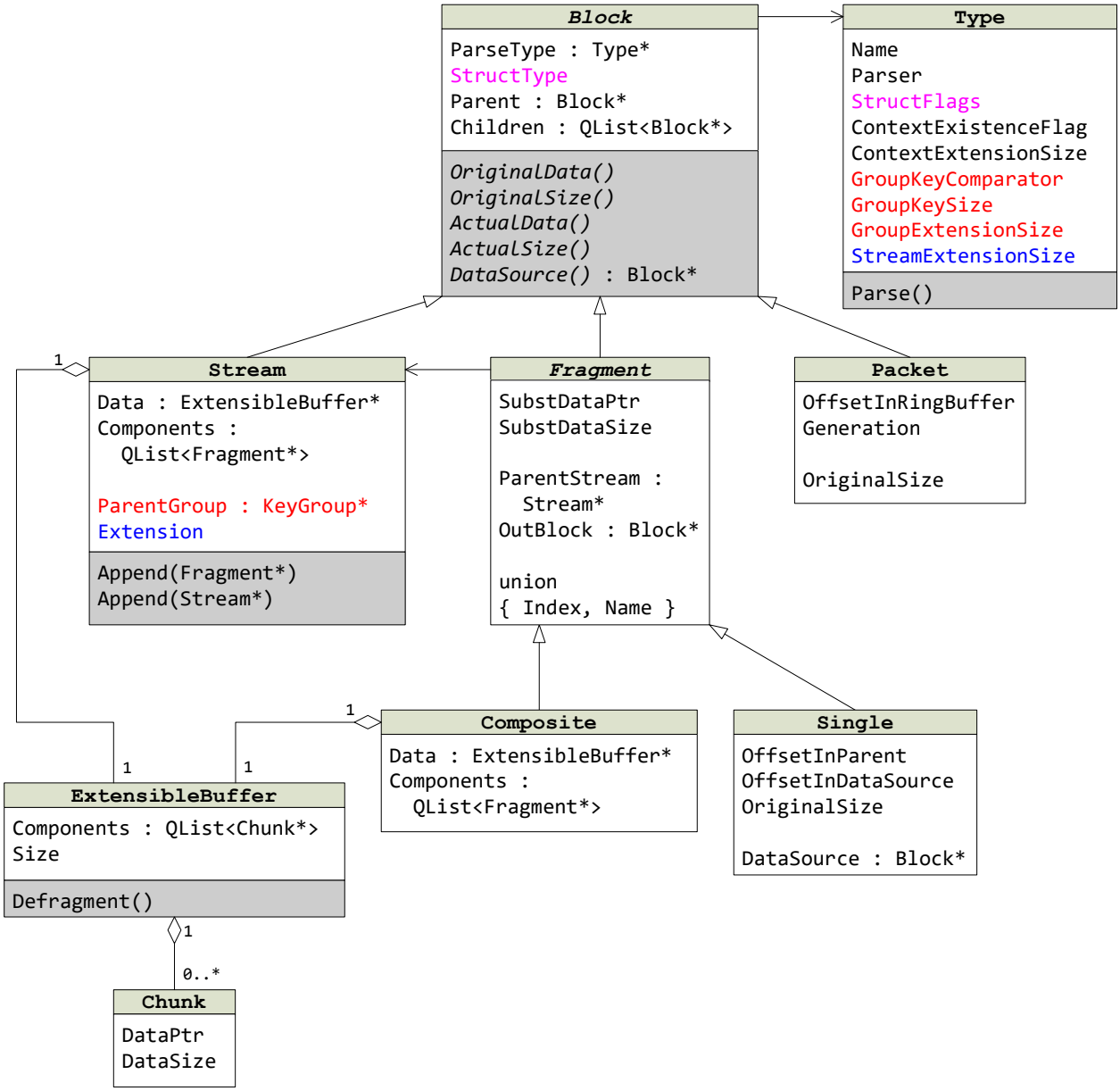

Рис. 10 – Диаграмма классов для описания разобранных сетевых пакетов.

Выделенные в процессе разбора структурные единицы (сетевые пакеты и их поля) описываются посредством *блоков (класс Block)*. Блок представляет массив байт некоторого фиксированного размера. При анализе блока может потребоваться отдельно обработать какую-либо его часть. Для этого будет создан новый блок, являющийся дочерним по отношению к исходному родительскому блоку. В процессе анализа каждому блоку ставится в соответствие *тип разбора (поле ParseType класса Block)*, определяющий функцию *(поле Parser класса Type)* для разбора данных этого блока. Разбор заключается в разделении буфера на именованные поля согласно заданному формату пакета *F*. Для блока определяется

базовый формат данных *(поле StructType класса Block)*. Основные базовые форматы:

- *Struct* – блок содержит разнородные поля, аналог типа *struct* языка Си (*ProtoStruct* для случаев, когда описывается заголовок сетевого протокола);
- *FieldSeq* – последовательность однородных полей, аналог типа массив языка Си;
- *PacketSeq* – последовательность сетевых пакетов (возможно различных форматов) одного протокола.

Вводится два вида блоков: *блок-поток* – для описания восстановленных потоков данных, и *блок-фрагмент* – для представления целых пакетов и отдельных полей, выделенных в потоках данных. Восстановление потока подразумевает объединение полей (как правило, это поле с полезной нагрузкой) из разных сетевых пакетов. С этой целью вводится понятие *накопительного буфера (класс ExtensibleBuffer)*, позволяющего объединять как данные блоков-фрагментов, так и данные блоков-потоков. Отметим, что для блока-потока блок-родитель не определен, поскольку отношение «родитель-потомок» не применяется для восстановленных потоков.

Для размещения в памяти расшифрованных (или распакованных) данных вводится понятие *замещающего буфера*: его размер фиксирован, в отличие от накопительного буфера, увеличивающегося по мере добавления данных. Блок-фрагмент с непустым замещающим буфером будем называть *замещающим блоком-фрагментом*: дальнейшему разбору подлежат данные замещающего буфера. Дочерние блоки замещающего блока-фрагмента характеризуются смещением относительно соответствующего замещающего буфера. Отметим, что размер замещающего буфера никак не связан с размером исходных (зашифрованных или сжатых) данных блока-фрагмента. По определению, блок-поток не может содержать замещающий буфер.

Замещающий блок-фрагмент, таким образом, характеризуется первоначальными (зашифрованными) и фактическими (расшифрованными) данными. Обобщая эти

понятия для блока, будем называть *фактическими* данные блока, подлежащие дальнейшему разбору, и *первоначальными* данные без учета возможного замещения. Для блока-потока первоначальные данные совпадают с фактическими.

Вводится два вида блоков-фрагментов: *составной (класс Composite)* – для описания дефрагментированных PDU, и *одиночный (класс Single)* – для представления простых полей. Для краткости далее будем называть составной блок-фрагмент *составным фрагментом*, а одиночный блок-фрагмент – *одиночным фрагментом*. Дефрагментация PDU, как и восстановление потока, подразумевает объединение полей из разных пакетов, поэтому составной фрагмент владеет расширяемым буфером *(поле Data класса Composite)*. В то же время, составной фрагмент предназначен для описания одной PDU, тогда как в буфере блока-потока, как правило, содержится несколько PDU.

Будем называть *источником данных блока (поле DataSource класса Block)* блок, содержащий буфер, в котором содержатся первоначальные данные первого. Значит, в качестве источника данных может выступать блок-поток, составной фрагмент, замещающий блок-фрагмент. Одиночный блок локализуется в данных своего источника посредством смещения *(поле OffsetInDataSource класса Signle).*

В соответствии со структурным типом родительского блока блок-фрагмент описывает либо поле структуры (характеризуется именем *Name*), либо элемент последовательности (характеризуется индексом *Index*).

2.1.2 Логическое соединение

На рис. 11 представлена диаграмма классов, посредством которых описываются логические соединения.

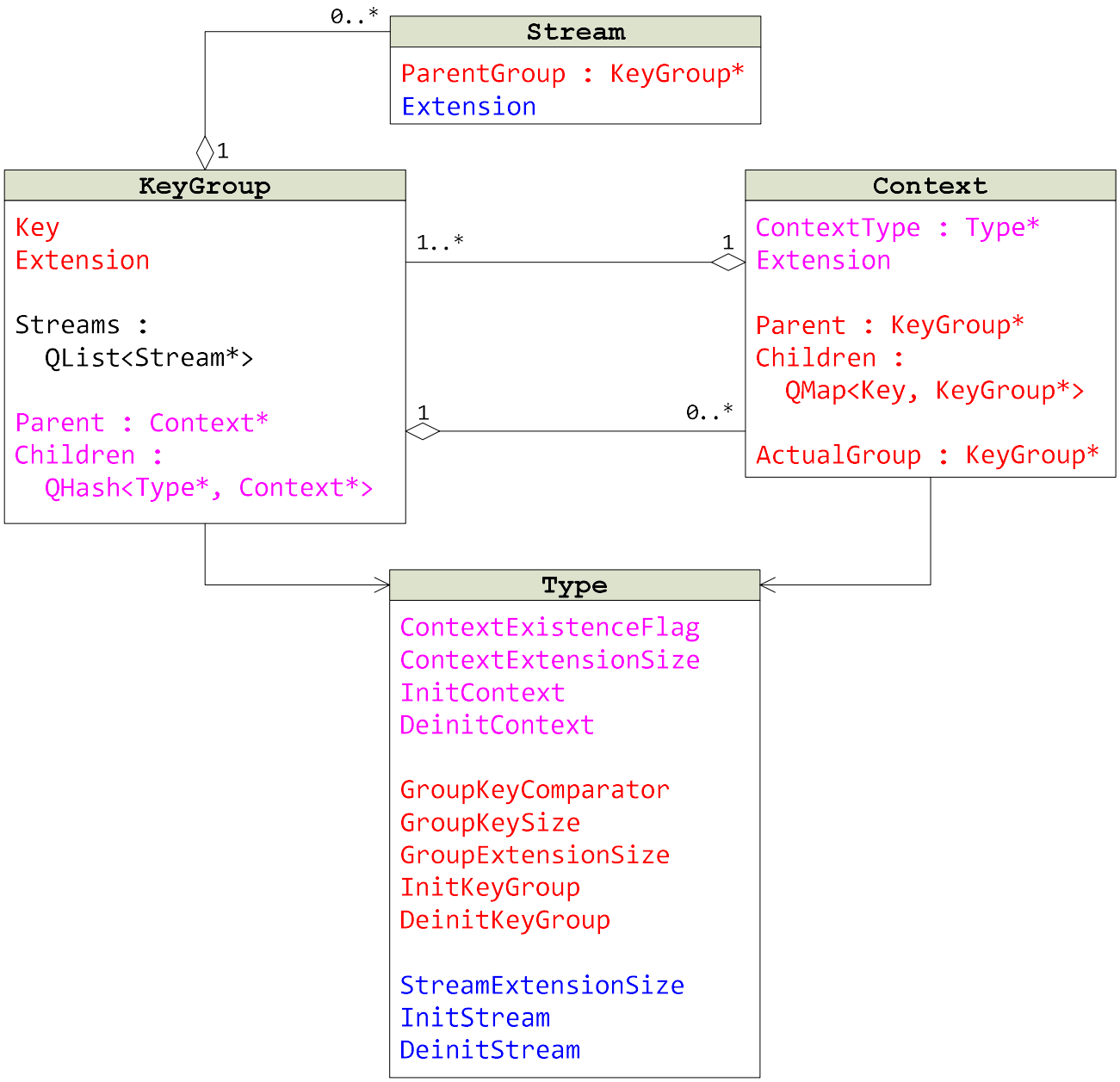

Рис. 11 – Диаграмма классов для описания логических соединений.

Каждое логическое соединение организовано посредством одного сетевого протокола и представляется в виде последовательности пакетов этого протокола. В то же время, на одном уровне стека протоколов могут быть представлены пакеты различных протоколов. Для выделения последовательностей пакетов, относящихся к заведомо различным логическим соединениям, необходимо сгруппировать пакеты заданного уровня стека в соответствии с протоколами этих логических соединений. Группа пакетов некоторого протокола в рамках заданного уровня стека описывается *контекстом (класс Context)*. Сетевой

протокол, на основании которого был сформирован контекст, по определению задает *тип контекста (поле ContextType класса Context)*. В *расширении (поле Extension класса Context)* контекста может храниться дополнительная информация, необходимая для проведения разбора пакетов, относящихся к данному контексту (например, состояние декомпрессора при анализе пакетов протокола PPP). Тип контекста используется, в частности, при построении идентификаторов логических соединений.

Логическое соединение протокола некоторого уровня должно быть представлено последовательностью восстановленных пакетов. Для восстановления (извлечения полезной нагрузки) необходимо предварительно выделить соответствующую подпоследовательность из последовательности пакетов этого протокола. Для группировки пакетов в подпоследовательности используется адресная информация из служебных полей. Каждая подпоследовательность описывается *ключевой группой (класс KeyGroup)*. Такая группа в контексте идентифицируется посредством уникального ключа *(Key)*, содержимое и размер которого определяются в соответствии с типом контекста. Контекст выступает в роли контейнера по отношению к таким группам и предоставляет к ним доступ по ключу. Ключевая группа обладает *расширением (Extension)* для сохранения информации, необходимой при проведении разбора блоков (пакетов) этой группы. Ключевая группа фактически представляет собой логическое соединение.

Будем называть *восстановленным потоком протокола P* блок-поток, в накопительный буфер которого добавлены данные блоков-фрагментов одной ключевой группы (возможно переупорядочивание и/или перекрытие блоков-фрагментов, в соответствии с семантикой протокола *P*) контекста с типом, описывающим протокол *P*. Соответствующая ключевая группа называется *родительской (поле ParentGroup класса Stream)* по отношению к восстановленному потоку. В рамках одной ключевой группы поддерживается одновременное существование нескольких потоков *(поле Streams класса KeyGroup)* – необходимость в этом может быть обусловлена как семантикой соответствующего протокола, так и потерей отдельных пакетов при передаче.

Расширение потока *(поле Extension класса Stream)* используется для сохранения необходимой при восстановлении этого потока информации.

Поясним на примере протокола TCP, как управляющая информация используется для переупорядочивания пакетов логического соединения.

- Начало потока определяется TCP-пакетом с установленным флагом SYN, конец – TCP-пакетом с установленным флагом FIN или флагом RST.
- В соответствии со значениями порядковых номеров (SEQ) и номеров подтверждения (ACK) может быть проведено переупорядочивание TCP-сегментов.

В результате восстановления логических соединений различных взаимодействующих по сети приложений будет построено дерево контекстов, ключевых групп и восстановленных потоков (далее, *дерево контекстов*) с вершинами трех типов:

- вершина-контекст задает протокол;
- вершина-группа характеризуется ключом и описывает логическое соединение на уровне протокола родительской вершины;
- вершина-поток описывает восстановленный поток данных протокола вышележащего уровня.

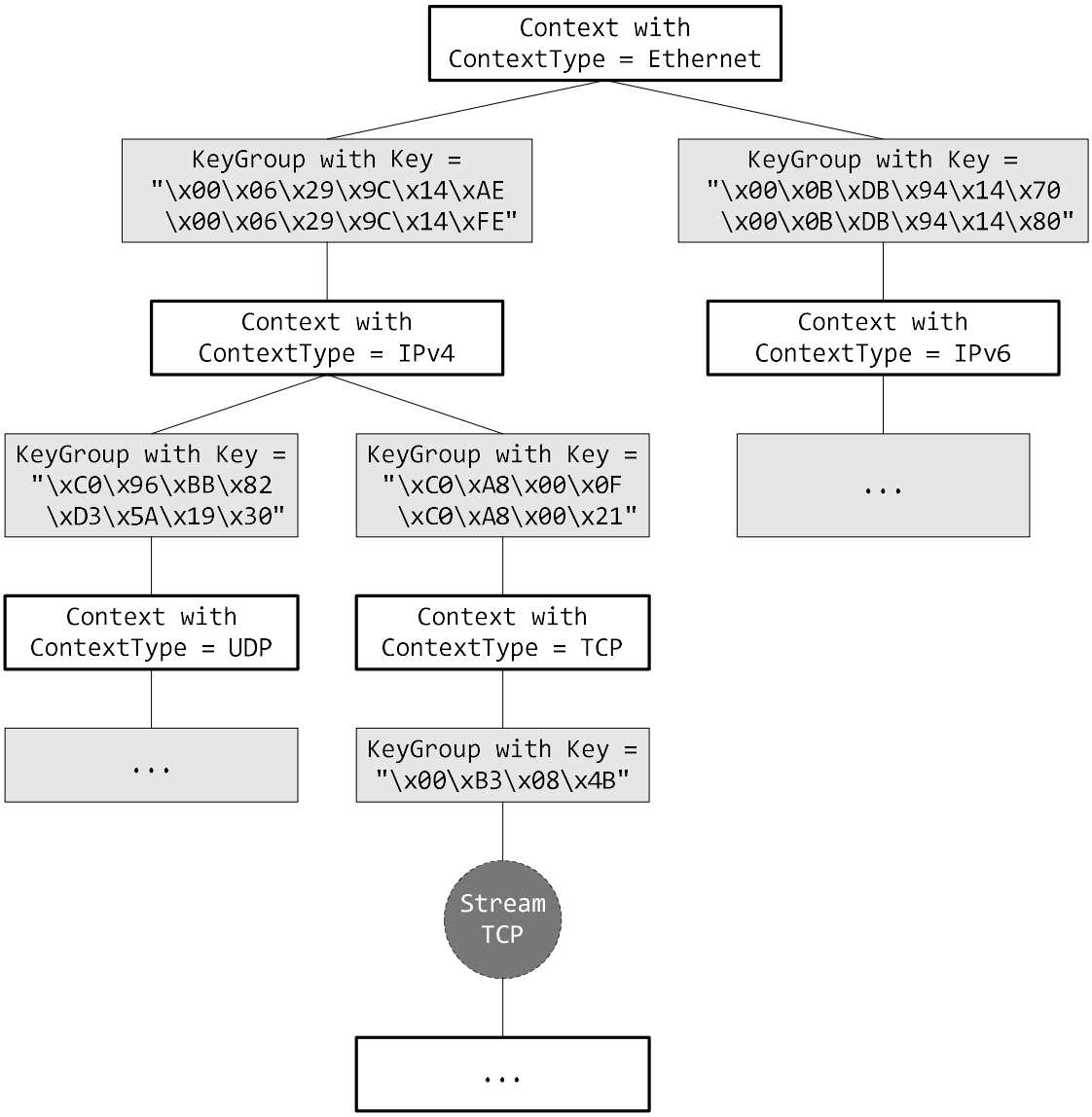

Рис. 12 – Пример дерева контекстов для сети Ethernet.

На рис. 12 приведен пример дерева контекстов, в котором в качестве протокола нижнего уровня, наблюдаемого в сетевом соединении, фигурирует Ethernet. Ключ для Ethernet-группы строится путем конкатенации MAC-адресов отправителя и получателя (размер ключа равен 12 байтам). Для протокола IPv4 ключ группы состоит из пары IP-адресов (8 байт), для протокола TCP – из пары портов (4 байта).

2.1.3 Участники сетевого обмена

Ключи выделяемых групп пакетов представляют собой массивы байт. Это, с одной стороны, позволяет конструировать ключ и группировать пакеты

произвольным образом, с другой – приводит к потере информации о направлении движения пакета в сети. В то же время знания об адресах отправителя и получателя могут использоваться для построения топологии сети, трафик которой анализируется, выявления запущенных на компьютерах служб, визуализации сетевых взаимодействий. Для решения этих задач вводится понятие *сетевого узла (класс NetworkNode),* относящегося к протоколу некоторого уровня. Для идентификации сетевого узла задается соответствующий протокол и имена полей в его сообщениях, в которых содержится адресная информация. Для унификации описания используется тройка *(ProtoType, ValueType, Value)*, где *ProtoType* – протокол*, ValueType* – тип разбора блока, описывающего «адресное» поле, *Value* – значение «адресного» поля. Например, сетевой узел протокола IPv4 задается с помощью IPv4-адреса следующим образом:

```
1. ProtoType : IPv4
2. ValueType : IPv4Addr
3. Value     : 10.10.14.102
```

Если для одних протоколов данные о сетевых узлах содержатся в заголовке непосредственно в виде полей, то для других имеются только косвенные признаки: несмотря на то, что в HTTP-заголовке в явном виде отсутствует поле, идентифицирующее отправителя (клиент или сервер), можно сделать соответствующий вывод, проанализировав содержимое стартовой строки. Для работы с такими сценариями `Value` может принимать предопределенные значения, соответствующие *сетевой роли* узла: *client, server, peer, unknown*. При этом значение поля `ValueType` будет пустым.

Адресной информации текущего протокола недостаточно для идентификации логического соединения. Например, при организации туннельного соединения GRE пакеты, отправляемые разными участниками сетевого обмена, будут содержать заголовки с одними и теми же IP-адресами. Для преодоления этой проблемы вводятся *цепочки сетевых узлов*. Цепочка строится путем добавления в ее конец сетевых узлов (либо отправителей, либо получателей) протоколов

нижележащих уровней, возникших в процессе разбора сетевого пакета. Сетевой узел в цепочке, непосредственно предшествующий данному, будем называть *родительским*. Сетевой узел, непосредственно следующий за данным узлом в цепочке, будем называть *дочерним*. Для первого узла цепочки, таким образом, не определен родительский узел, а для последнего – дочерний. Будем называть *цепочкой протоколов* последовательность значений поля ProtoType для заданной цепочки сетевых узлов.

Для компактного представления всех построенных цепочек и организации быстрого доступа к ним используется префиксное дерево. Для гарантированного наличия общего префикса у всех цепочек добавим в начало каждой цепочки дополнительный фиктивный сетевой узел с неопределенными *ProtoType* и *ValueType* и пустым *Value*. В построенном префиксном дереве *(класс NetworkTree)* путь от корня к листовой вершине однозначно идентифицирует участника сетевого обмена. Для описания обмена пакетами между сетевыми узлами одного уровня NetworkTree дополняется *горизонтальными ребрами (класс Edge)*. Горизонтальное ребро соединяет два сетевых узла, если:

- цепочки протоколов сетевых узлов одинаковы;
- между узлами передан хотя бы один сетевой пакет протокола, описываемого с помощью ProtoType.

На рис. 13 приведен пример NetworkTree, где в качестве протоколов наиболее низкого и высокого уровней фигурируют Ethernet и HTTP соответственно.

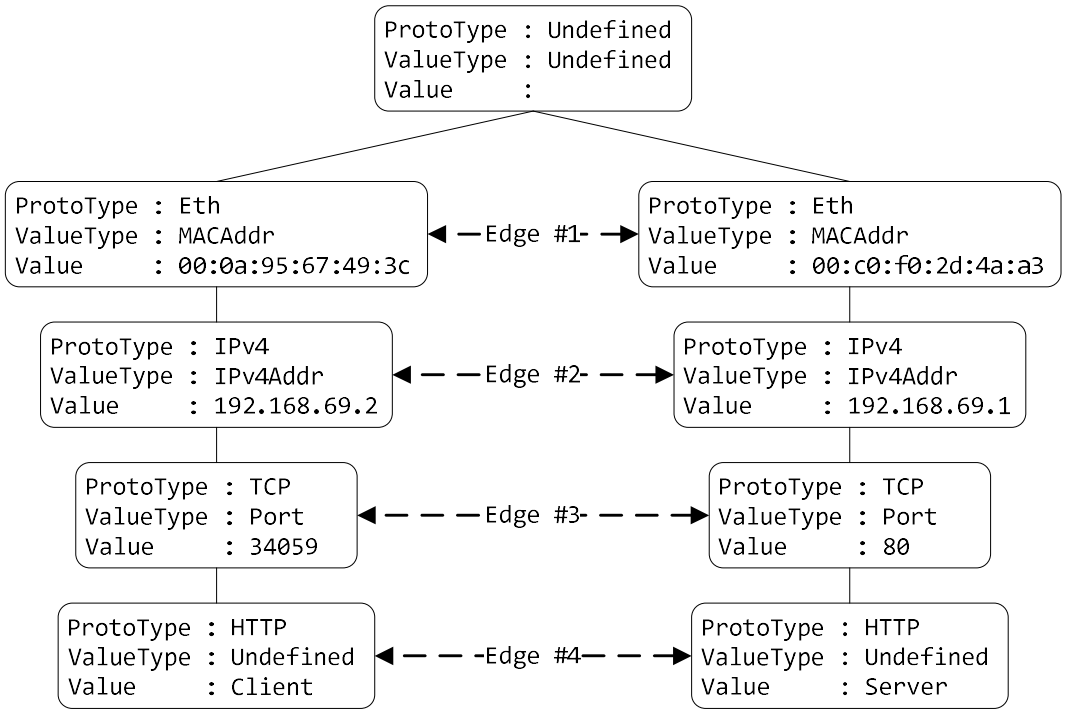

Рис. 13 – Пример NetworkTree.

Для описания множества пакетов, передаваемых между парой сетевые узлов, вводится понятие *диалога (Dialog)*. Диалог описывает взаимодействие между узлами с одинаковыми цепочками протоколов. Тип диалога определяется значением ProtoType взаимодействующих узлов. Для каждого пакета диалога определено направление передачи: от первого узла ко второму или наоборот. Между диалогами и ребрами, таким образом, устанавливается взаимно однозначное соответствие. Для широковещательных рассылок вводятся фиктивные сетевые узлы.

NetworkTree позволяет рассматривать сетевой обмен на разных уровнях, начиная от канального и заканчивая прикладным (в соответствии с уровнями модели OSI). Диаграмма классов для его описания представлена на рис. 14.

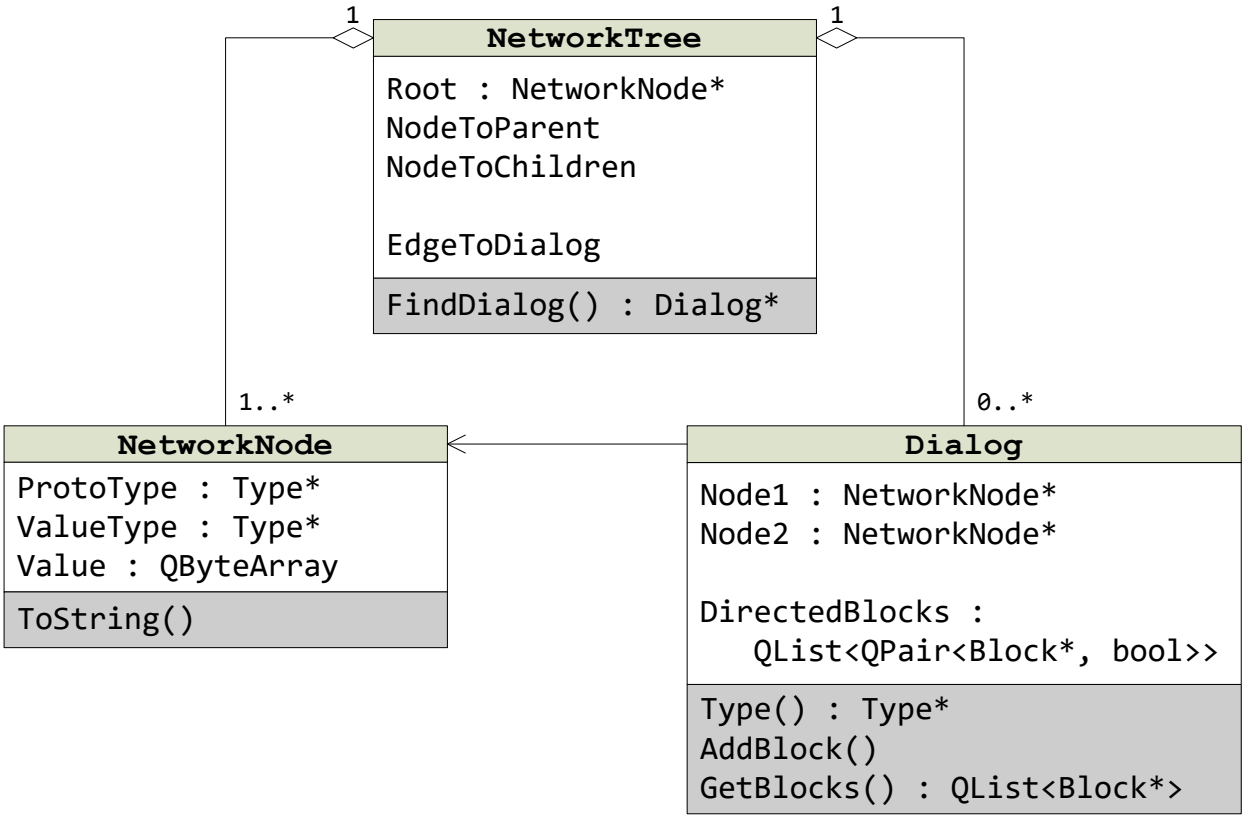

Рис. 14 – Диаграмма классов для описания участников сетевого обмена.

2.2 Связывание уровней произвольного стека протоколов

Для обеспечения разбора пакетов произвольного стека протоколов вводится понятие распознавателя. *Распознаватель* – это функция, которая по данным блока и, возможно, некоторой дополнительной информации определяет тип разбора этого блока. Обычно функциональность по распознаванию требуется для полей в заголовках протоколов, допускающих данные различных форматов.

Например, полезная нагрузка в пакете протокола IPv4 может содержать пакеты протоколов TCP, UDP, GRE, ICMP и др. Служебное поле, определяющее вышележащий протокол, содержится в IPv4-заголовке. Таким образом, для определения типа разбора полезной нагрузки используются данные блока-родителя. Соответствующую категорию распознавателей будем называть *распознавателями полей*.

Для определения типа разбора восстановленных потоков используется другая категория распознавателей – *распознаватели потоков с известным типом сборки*. Распознаватель получает на вход содержимое накопительного буфера

потока и информация о типе разбора блоков, данные которых были добавлены в этот накопительный буфер.

Распознаватели, принимающие на вход только данные буфера, будем называть *универсальными*. Такие распознаватели применяются к блоку-потоку, если его тип сборки неизвестен, а также блоку-фрагменту, при условии, что с помощью распознавателей полей не удалось определить тип разбора.

2.3 Алгоритм восстановления потоков сетевых данных при нарушении порядка следования пакетов

На рис. 15 приведен пример несовпадения порядка отправления с порядком получения пакетов. В табл. 7 описана последовательность типовых действий принимающей стороны для обработки такой ситуации.

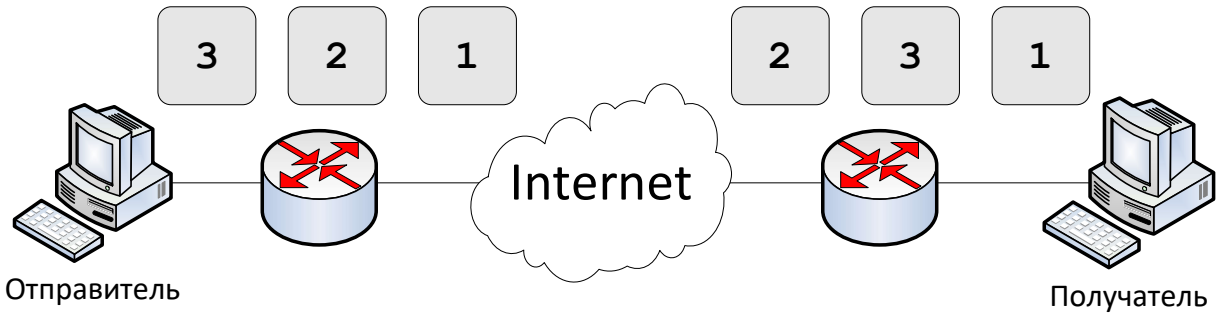

Рис. 15 – Пример переупорядочивания пакетов при передаче.

Будем называть поток восстановленным *полностью*, если выполнены следующие три условия:

1. поток обладает признаком начала (например, флаг SYN для TCP),
2. поток обладает признаком конца (например, флаг FIN для TCP),
3. буфер потока содержит все данные потока без пропусков.

В противном случае будем называть поток *частично* восстановленным.

Табл. 7 – Обработка переупорядоченных пакетов.

Событие	Действие
Получен пакет #1	• Создать блок-поток #1 • Добавить данные блока-фрагмента #1 в блок-поток #1
Получен пакет #3	• Создать блок-поток #2

	• Добавить данные блока-фрагмента #3 в блок-поток #2
Получен пакет #2	• Добавить данные блока-фрагмента #2 в блок-поток #1 • Объединить блоки-потоки #1 и #2

Будем называть блок b_1 *префиксом* блока b_2, если выполнены следующие три условия:

1. b_1 и b_2 принадлежат одной ключевой группе;
2. b_1 и b_2 описывают полезную нагрузку некоторого протокола P;
3. в соответствии с семантикой протокола P данные b_1 непосредственно предшествуют данным b_2.

На рис. 16 приведено формальное описание алгоритма восстановления потоков (для упрощения опущены детали обработки пакетов-дубликатов).

```
Пусть M - множество частично восстановленных потоков ключевой группы G.

для каждого блока-фрагмента f группы G
{
  для каждого блока-потока s из M
  {
    если s является префиксом f
    {
      добавить данные f в s

      для каждого блока-потока s' из M\s
      {
        если s является префиксом s'
        {
          добавить данные s' в s
          выход из цикла
        }
      }

      если s восстановлен полностью
      {
        удалить s из M
        выполнить разбор s
      }

      выход из алгоритма
    }
  }

  создать блок-поток s
```

```
  добавить данные f в s
  добавить s в M
}


выход из алгоритма
```

Рис. 16 – Формальное описание метода восстановления потоков.

При использовании такого алгоритма не возникает неопределенности, к какой единице передачи данных (PDU) следует отнести пакет, полученный в неправильном порядке: решение этого вопроса откладывается до тех пор, пока не будет обработан непосредственно предшествующий пакет, или выяснится, что этот пакет в трафике отсутствует. Предложенный подход требует большего количества физической памяти по сравнению с попыткой определить границы PDU сразу (в момент разбора пакета), однако результат анализа становится более точным.

2.4 Выводы

Представление разобранного сетевого пакета в виде дерева блоков, а также механизм распознавателей позволяют анализировать трафик вложенных туннелей любой конфигурации. Благодаря замещающим блокам-фрагментам описываются модификации (дешифрование, распаковка) данных. Введенные понятия контекста и ключевой группы позволяют единым образом описывать разбор пакетов протоколов с сохранением состояния, а также выделять множества пакетов для последующего восстановления потоков. Разработанный алгоритм восстановления потоков данных устойчив к потере и переупорядочиванию отдельных пакетов.

3. Архитектура системы анализа

С учетом сформулированных в подразделе 1.2 требований была разработана архитектура системы, позволяющая обходиться единым комплектом исходных кодов разборщиков протоколов, применяя их как в online-, так и в offline- режиме. Предложенная модель представления данных реализована в виде ядра системы, которое компилируется в динамическую библиотеку. На базе API ядра построена работа модулей разбора и распознавания данных. Было разработано два инструмента – для проведения online- и offline-анализа соответственно, – причем оба используют единую инфраструктуру (рис. 17). В зависимости от установленных параметров проводится сборка ядра и модулей разбора либо для offline-анализатора, либо для online-анализатора. Построенная по такому принципу система позволяет в полной мере использовать преимущества offline-анализа при разработке (отладке) модулей разбора и распознавания и дальнейшей эксплуатации этих модулей (используется тот же исходный код) при работе online.

Главное отличие между offline- и online-версиями ядра заключается в механизме управления памятью: при работе на бесконечном потоке данных необходимо гарантировать своевременное освобождение памяти. Инструмент анализа сетевых трасс, в свою очередь, нуждается в развитом графическом интерфейсе, возможности которого нацелены на автоматизацию действий пользователя при решении конкретных классов задач, таких как: локализация и детализация сетевых соединений, обратная инженерия протоколов (включая отладку разборщиков), интерактивное управление порядком разбора пакетов и др.

Система является модульной, как и большинство анализаторов сетевого трафика: для каждого протокола создается отдельный модуль, в котором локализована функциональность по работе с этим протоколом. Каждый из инструментов использует свой комплект модулей разбора. Важно, что при компиляции разборщиков для разных подсистем (инструментов) используется один и тот же исходный код. В рамках разработанной системы модули разбора также являются

независимыми: при добавлении новых разборщиков не требуется вносить изменения в существующие. Такая независимость достигается благодаря механизму связывания разборщиков, основанному на применении распознавателей.

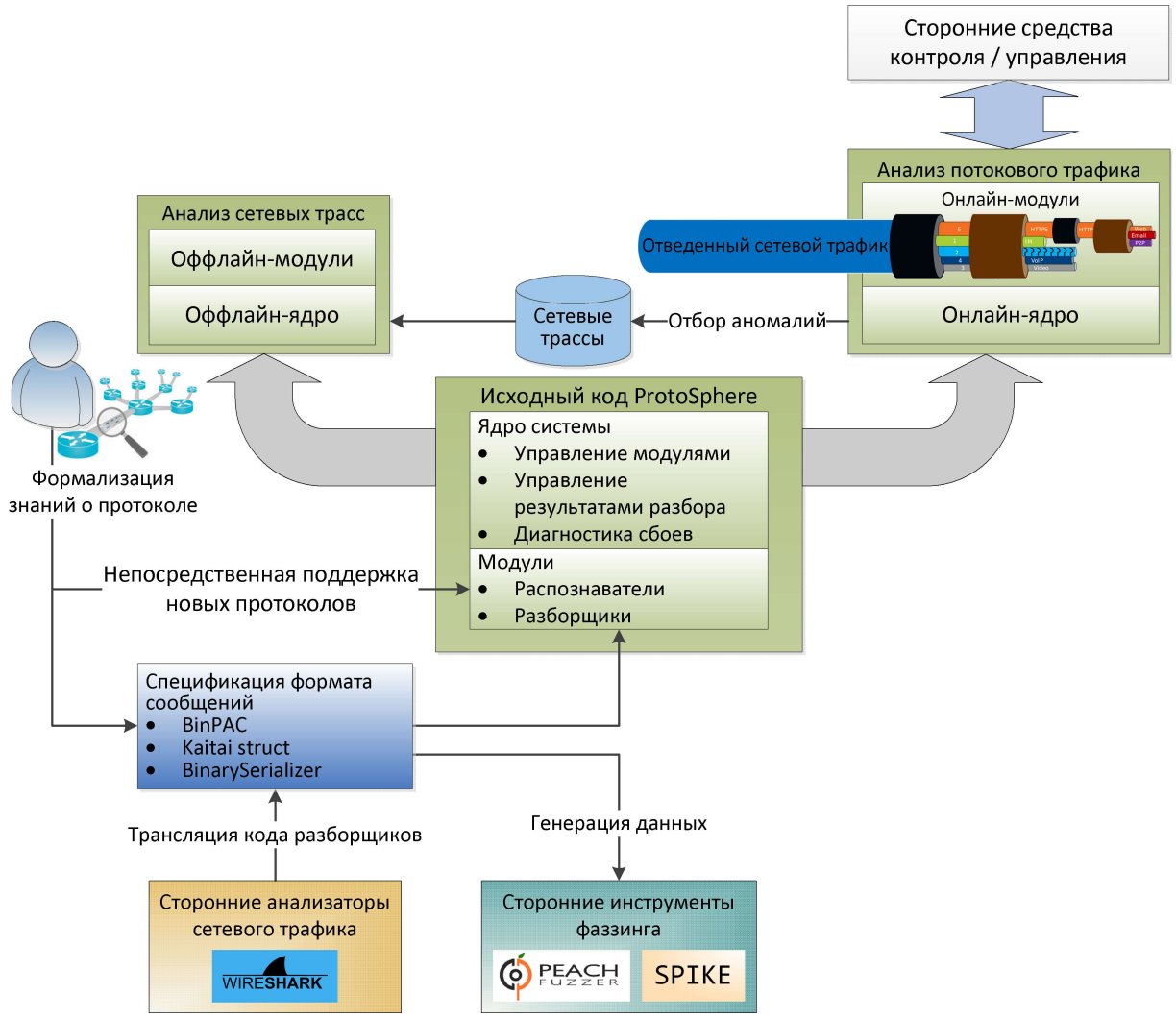

Рис. 17 – Структурные компоненты системы.

База поддерживаемых протоколов расширяется не только посредством ручной разработкой кода разборщиков, но и путем автоматизированного переноса (портирования) разборщиков системы Wireshark. Для функций API предложенной системы и системы Wireshark задается отображение, исходный код транслируется в промежуточное представление, на основе которого генерируется перенесенный код. Инструмент портирования детально описывается в подразделе 3.3.

3.1 Процедура анализа

Анализ состоит из последовательного выполнения операций распознавания и разбора данных. Для некоторых протоколов в соответствии с их семантикой проводится восстановление потоков, данные которых в дальнейшем также разбираются. Таким образом, обеспечивается последовательное повышения уровня представления данных (рис. 18). В итоге пользователь оперирует над высокоуровневыми данными, что потенциально повышает качество проводимого им анализа.

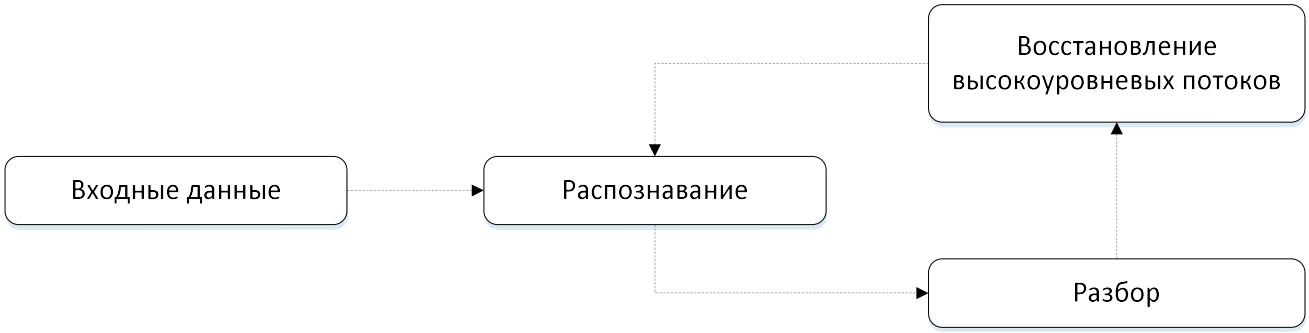

Рис. 18 – Последовательное повышение уровня представления данных.

На уровне ядра состояние разбора описывается посредством трех стеков:

- стек блоков;
- стек контекстов (элемент – пара контекст и ключевая группа);
- стек сетевых узлов (элемент – пара из отправителя и получателя).

На рис. 19 упрощенно приведен алгоритм разбора блока ядром системы.

```
 1. Добавить блок B в стек блоков
 2. Определить тип разбора T для B
 3. Если для T выставлен флаг ContextExistence, то
 4.       Добавить контекст C в стек контекстов
 5. Если для T предусмотрена лог-тема L, то
 6.       Добавить тему L в стек тем журнала
 7. Вызвать разборщик для данных B в соответствии с T
 8. Если для T предусмотрена лог-тема L, то
 9.       Вытолкнуть тему из стека тем журнала
10. Если для T выставлен флаг ContextExistence, то
11.       Вытолкнуть контекст из стека контекстов
```

```
12. Вытолкнуть блок из стека блоков
```

Рис. 19 – Алгоритм разбора блока ядром системы.

Добавление контекста (в строке 4) подразумевает, что в родительской ключевой группе контекст с типом ***T*** существует: если нет, то он должен быть создан.

Стек блоков применяется для построения дерева разбора: вершина стека становится родителем для блоков, созданных разборщиком.

Распознаватели применяются последовательно до первого успеха. В зависимости от того, какой блок анализируется, применяются разные группы распознавателей:

- Блок-поток:
  - распознаватели потоков с известным типом сборки;
  - универсальные распознаватели;
- Составной или одиночный блок:
  - распознаватели полей;
  - универсальные распознаватели.

Посредством стека контекстов осуществляется группировка блоков (строится дерево контекстов): вершина стека характеризует протокол, заголовок которого разбирается в настоящий момент. Будем называть контекст и ключевую группу в вершине стека *активными*. Контекст помещается в стек непосредственно перед разбором блока при условии, что тип разбора обладает контекстом (выставлен флаг *ContextExistence*), и выталкивается из стека по окончании разбора. Перед началом анализа создаются и помещаются в стек фиктивные контекст с неопределенным типом и ключевая группа с пустым ключом: благодаря этому в каждый момент времени определены активные контекст и ключевая группа. Фиктивная ключевая группа становится родительской по отношению к первому настоящему контексту. Перед добавлением контекста в стек проверяется, существует ли в активной ключевой группе дочерний контекст соответствующего типа. Если да, то новый контекст не создается, а в стек помещается уже существующий. При создании контекста проводится инициализация *расширения* – специализированных атрибутов, описывающих состояние соединения. В

расширении контекста протокола SSL, в частности, хранится ключ шифрования, загруженный в систему пользователем с целью дальнейшего использования.

Если контекст помещается в стек перед разбором блока, то ключевая группа определяется уже в процессе разбора. Исключение составляет случай с ключом нулевого размера *GroupKeySize*: в рамках контекста может быть сформирована только одна группа (в частности, нулевой размер ключа зафиксирован для протокола HTTP).

Стек сетевых узлов используется для построения NetworkTree. Если протокол предполагает наличие отправителя и получателя, то при разборе заголовка на стек помещаются нужные сетевые узлы.

Журнал предназначен для регистрации ошибок разбора. Каждая запись в нем характеризуется меткой времени, текстовым сообщением, а также блоком, разбор которого привел к возникновению этой записи. Записи журнала также группируются по *темам*: привязка к теме осуществляется посредством типа разбора блока. В журнале поддерживается стек тем: перед началом анализа в него помещается тема по умолчанию. Если тип разбора блока предполагает наличие темы, то соответствующая тема помещается в стек непосредственно перед его разбором и выталкивается из стека после разбора. Если тема для типа не определена, соответствующий записи ассоциируются с темой, находящейся на вершине стека. Как правило, одна тема задается для каждого модуля разбора.

Анализ файла с трафиком завершается, когда все блоки-потоки разобраны. В режиме online анализ прекращается по команде пользователя.

3.2 API ядра

API системы позволяет аналитику регистрировать типы (описывать разборщики) и распознаватели, создавать блоки-потоки, одиночные и составные фрагменты, управлять восстановление потоков. В частности, для корректного восстановления потока из переупорядоченных пакетов предусмотрена функция объединения двух блоков-потоков в один.

3.2.1 Разбор

В табл. 8 приведены основные функции, используемые при разработке разборщиков. Отметим, что модули разбора полностью независимы между собой.

Табл. 8 – API для разработки модулей разбора.

Имя функции	Описание
Операции над блоками	
processSingle	Создать и разобрать single-блок
processComposite	Создать и разобрать composite-блок
createStream	Создать блок-поток
completeStream	Выполнить разбор блока-потока
streamAppend	Добавить данные блока в блок-поток
Управление состоянием разбора	
contextExtension	Получить данные расширения активного контекста
activateKeyGroup	Активировать в рамках текущего контекста группу, определяемую заданным ключом
keyGroupExtension	Получить данные расширения активной ключевой группы
setSrcDst	Активировать отправителя и получателя
Регистрация разборщиков и распознавателей	
regType	Зарегистрировать тип
regRecognizer	Зарегистрировать распознаватель
getType	Получить тип, зарегистрированный в другом модуле
Журнал сообщений	
log	Записать сообщение в журнал

Ключ для формирования группы строится при разборе заголовка: для активации группы применяется функция *activateKeyGroup*. В процессе разбора используются и изменяются данные расширений активных контекста (*функция contextExtension*) и группы (*функция keyGroupExtension*). Предусмотрено добавление данных блоков-потоков, одиночных и составных фрагментов в потоки (*функция streamAppend*).

При регистрации распознавателя поля необходимо указать имя соответствующего поля, а также тип разбора блока-родителя. Таким образом, распознаватели, локализованные в разных модулях, могут относиться к одному и тому же полю. Благодаря такому способу привязки появляется возможность расширения функциональности без внесения изменений в уже существующий код модулей разбора.

3.2.2 Представление в заданном формате результатов анализа трафика

Интерфейс построения (табл. 9) предназначен для сохранения декодированных данных в формате, определяемом пользователем. Формат задается путем создания модуля построения. Обычно такие модули применяются при проведении online-анализа. В частности, могут быть сохранены восстановленные PNG-, JPEG-, HTML-файлы, HTTP-потоки и т.д. Поддерживается два способа привязки модулей построения:

- по типу разбора блока;
- по стеку сетевых узлов.

При регистрации модуль построения сообщает ядру системы тип блоков, которые следует ему передавать. В случае привязки по стеку сетевых узлов вместе с типом блока проверяется содержимое стека отправителей и получателей: только при совпадении блок будет отправлен в соответствующий модуль.

Табл. 9 – API для разработки модулей построения.

Имя функции	Описание
createBuffer	Создать буфер
completeBuffer	Сохранить буфер в файл
bufferAppend	Добавить данные блока в буфер в соответствии с заданным форматом

3.3 Инструмент портирования разборщиков из Wireshark

Для увеличения базы поддерживаемых протоколов системы был разработан инструмент портирования разборщиков анализатора Wireshark. Процесс переноса модуля разбора состоит из трех этапов:

1. трансляция исходного кода разборщика в промежуточное представление
2. модификация промежуточного представления;
3. генерация кода на основе модифицированного промежуточного представления.

Код модулей разбора Wireshark написан на языке СИ. Разработка разборщиков, используемых описываемой системой, ведется в рамках языка СИ++. Поскольку

языки СИ и СИ++ во многом похожи, в качестве промежуточного представления, используемого при портировании, было выбрано абстрактное синтаксическое дерево (АСД). АСД строится посредством инструмента Elsa, являющегося частью системы статического анализа Oink [45]. Выбор инструмента обусловлен простотой его применения, а также наличием у него программного интерфейса на языке СИ++.

Первый этап состоит в построении АСД по исходному коду модуля разбора Wireshark. На втором этапе проводится модификация построенного АСД:

1. замена используемых типов и имён типов, если они различны;
2. выделение части АСД, соответствующей коду модуля (отделение части АСД, соответствующей включаемым файлам);
3. сбор информации, необходимой для проведения следующих этапов;
4. замена вызовов функций одного интерфейса на вызовы функций другого интерфейса, замена переменных и параметров функций;
5. удаление ненужных инструкций, проверка на нереализованные преобразования.

Третий этап состоит в генерации целевого кода по АСД. При этом, некоторые участки кода (в частности, специфичные для целевой системы функции) могут быть сгенерированы непосредственно, без добавления в АСД.

Перед проведением трансляции код модуля подвергается предобработке с целью сохранения информации о типах (разбора). Граница между включаемыми заголовочными файлами библиотеки Wireshark и кодом самого модуля помечается специальной pragma-директивой для того, чтобы в дальнейшем отделить код включаемых файлов от кода модуля на уровне АСД. Затем код модуля обрабатывается стандартным системным препроцессором, после чего с помощью инструмента Elsa проводится трансляция препроцессированного кода в АСД.

Для сбора информации о типах полей разборщика Wireshark используется список инициализации массива структур типа `hf_register_info` в АСД.

Для получения информации о регистрируемых протоколах анализируются функции *proto_register_protoname* и *proto_reg_handoff_protoname*, где *protoname* – название протокола. Локализуются вызовы следующих функций:

1. proto_register_protocol – регистрация протокола;
2. create_dissector_handle – создание описателя разборщика;
3. register_dissector – присвоение имени разборщику;
4. find_dissector – возврат описателя по имени;
5. dissector_add_uint – связывание разборщика в таблице привязки.

Далее проводится замена вызовов функций ядра разбора Wireshark на вызовы функций ядра разбора разработанной системы в функциях, производящих разбор.

Разборщики в Wireshark принимают три или четыре параметра:

- указатель на объект буфера;
- указатель на структуру packet_info, содержащую значения набора служебных полей, выделенных в заголовках этого пакета ранее;
- указатель на поддерево дерева разбора, в которое разборщик будет добавлять данные;
- Необязательный пользовательский параметр: его значение определяется соглашением между вызывающим и вызываемым разборщиками.

Функция-разборщик разработанной системы принимает 2 параметра: указатель на буфер памяти и размер этого буфера.

Заменяемые функции можно разделить на группы:

1. функции обращения к данным;
2. функции добавления полей в дерево разбора;
3. функции разбора данных, передаваемых посредством TCP-сегментов.

Функции обращения к данным преобразуются в прямое обращение к данным с использованием функций для преобразования порядка байтов.

Функции добавления полей в дерево разбора заменяются функциями обработки простого или составного блока-фрагмента с соответствующим типом разбора.

Функция tcp_dissect_pdus предназначена для разделения TCP-потока на сообщения, границы которых могут не совпадать с границами сегментов. Так как в разработанной системе сборка TCP-потоков проводится модулем TCP, то в разборщиках пакетов вышележащих протоколов необходимо реализовать лишь разбиение данных потока на сообщения. Поэтому вызов функции tcp_dissect_pdus заменяется последовательностью инструкций, производящих это разбиение.

Новый исходный код генерируется по АСД с помощью инструмента Elsa.

4. Разработанные анализаторы

4.1 Анализ трафика в режиме online

Инструмент online-анализа предназначен для извлечения данных из трафика в режиме реального времени: он должен работать непрерывно с производительностью, достаточной для разбора пакетов, поступающих на сетевой интерфейс (потенциально бесконечный входной поток данных).

Инструмент состоит из трех компонентов (рис. 20):

- модуль *listener* осуществляет взаимодействие с сетевым интерфейсом и сохраняет поступающие пакеты в кольцевой буфер;
- модуль *kernel* выполняет разбор пакетов кольцевого буфера и извлекает необходимые данные;
- модуль *saver* сохраняет извлеченные данные в файлы на жестком диске (формат файла определяется модулем построения).

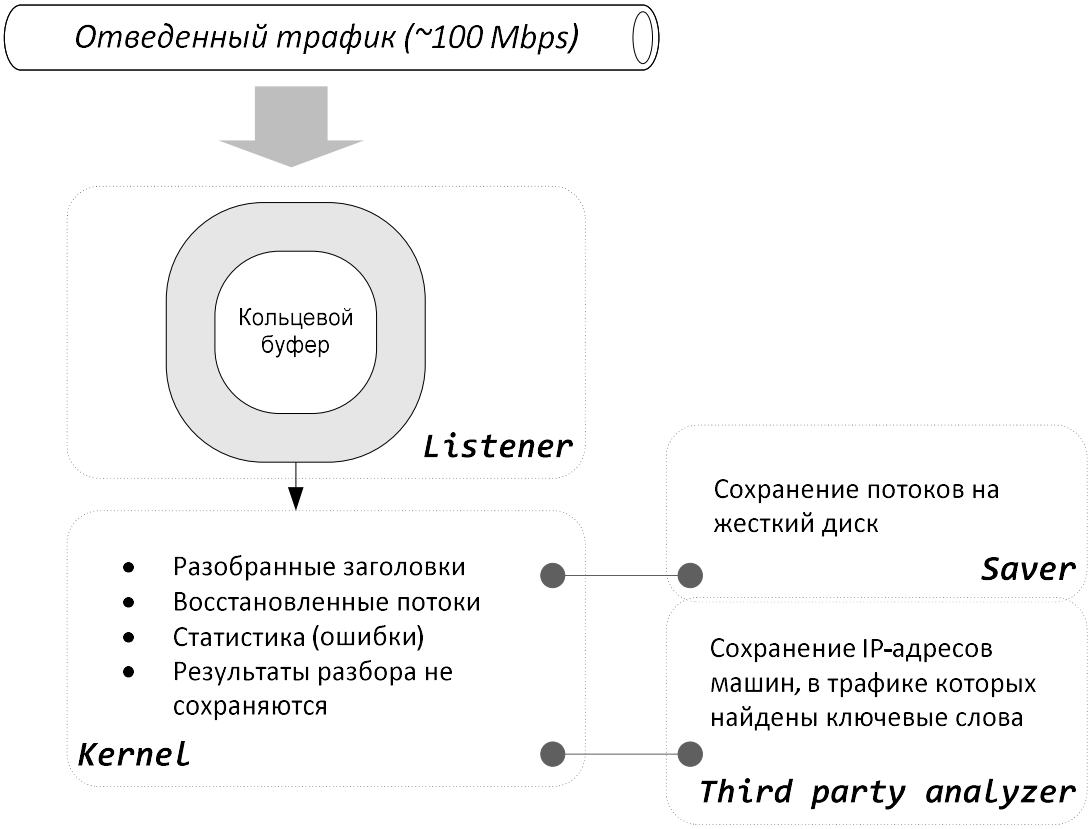

Рис. 20 – Компоненты online-анализатора.

В настоящий момент компонент *listener* может получать пакеты либо с помощью API библиотеки PCAP [31], либо посредством ZeroMQ-сокета [46]. Пакеты последовательно записываются в кольцевой буфер *RingBuffer*, после чего анализируются компонентом *kernel*.

При разборе потенциально бесконечного потока данных необходимо явно ограничивать размер доступной анализатору физической памяти: в противном случае свободная память закончится, и анализатор в лучшем случае завершит работу аварийно. *RingBuffer* занимает фиксированный объем в физической памяти, вычисляемый перед началом проведения анализа. В процессе анализа память используется для хранения потоков, восстановление которых еще на завершилось, и, как следствие, дерева контекстов. Количество таких потоков и вершин в дереве контекстов может неограниченно расти: необходимо введение пороговых значений. Предложена следующая стратегия освобождения памяти: при достижении порогового значения удалять первыми объекты (потоки, ключевые группы, контексты), которые дольше других не обновлялись. Такая стратегия реализована посредством разработанного *generic*-контейнера *PriorityStorage*.

Online-анализатор отдельно сохраняет в файл пакеты, разбор которых происходит с ошибками. Под *ошибкой* в данном случае понимается несоответствие фактических данных описанию протокола в модуле разбора. Если количество таких ошибок превышает заданное пороговое значение, велика вероятность того, что соответствующий модуль разбора содержит неточности. В таком случае необходимо проанализировать сохраненные пакеты с помощью offline-анализатора, после чего внести в код модуля разбора (общий для двух инструментов) необходимые правки.

Инструмент online-анализа может использоваться сторонними инструментами для получения данных в заданном формате (см. интерфейс построения). Предусмотрен интерфейс получения результатов разбора через ZMQ-сокет. Результаты разбора сетевых пакетов могут использоваться в различных целях, в

частности, для проверки выполнимости правил, описывающих политику безопасности.

4.1.1 Управление ресурсами

При проведении анализа в режиме online происходит постепенное увеличение числа контекстов, ключевых групп, потоков. Чтобы не выходить за рамки физической памяти, отведенной анализатору, необходимо перераспределять ее между объектами, которые с большой вероятностью больше не потребуются (могут быть удалены), и объектами, с которыми еще предстоит работать (удаление нежелательно). Для этого используется расширенная очередь с приоритетом и фиксированной вместимостью *PriorityStorage*.

4.1.1.1 Расширенная очередь с приоритетом: PriorityStorage

PriorityStorage фактически представляет собой кэш с возможностью выполнения некоторой функции над элементом непосредственно перед его удалением: выполнить разбор потока, удалить дочерние ключевые группы контекста. Элементы очереди упорядочены по времени *обновления*: обновление элемента подразумевает максимизацию его приоритета. Для контекста и ключевой группы обновление происходит при активации (т.е. всякий раз, когда они оказываются на вершине стека контекстов). Обновление потока подразумевает либо его попадание на вершину стека блоков, либо добавление в его буфер новой порции данных.

Вместимость (*capacity*) очереди задается при создании, однако может быть изменена, когда контейнер пуст. Следует отличать от вместимости размер очереди, равный количеству элементов в ней. При этом размер очереди не превосходит ее вместимости.

Все элементы в контейнере различны, дубликаты не допускаются. Наибольшим приоритетом обладает последний добавленный в контейнер элемент или элемент, для которого выполнена операция максимизации приоритета. Поддерживаются следующие операции:

- добавление элемента (по значению);

- максимизация приоритета элемента (по значению);
- получение/извлечение элемента с минимальным приоритетом;
- извлечение произвольного элемента (по значению).

Контейнер реализован с использованием словаря $\boldsymbol{M}$ и двусвязного списка $\boldsymbol{L}$: $\boldsymbol{M}$ используется для быстрого доступа к элементам, $\boldsymbol{L}$ – для поддержания упорядоченности в соответствии с обновлениями. $\boldsymbol{M}$ задает отображение множества элементов контейнера на множество элементов $\boldsymbol{L}$. Каждый элемент $\boldsymbol{L}$ хранит информацию о предыдущем и следующем (по приоритету) элементах $\boldsymbol{L}$, а также копию элемента контейнера (элементы контейнера должны допускать копирование). В PriorityStorage также содержится информация о первом и последнем элементах $\boldsymbol{L}$.

Определим вычислительную сложность основных операций. Будем считать, что сравнение и копирование элементов контейнера выполняются за константное время.

Чтобы добавить элемент $\boldsymbol{e}$ в PriorityStorage, нужно:

1. добавить новый элемент $\boldsymbol{l}$ в конец списка $\boldsymbol{L}$ (в том числе сделать копию добавляемого элемента $\boldsymbol{e}$): $O(1)$;
2. обновить информацию об элементе $\boldsymbol{L}$ с максимальным приоритетом: $O(1)$;
3. Добавить пару $(\boldsymbol{e}, \boldsymbol{l})$ в словарь $\boldsymbol{M}$: $O(\log n)$.

Таким образом, алгоритмическая сложность добавления составляет $O(\log n)$, где n – размер очереди.

При выполнении максимизации приоритета элемента $\boldsymbol{e}$ необходимо:

1. получить элемент $\boldsymbol{l}$ списка $\boldsymbol{L}$ по ключу $\boldsymbol{e}$ в словаре $\boldsymbol{M}$: $O(\log n)$;
2. обновить информацию об элементе $\boldsymbol{L}$ с максимальным приоритетом: $O(1)$.

Итоговая сложность: $O(\log n)$.

Получение элемента с минимальным приоритетом выполняется за константное время, извлечение – за $O(\log n)$. Извлечение произвольного элемента также выполняется за логарифмическое время.

4.1.1.2 Кольцевой буфер: RingBuffer

Кольцевой буфер RingBuffer организован в виде списка блоков памяти фиксированного размера. В каждый момент времени два блока в буфере помечены: *writable* – блок для записи и *readable* – блок для чтения. Данные (сетевые пакеты) записываются в *writable*-блок последовательно до тех пор, пока в нем достаточно свободного места. Когда место заканчивается, соответствующая пометка переносится на следующий по списку блок. С течением времени все блоки будут исчерпаны и начнется перезапись. Будем называть *поколением* блока памяти количество перезаписей его нулевого байта: с каждой перезаписью поколение инкрементируется. При записи данных в буфер запоминается поколение *writable*-блока: если при последующем чтении выяснится, что его поколение изменилось, данные будут проигнорированы. Так осуществляется контроль за целостностью обрабатываемых данных.

Операции чтения и записи (более точно, перенос пометок) являются потокобезопасными. Если требуется прочитать данные *writable*-блока, то очередная запись данных независимо от наличия свободного места будет произведена в следующий по списку блок памяти.

В режиме реального времени важна высокая скорость обработки: если сетевые пакеты поступают быстрее, чем обрабатываются, то проводить анализ бессмысленно. Операция копирования данных занимает значительное время, поэтому при чтении из буфера не выполняется – вместо этого блок памяти, содержащий нужный пакет, блокируется на чтение. Для очередной записи будет выбран следующий по списку блок.

4.2 Offline-анализ сетевых трасс

Инструмент offline-анализа (рис. 21) главным образом предназначен для разработки и отладки модулей разбора: с его помощью аналитик получает максимально детализированную "картину происходящего" в сетевой трассе с указанием возникших в процессе разбора ошибок. Инструмент обладает графическим интерфейсом, позволяющим проследить за тем, как происходит восстановление потоков высокоуровневых данных (например, файлов формата

PNG или PDF) из сетевых пакетов. Также предусмотрены механизмы поиска по трафику и навигации между различными представлениями результатов разбора.

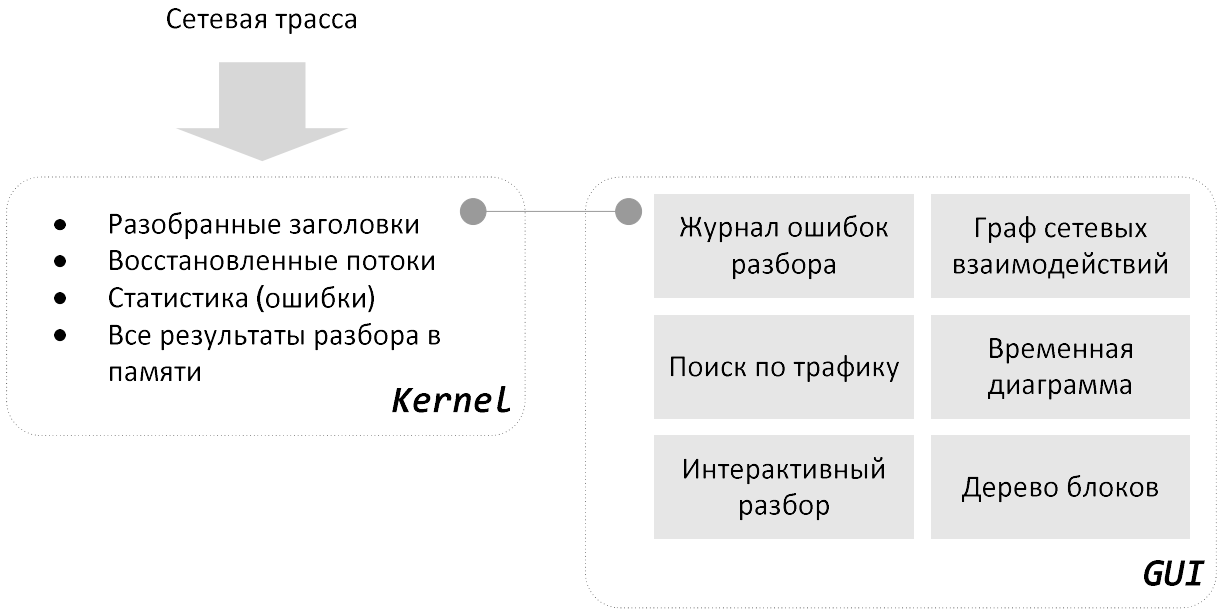

Рис. 21 – Компоненты offline-анализатора.

Центральное место в инструменте offline-анализа занимают компоненты отображения результатов разбора:

- дерево блоков;
- журнал ошибок разбора;
- граф сетевых взаимодействий;
- временная диаграмма.

4.2.1 Представление результатов разбора

4.2.1.1 Дерево блоков

Дерево блоков представляет собой результат разбора одного блока-потока: корень описывает блок-поток, остальные вершины – выделенные блоки-фрагменты (сетевые пакеты и поля в них). Дерево блоков синхронизировано с компонентом отображения буфера данных: при выделении блока в дереве соответствующие ему байты подсвечиваются. Для блока-потока можно просмотреть список блоков, данные которых сформировали его накопительный буфер. При этом блоки, порядок которых был изменен при передаче, будут помечены (рис. 22).

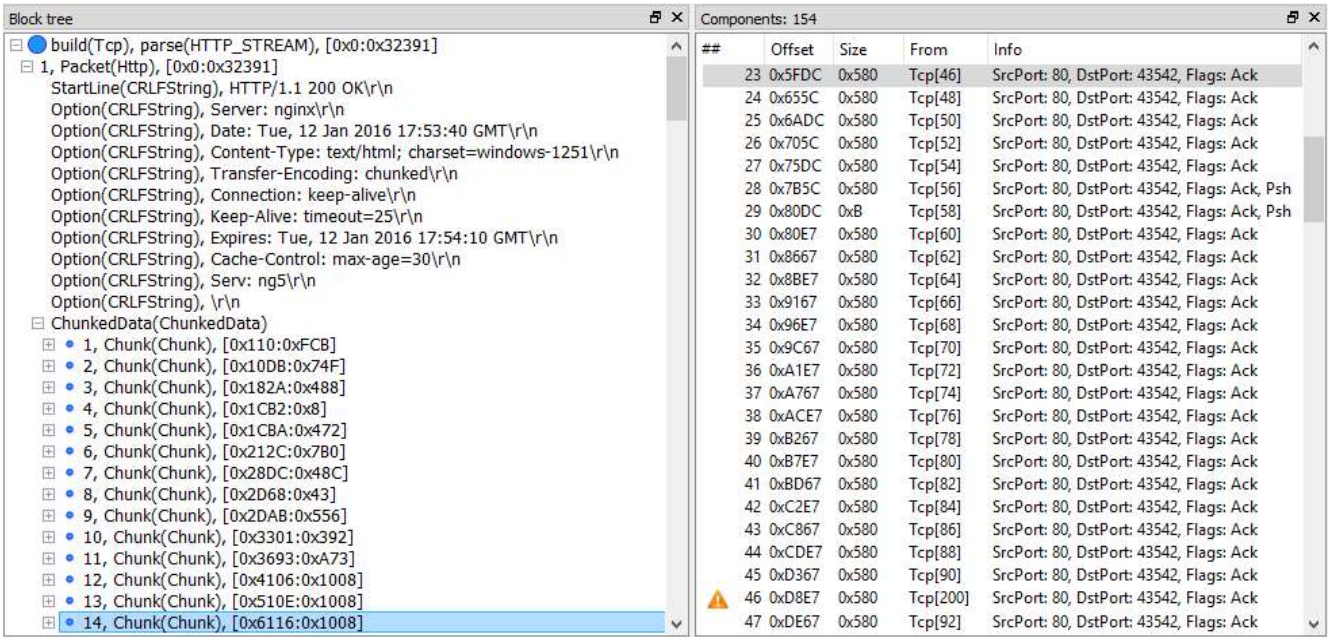

Рис. 22 – Отображение дерева блоков (слева) и списка компонентов (справа) для блока-потока.

Вспомним, что основным средством представления сетевой трассы в Wireshark являются разобранные пакеты в виде списка: при этом только для одного выделенного пакета отображается полный стек протоколов и значения полей в заголовках этих протоколов. Пакет, как элемент списка, представляется посредством строки, состоящей из значений фиксированного набора полей, выделенных в заголовке протокола сетевого уровня (IP-адреса), а также полей заголовка протокола самого высокого уровня, который удалось разобрать. Фиксированность представления пакетов может вызывать трудности во многих случаях, например, при анализе туннельных протоколов. При инкапсуляции вида IP-in-IP каждый сетевой пакет будет содержать по два заголовка протокола IP. В списке пакетов инструмент Wireshark отобразит поля последнего (верхнего) IP-заголовка, который с точки зрения туннельного соединения не является репрезентативным (рис. 23): чтобы понять, какой хост из внутренней сети инициировал взаимодействие, аналитику придется выделять каждый пакет и проверять значение поля сетевого адреса нижележащего IP-заголовка. При использовании дерева блоков таких затруднений, очевидно, не возникает.

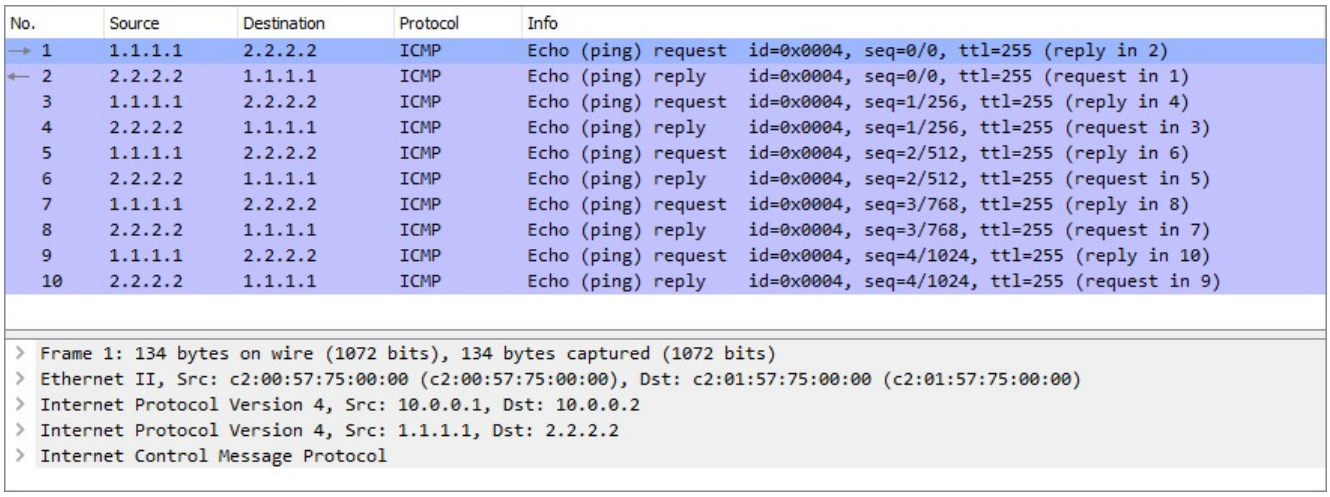

Рис. 23 – Отображение инкапсуляции IP-in-IP в Wireshark.

4.2.1.2 Журнал сообщений

Компонент отображения журнала синхронизирован с деревом блоков: это позволяет быстро локализовать блок, разбор данных которого привел к возникновению сообщения (рис. 24). Поддерживается фильтрация записей по потокам и темам сообщений.

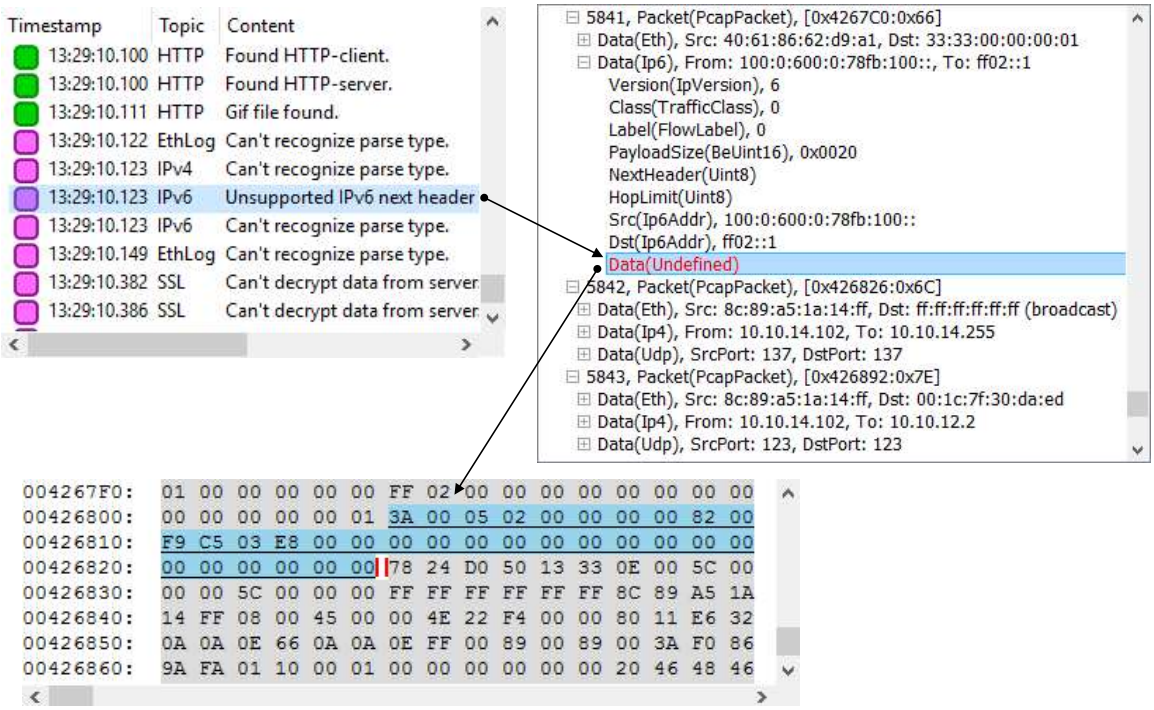

Рис. 24 – Отладка разборщика с использованием журнала.

4.2.1.3 Граф сетевых узлов

При расследовании инцидента нарушения информационной безопасности необходимо локализовать сетевые соединения, посредством которых этот

инцидент возник и развивался во времени: аналитик должен обладать некоторым критерием (или множеством таких критериев) на содержимое сетевых пакетов. Одним из подходов к решению задачи локализации является представление сетевых взаимодействий посредством графа, в котором вершинам соответствуют стороны сетевого взаимодействия, а ребра отображают факт взаимодействия и, возможно, некоторые его характеристики (например, такие как интенсивность. При этом одна и та же сторона может участвовать сразу в нескольких взаимодействиях. Далее требуется провести детальный анализ выделенных соединений:

- проследить за порядком отправки/получения пакетов;
- просмотреть значения интересующих полей.

Рассмотренные в главе 1 инструменты не предоставляют графических компонентов для работы с таким сценарием.

В инструменте offline-анализа реализовано два способа визуализации сетевых взаимодействий:

- граф оконечных узлов (*Endpoints*);
- граф, детализирующий сетевые взаимодействия выбранного оконечного узла (*Nodes*).

Граф *Endpoints* отображает сетевые соединения между узлами, характеризующимися одинаковыми цепочками протоколов: при этом можно выбрать протокол, взаимодействия в рамках которого будут показаны. Вершины графа соответствуют сетевым узлам. Имеются ребра двух видов (рис. 25): одни для представления стека нижележащих заголовков (пунктирные), другие для отображения сетевых взаимодействий между узлами (сплошные).

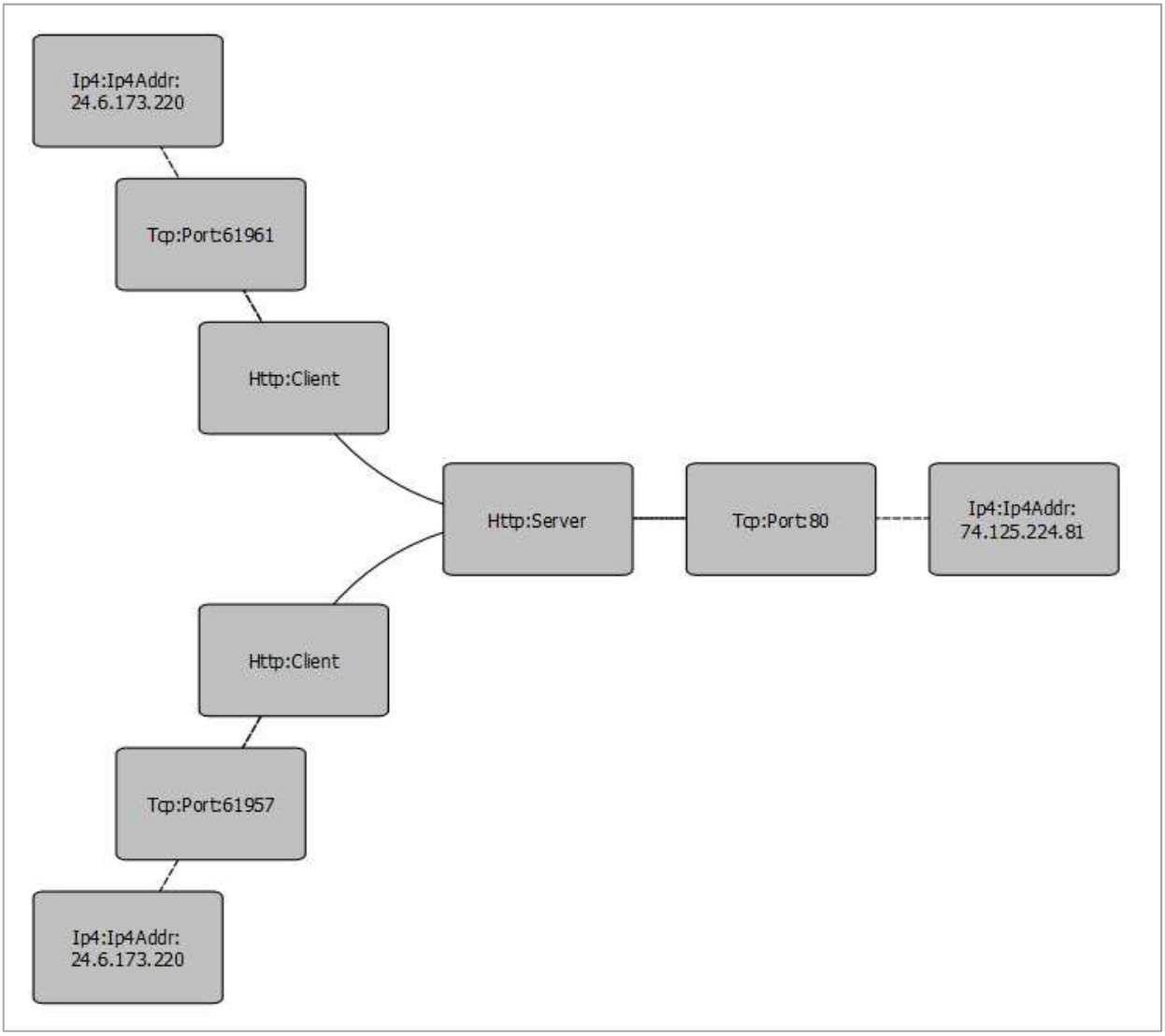

Рис. 25 – Пример графа *Endpoints*.

Граф *Nodes* детализирует взаимодействия выбранного на графе *Endpoints* оконечного узла: для каждого взаимодействия отображается весь стек сетевых протоколов (рис. 26). Предусмотрена возможность фильтрации графов по содержимому сетевых пакетов, в том числе с помощью регулярных выражений.

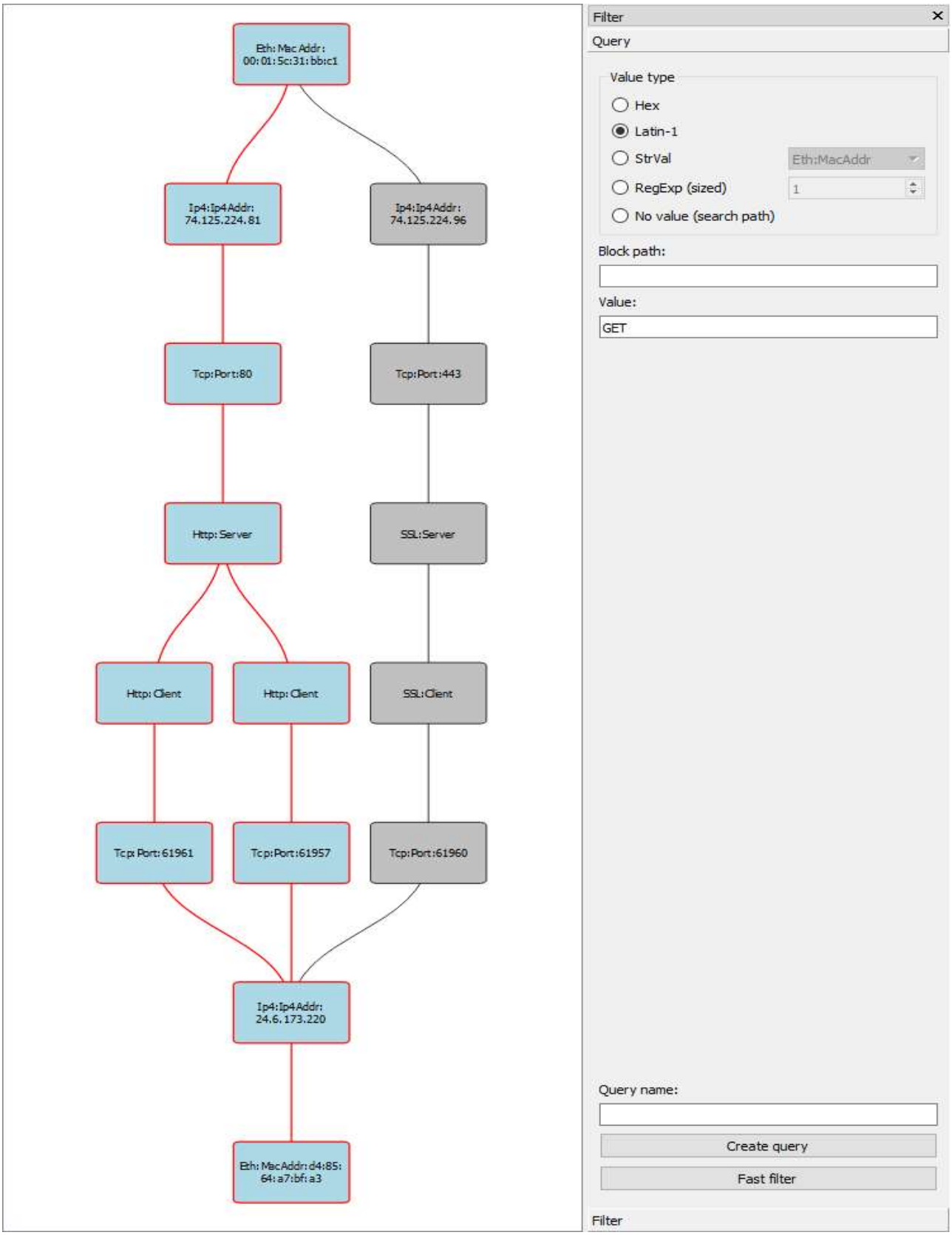

Рис. 26 – Пример применения фильтра к графу *Nodes*.

4.2.1.4 Временная диаграмма

Для проведения детального анализа отдельного сетевого взаимодействия предлагается временна́я диаграмма, где каждый пакет отображается в виде стрелки с указанием отправителя и получателя (рис. 27). При этом можно указать, какие заголовки протоколов и какие поля в них должны отображаться над стрелками: аналитик адаптирует графический компонент под свои нужды. Следует отметить, что в анализаторе Wireshark имеется аналогичный компонент [47], однако в нём отсутствует возможность настройки полей, значения которых будут отображаться для каждого пакета: это может вызывать затруднения, если интересующее поле отсутствует, или приводить к перегруженности отображения, если полей слишком много.

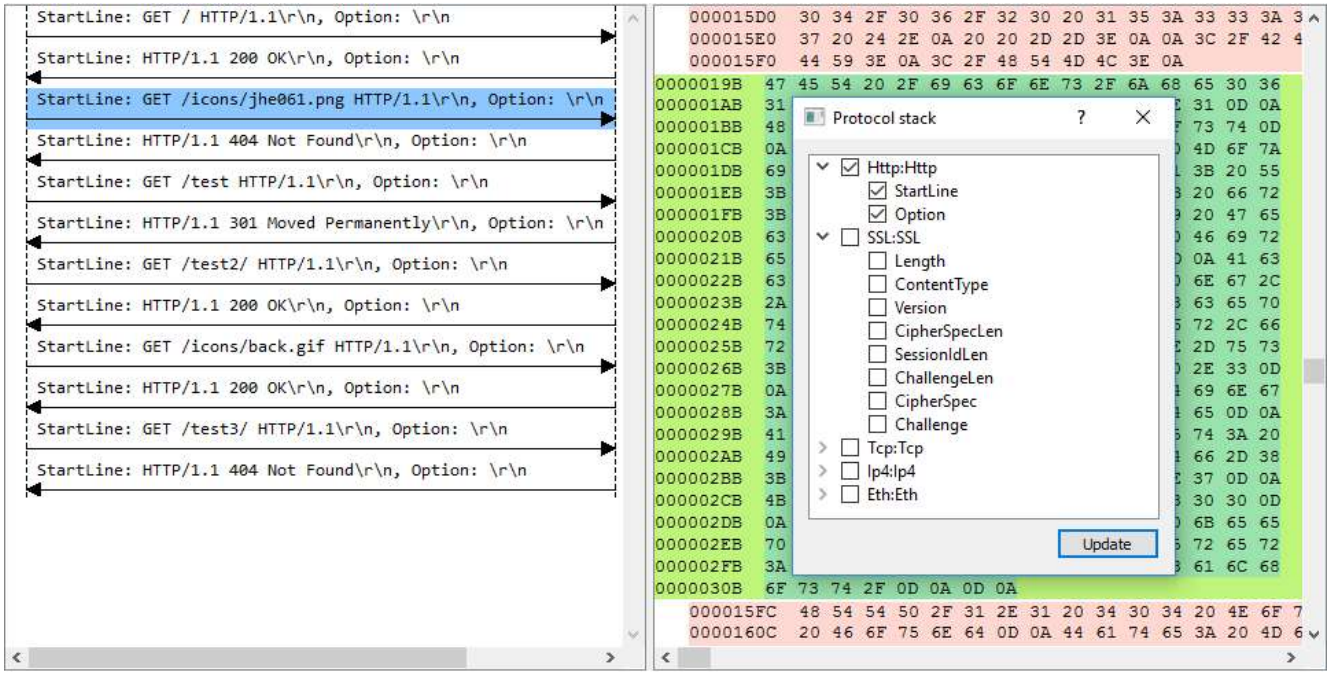

Рис. 27 – Выбор полей для отображения на временной диаграмме.

4.2.2 Интерактивный разбор

Тип разбора блока не всегда удается определить с помощью распознавателей: применяемые эвристики могут не учитывать всех возможных вариантов содержимого сетевого пакета (особенно, если для протокола отсутствует документация). Более того, система может определить тип разбора неверно, если критерий соответствия типу сформулирован недостаточно полно. В таких случаях аналитик может указать системе, как нужно разбирать тот или иной блок,

предоставив файл с подсказками. В нем для заданных блоков тип разбора определен непосредственно. Подсказка для определения типа применяется прежде, чем будут использованы соответствующие распознаватели.

С помощью подсказок реализована работа с зашифрованными данными: в подсказку записывается путь к файлу, содержащему необходимую для дешифрования информацию (как правило, ключ шифрования), которая используется при создании заданных контекста (*InitContext*) или ключевой группы (*InitKeyGroup*). Локализация контекста (ключевой группы) осуществляется посредством сохранения в подсказке содержимого стека контекстов. В процессе анализа ключ считывается из файла, данные блоков дешифруются, и разбор продолжается.

5. Практическое применение разработанной системы

5.1 Восстановление TCP-потока в случае переупорядочивания пакетов

Для трассы Trace14.pcap демонстрируется применение разработанного алгоритма восстановления потоков. Анализ показал, что к восстановленному TCP-потоку (#3) относится 154 TCP-пакета (рис. 28).

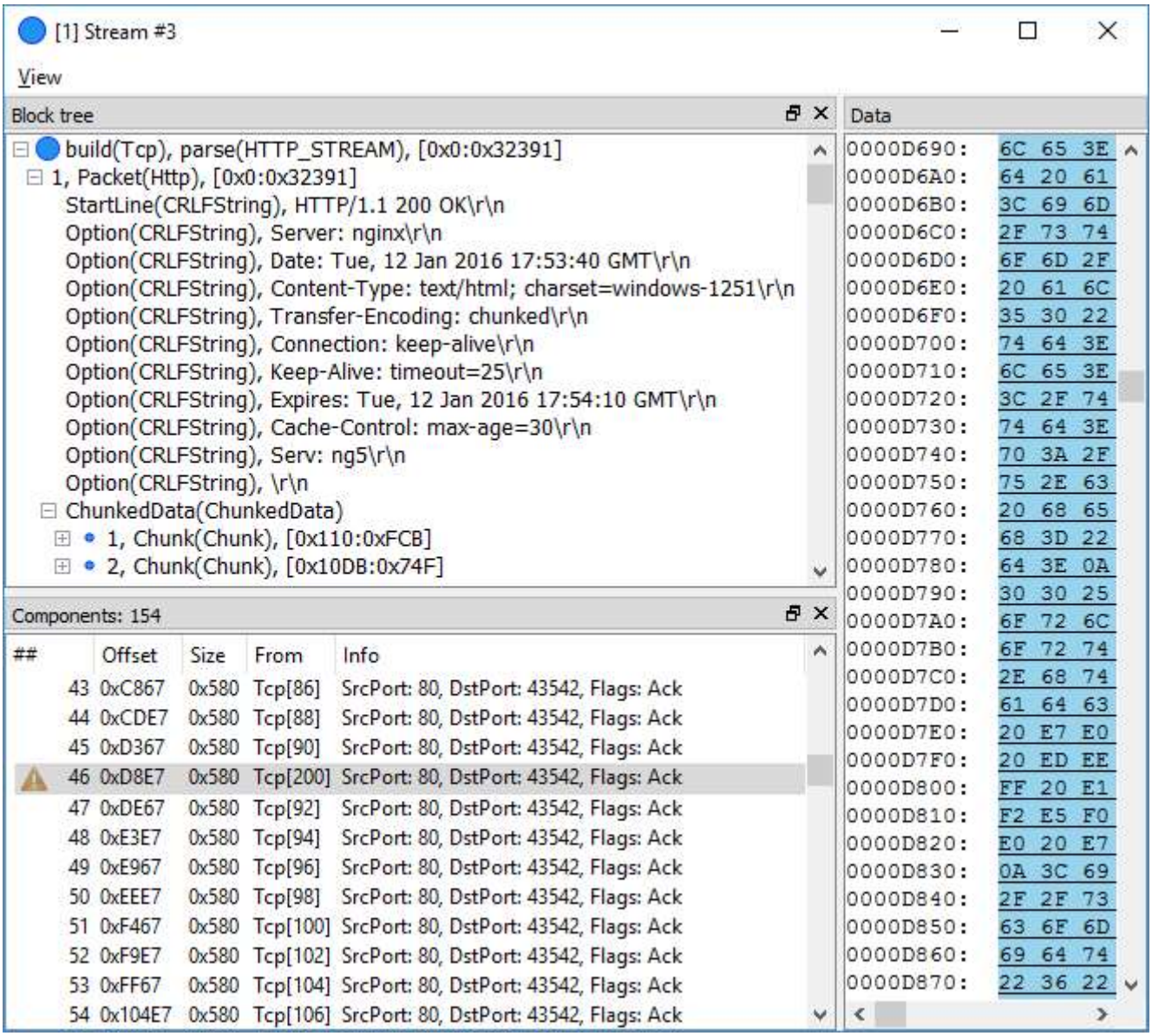

Рис. 28 – Анализ восстановления потока данных из переупорядоченных TCP-пакетов.

В первом столбце окна «Components» указаны порядковые номера, под которыми в восстановленный поток добавлены TCP-сегменты из пакетов с номерами, указанными в четвертом столбце (это номера пакетов в сетевой трассе): например,

полезная нагрузка 86-го TCP-пакета была добавлена под номером 43. Отметим, что TCP-пакеты с меньшими номерами в четвертом столбце разбираются раньше остальных.

Исходя из значения поля Sequence Number TCP-заголовка полезная нагрузка 92-го TCP-пакета не может быть добавлена в поток, частично восстановленный на момент окончания обработки пакета с номером 90. В результате, создается TCP-поток (#4), в который добавляются данные TCP-сегмента 92-го пакета. TCP-потоки #3 и #4 будут объединены в дальнейшем, после проведения разбора 200-го пакета.

Последующий разбор восстановленного TCP-потока позволил выделить HTTP-пакеты в нем (окно «Block tree») и извлечь HTML-страницу.

5.2 Анализ зашифрованного SSL-трафика

На примере трассы Trace31.pcap показана возможность анализа зашифрованных данных с использованием подсказки. Общий секретный ключ в данном случае генерируется с помощью алгоритма RSA [48]. Публичный ключ сервера извлекается клиентом из сертификата: для проведения дешифрования требуется секретный ключ, который считывается из файла с подсказками SSL- разборщиком при инициализации расширения соответствующего контекста. На рис. 29 и 30 показаны деревья контекстов, полученные соответственно без использования и с использованием подсказки, содержащей секретный ключ сервера. Во втором случае дешифрованные данные объединяются в HTTP-потоки, после чего проводится их разбор.

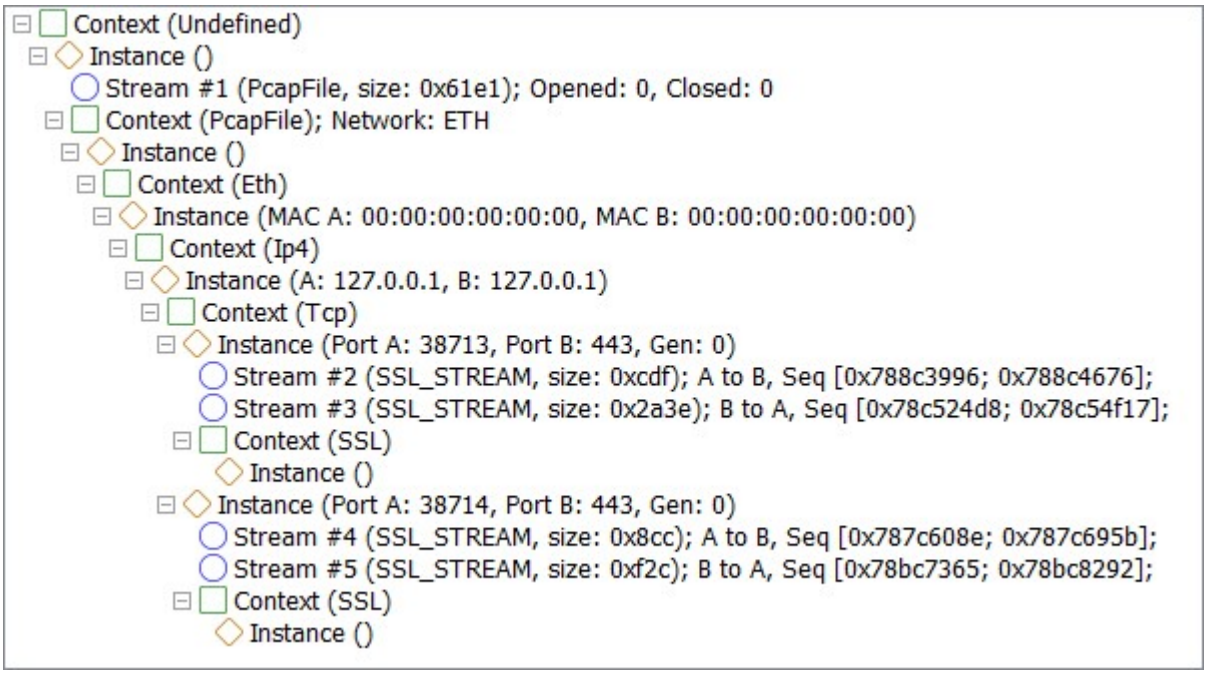

Рис. 29 – Дерево контекстов без использования подсказки.

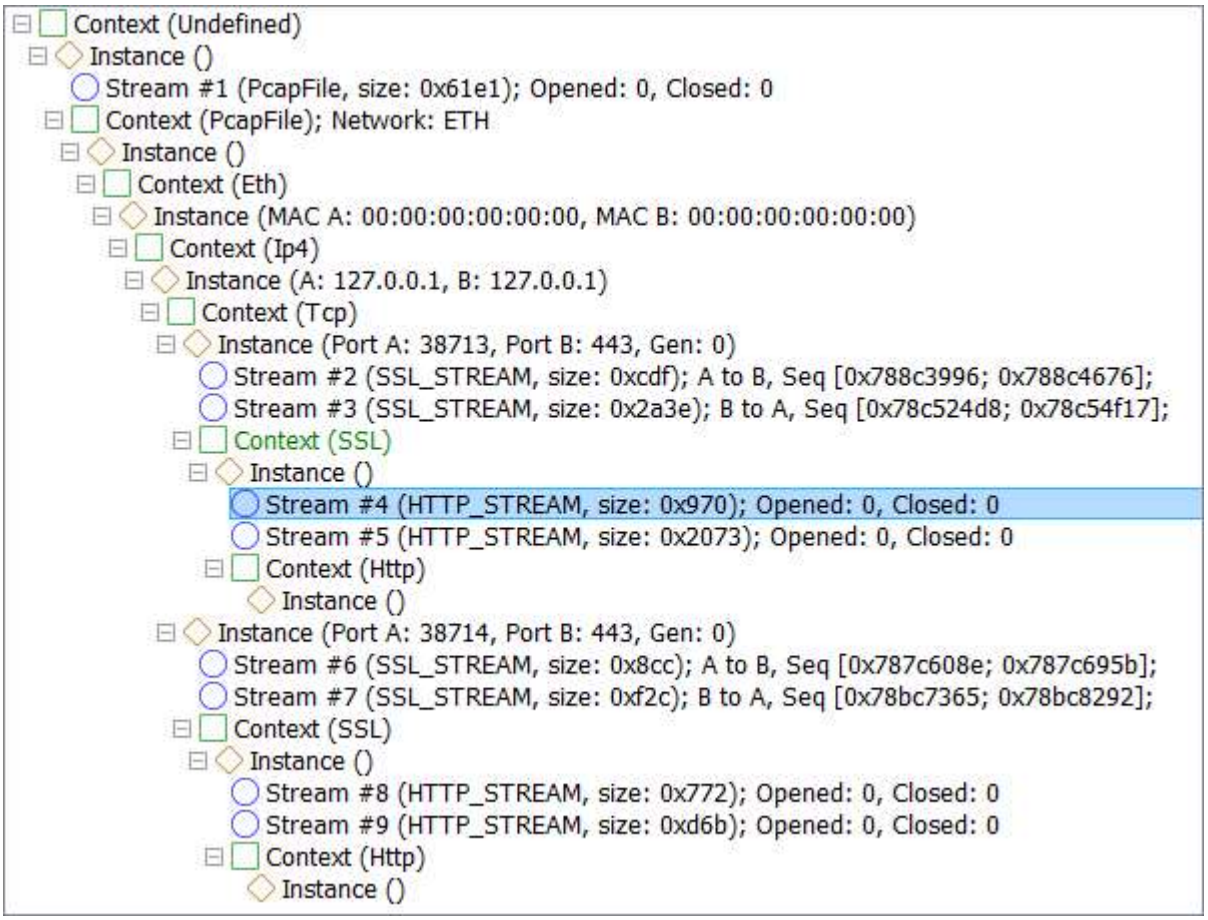

Рис. 30 – Дерево контекстов с использованием подсказки.

5.3 Извлечение файлов при проведении online-анализа

Для online-анализатора показана возможность извлечения из трафика файлов формата PNG. Для проведения эксперимента использовалась сетевая трасса TracePNG.pcap, в которой файлы обозначенного формата передаются посредством пакетов протокола HTTP. Для отправки пакетов трассы на сетевой интерфейс использовалась программа Colasoft Packet Player [49]: разработанный online-анализатор получает пакеты с помощью библиотеки Pcap. В результате эксперимента на жестком диске была создана директория с именем PNG. По окончании эксперимента в ней содержалось 24 PNG-файла. После этого отправка пакетов трассы TracePNG.pcap на сетевой интерфейс была проведена повторно, однако в качестве анализатора трафика при этом использовался Wireshark. В результате повторной отправки пакетов также было восстановлено 24 файла формата PNG. Анализ содержимого извлеченных файлов показал, что результаты online-анализатора совпадают с результатами Wireshark.

5.4 Обратная инженерия закрытого протокола ботнета rbot

Одним из актуальных сценариев использования разработанных инструментов является анализ активности ботнетов [50] – зараженных машин, объединенных в сеть с целью проведения массовых нелегальных или неодобряемых действий. Характерной особенностью функционирования вредоносного программного обеспечения на зараженной машине (*боте*) является координация деятельности злоумышленником с помощью управляющего сервера. Исследователи выделяют несколько схем управления ботами: централизованная и децентрализованная на основе одноранговых сетей (*P2P*). Анализ активности ботнета требует разбора командного протокола – канала связи между зараженной машиной и управляющим сервером.

Функционирующий ботнет, как правило, является модификацией распространяемого в сети ботнета, предоставляющего базовый функционал, которой может быть расширен злоумышленником с помощью дополнительных модулей или плагинов. Таким образом командный протокол ботнетов, принадлежащих одному семейству, может иметь общую основу и различаться в

зависимости от модификаций, произведенных злоумышленником. Злоумышленники предпочитают использовать распространенные протоколы, а не разрабатывать собственные нестандартные решения, прежде всего в целях маскировки вредоносного трафика среди обычного. При реализации командных протоколов нередко используются протоколы интернет-чатов, позволяющих организовать широковещательную рассылку команд вредоносным программам от оператора ботнета. По этой причине протокол IRC [51] часто используется разработчиками ботнетов. Несмотря на то, что популярность IRC-мессенджеров снизилась среди пользователей Интернет, было разработано несколько семейств ботнетов, до сих пор встречающихся в сети.

Программный комплекс анализа сетевого трафика может быть использован при решении задач обратной инженерии протоколов ботнетов. Модульная структура анализатора позволяет создать модуль, в котором локализована функциональность по работе с командным сетевым протоколом. Предполагается итеративная техника разработки модуля, включающая чередование этапов анализа результатов разбора трафика и отладки модуля. При разборе пакета, в котором присутствуют данные, не поддающиеся анализу имеющимся набором разборщиков, в журнал ошибок заносится сообщение о нераспознанном протоколе. Просматривая журнал ошибок, аналитик может локализовать трафик командных протоколов, неверно обрабатываемый модулем разбора. Анализируя неразобранный трафик командных протоколов, разработчик выдвигает гипотезу о структуре протокола. Гипотеза включает в себя набор команд и формат ее параметров, на основании гипотезы дорабатывается модуль разбора, запускается анализатор и повторно выполняется анализ результатов разбора. Ошибки разбора, связанные с командным протоколом, позволяют модифицировать гипотезу. Причиной ошибок может быть отсутствие сигнатуры команды в базе модуля или отсутствие подходящей сигнатуры среди возможных форматов данной команды. Отсутствие ошибок при разборе исследуемой сетевой трассы не позволяет утверждать о полном восстановлении спецификации протокола, поскольку исследуемая трасса может не включать в себя полный набор возможных команд:

доработка модуля разбора возможна на другой сетевой трассе, содержащей командные протоколы данного ботнета.

В качестве примера рассмотрим фрагмент взаимодействия ботнета и управляющего сервера с помощью протокола IRC (рис. 31), выделенный в трафике посредством журнала уведомлений.

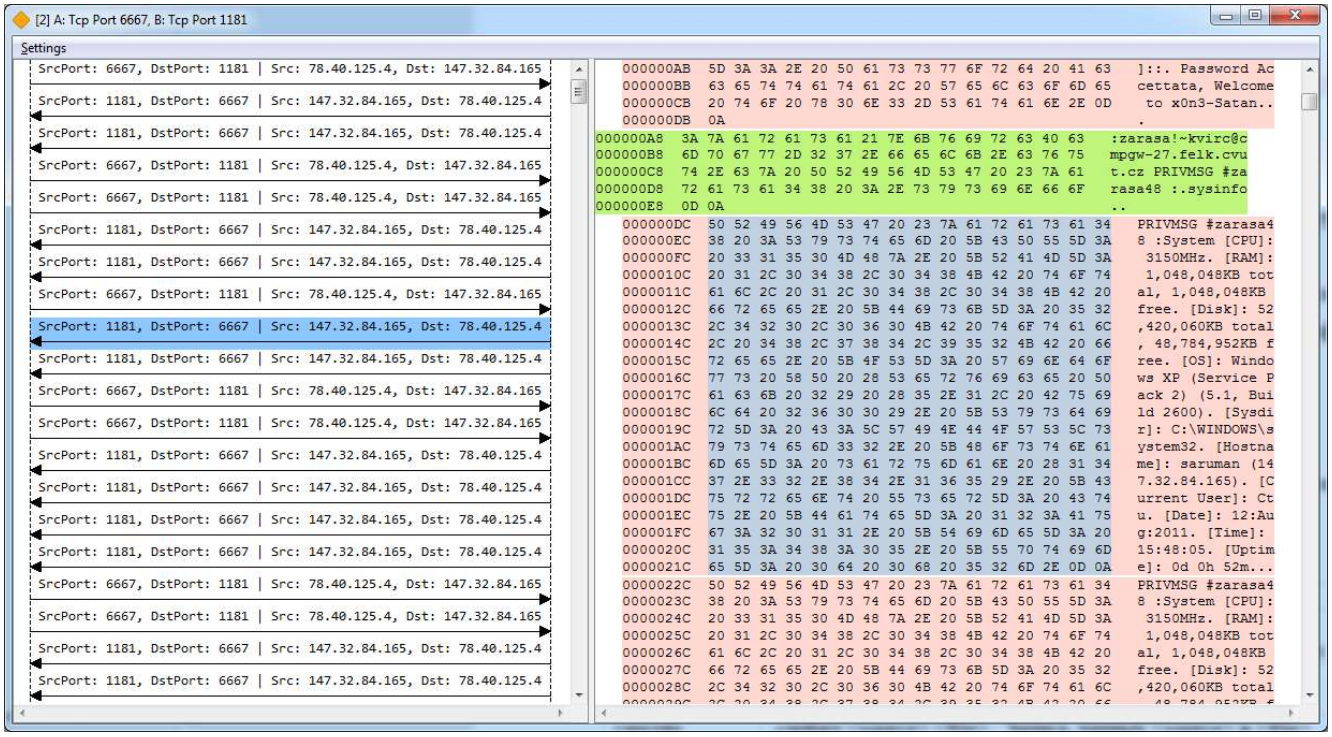

Рис. 31 – Фрагмент взаимодействия ботнета и управляющего сервера.

Данные пакетов позволяют предположить наличие команд в управляющем протоколе, представленных в табл. 10.

Табл. 10 – Команды управляющего протокола ботнета.

Команда	Формат	Описание
login	.login <password>	Авторизация управляющего сервера для управления ботнетом
sysinfo	.sysinfo	Отображение сведений о конфигурации ОС бота
capcute	.capture <source> <file>	Запись данных <source> в <file>
cmd	.cmd <command>	Выполнение команды в командной строке
driveinfo	.driveinfo	Отображение сведений о дисковых устройствах ОС
get	.get <file>	Отправка файла с бота на управляющий сервер

netinfo	.netinfo	Отображение сведений о сетевой активности

На основе данного набора возможных команд был создан прототип модуля разбора, способного в автоматическом режиме распознавать командные протоколы в трафике. Распознавание типа командного протокола было реализовано посредством регистрации распознавателя для поля <message> пакетов протокола IRC. Распознаватель проверяет текст сообщения на наличие команды из списка и в случае обнаружения выполняет проверку формата команды с помощью регулярного выражения. Чем больше команд и их форматов будет добавлено в распознаватель, тем точнее будет проводиться анализ.

Заключение

Основные результаты работы

1. Разработана модель представления разобранных сетевых пакетов. В модели учтены следующие особенности передачи данных по сети:
   - потери/переупорядочивание отдельных пакетов;
   - сжатие и шифрование данных;
   - вложенное туннелирование.
2. Разработан алгоритм восстановления потоков данных для протоколов произвольного уровня, в том числе прикладного, устойчивый к потерям отдельных сетевых пакетов, а также их переупорядочиванию.
3. Разработана архитектура системы углубленного анализа сетевого трафика, позволяющая разрабатывать и отлаживать модули поддержки протоколов на предварительно сохраненном трафике (offline) и впоследствии использовать эти модули в режиме online.
4. На основе предложенных автором модели, алгоритма и архитектуры разработаны и реализованы программные инструменты для проведения углубленного анализа сетевого трафика в online и offline режимах.

Представленная система может успешно применяться при анализе трафика как в online-, так и в offline-режиме. Предложенная модель описания данных охватывает существующие особенности передачи сетевого трафика. Разработанный алгоритм восстановления потоков устойчив к потерям отдельных сетевых пакетов и их переупорядочиванию.

Взаимодействие между узлами сети, как правило, не ограничивается одним потоком данных. Например, взаимодействие клиента с FTP-сервером предполагает организацию как минимум двух таких потоков: для отправки команд на сервер (управляющий поток) и для передачи файлов клиенту (поток

передачи контента). Аналитику может быть полезна функциональность по выявлению связей между такими потоками: она позволит рассматривать взаимодействие между узлами сети на качественно более высоком уровне.

Для формального описания разборщиков применяется язык C++: аналитик непосредственно реализует функции разбора и распознавания данных. В то же время функции разбора могут быть сгенерированы автоматически на основе некоторого представления структуры сообщения протокола. Разработка и внедрение высокоуровневого декларативного языка [52] описания формата позволит ускорить процесс разработки разборщиков. Формальное описание структуры также может использоваться для генерации трафика с заданными характеристиками. Подобная функциональность необходима при решении задачи обеспечения информационной безопасности с помощью фаззинга и, в частности, таких инструментов, как Peach или SPIKE [53].

Список литературы

[1] Mike Cloppert. An Overview Of Protocol Reverse-Engineering. https://digital-forensics.sans.org/blog/2012/07/03/an-overview-of-protocol-reverse-engineering, дата обращения 02.02.2017

[2] IETF RFC 791. J. Postel. Internet Protocol, September 1981

[3] Antonios Atlasis. Fragmentation (Overlapping) Attacks One Year Later, Troopers 13 – IPv6 Security Summit, 2013

[4] IETF RFC 793. J. Postel. Transmission Control Protocol, September 1981

[5] Judy Novak, Steve Sturges. Target-Based TCP Stream Reassembly, 2007

[6] Jon C. R. Bennett, Craig Partridge, Nicholas Shectman. Packet reordering is not pathological network behavior // IEEE/ACM Transactions on Networking (TON) archive, Volume 7 Issue 6, Dec. 1999, Pages 789-798

[7] IETF RFC 2616. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999

[8] А. Ализар. Доля зашифрованного трафика в интернете выросла в несколько раз. https://xakep.ru/2014/05/15/62500/, дата обращения 10.02.2017

[9] VPN Tunneling Protocols. https://technet.microsoft.com/en-us/library/6e3cd69c-dc8c-483e-98bc-8d2e7e76e048, дата обращения 01.02.2017

[10] Мэйволд Э. «Межсетевые экраны» // Безопасность сетей: Информация, ИНТУИТ, 2006. http://www.intuit.ru/studies/courses/102/102/lecture/2989, дата обращения 10.01. 2017

[11] Hussain Ahmad Madni Uppal, Memoona Javed and M.J. Arshad. An Overview of Intrusion Detection System (IDS) along with its Commonly Used Techniques and Classifications // International Journal of Computer Science and Telecommunications, volume 5, Issue 2, 2014

[12] Christos Douligeris, Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art // Computer Networks, 2003.

[13] Laura Chappell. Troubleshooting Tips and Tricks for TCP/IP Networks // SHARKFEST ‘11, Stanford University, June 13–16, 2011

[14] P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012

[15] Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Автоматизация тестирования соответствия для телекоммуникационных протоколов // Труды Института системного программирования РАН, том 26, выпуск 1, 2014, стр. 109-148

[16] Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800-94, 127 pages, February 2007

[17] Maurizio Dusi, Francesco Gringoli, Luca Salgarelli. IP Traffic Classification for QoS Guarantees: the Independence of Packets // In: Proceedings of The 1st IEEE International Workshop on IP Multimedia Communication (IPMC 2008), Augush 2008.

[18] ГОСТ Р ИСО/МЭК 7498-1-99. — «ВОС. Базовая эталонная модель. Часть 1. Базовая модель». — ОКС: 35.100.70. — Действует с 01.01.2000. — 62с.

[19] Christopher Parsons. Deep Packet Inspection in Perspective: Tracing its lineage and surveillance potentials // Working Paper, January 2009

[20] IETF RFC 768. J. Postel. User Datagram Protocol, August 1980

[21] IEEE Standard for Ethernet, 802.3-2012 – section one. 2012-12-28. p. 53. Retrieved 2014-07-06.

[22] Xiaoming Zhou and Piet Van Mieghem. Reordering of IP Packets in Internet // International Workshop on Passive and Active Network Measurement, PAM 2004, pp 237-246

[23] Arjuna Sathiaseelan and Tomasz Radzik. Improving the performance of TCP in the case of packet reordering // IEEE International Conference on High Speed Networks and Multimedia Communications, HSNMC 2004, pp 63-73

[24] IETF RFC 5246. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[25] А. В. Никешин, Н. В. Пакулин, В. З. Шнитман. Тестирование реализаций клиента протокола TLS // Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 145-160

[26] Andrew Moore, James Hall, Christian Kreibich, Euan Harris, and Ian Pratt. Architecture of a Network Monitor // International Workshop on Passive and Active Network Measurement, PAM 2003

[27] Arnt Brox. Signature-Based or Anomaly-Based Intrusion Detection: The Practice and Pitfalls. https://www.scmagazine.com/signature-based-or-anomaly-based-intrusion-detection-the-practice-and-pitfalls/article/548733/, дата обращения 10.10.2016

[28] Snort. http://www.snort.org/, дата обращения 07.10.2016

[29] The Bro Network Security Monitor. http://www.bro.org/, дата обращения 07.11.2016

[30] Wireshark. http://www.wireshark.org/, дата обращения 07.05.2016

[31] Tcpdump. http://www.tcpdump.org/, дата обращения 24.02.2017

[32] Wireshark Display Filter Reference. https://www.wireshark.org/docs/dfref/, дата обращения 24.02.2017

[33] IETF RFC 4251. T. Ylonen, C. Lonvick, The Secure Shell (SSH) Protocol Architecture, January 2006

[34] Documentation to DCE/RPC, http://www.dcerpc.org/documentation/, дата обращения 19.02.2017

[35] Microsoft SMB Protocol and CIFS Protocol Overview, https://msdn.microsoft.com/en-us/library/aa365233(VS.85).aspx, дата обращения 19.02.2017

[36] Signature Framework. https://www.bro.org/sphinx/frameworks/signatures.html, дата обращения 23.04.2016

[37] Таненбаум Э., Уэзеролл Д. Компьютерные сети, с. 371 – 378 // Э. Таненбаум, Д. Уэзеролл. – 5-е изд. — СПб.: Питер, 2012. – 960 с.

[38] IETF RFC 2784. D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina. Generic Routing Encapsulation (GRE), March 2000

[39] IETF RFC 1661. W. Simpson. The Point-to-Point Protocol (PPP), July 1994

[40] IETF RFC 1035. P. Mockapetris. Domain Names – Implementation and Specification, November 1987

[41] Tunneling Protocol Support. Multiple Encapsulations. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node10.html, дата обращения 22.01.2017

[42] SSL/TLS. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node17.html#SECTION00321400000000000000, дата обращения 26.02.2017

[43] Сетевые модели OSI и TCP/IP. http://www.quizful.net/post/osi_tcpip_layers, дата обращения 26.02.2017

[44] Стек IPX/SPX. http://citforum.ru/nets/protocols/1_02_03.shtml, дата обращения 20.03.2017

[45] Oink: a Collaboration of C/C++ Tools for Static Analysis and Source-to-Source Transformation. http://daniel-wilkerson.appspot.com/oink/index.html/, дата обращения 10.10.2016

[46] Distributed Messaging. http://zeromq.org/, дата обращения 07.03.2017

[47] Robert Shimonski. The Wireshark Field Guide: Analyzing and Troubleshooting Network Traffic, pp. 87 – 90, 2013 Elsevier Inc

[48] Rivest R., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems // Commun. ACM — New York City: ACM, 1978. — Vol. 21, Iss. 2. — P. 120 – 126

[49] Colasoft Packet Player. http://www.colasoft.com/packet_player/, дата обращения 21.02.2017

[50] Sebastian Garcıa. Identifying, Modeling and Detecting Botnet Behaviors in the Network // Doctoral Thesis, Novemver 28, 2014

[51] IETF RFC 2812. C. Kalt. Internet Relay Chat: Client Protocol, April 2000

[52] BinPAC. https://www.bro.org/sphinx/components/binpac/README.html, дата обращения 20.11.2016

[53] Саттон М., Грин А., Амини П. Fuzzing: исследование уязвимостей методом грубой силы. - Пер. с англ. // СПб.: Символ-Плюс, 2009, с. 371 – 389