МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра теоретических основ компьютерной безопасности и криптографии

ОБНАРУЖЕНИЕ СЕТЕВОГО P2P ТРАФИКА

КУРСОВАЯ РАБОТА

студента 3 курса 331 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Научный руководитель
доцент _____ А. В. Гортинский

Заведующий кафедрой
д. ф.-м. н., доцент _____ М. Б. Абросимов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Введение

1 Peer-to-Peer

P2P (**p**eer-**to**-**p**eer), также известные как одноранговые, децентрализованные или пиринговые сети, — это распределенная архитектура приложения, которая разделяет задачи между узлами (peer). Узлы имеют одинаковые привилегии в приложении и образуют сеть равносильных узлов.

Узлы делают свои ресурсы, такие как вычислительная мощность, объем диска или пропускная способность напрямую доступными остальным членам сети, без необходимости координировать действия с помощью серверов. Узлы являются одновременно поставщиками и потребителями ресурсов, в отличие от стандартной клиент-сервер модели, где поставщик и потребитель ресурсов разделены. [1]

1.1 История

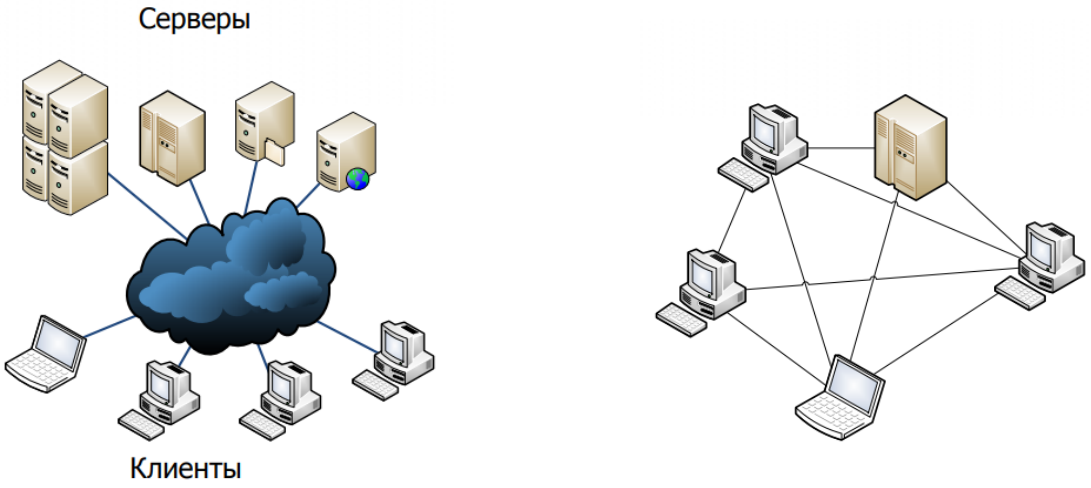
В то время как P2P системы использовались во многих доменных приложениях, архитектура популяризовалась благодаря файлообменной системе Napster, разработанной в 1999 году. Концепция вдохновила новую философию во многих областях человеческого взаимодействия. P2P-технология позволяет пользователям интернета образовывать группы и коллаборации, формируя, тем самым, пользовательские поисковые движки, виртуальные суперкомпьютеры и файловые системы. Основная идея P2P-систем исходит из первых принципов метода Request for Comment (RFC). Видение Всемирной паутины Тима Бернерса-Ли было близко к P2P-сети, в том смысле, что каждый пользователь является активным создателем и редактором контента.

Ранней версией P2P-сетей является USENET — распределенная система обмена сообщениями. USENET был разработан в 1979 году и представлял собой систему, обеспечивающую децентрализованную модель управления. Основа представляет собой клиент-серверную модель, предполагающую самоорганизацию группы серверов. Тем не менее сервера взаимодействуют друг с другом как равноправные узлы, распространяя информацию по всей сети USENET.

В мае 1999 года, в Интернет с более чем миллионом пользователей, Шон Фэннинг внедрил приложение файлообменник Napster. Napster стал началом P2P-сети, такой какую мы знаем её сейчас, пользователи участвуют в создании виртуальной сети, полностью независимой от физической, без администрирования и каких-либо ограничений.

2 Архитектура

P2P-сеть строится вокруг понятия равноправных узлов — клиенты и серверы одинаково взаимодействуют с другими узлами сети. Такая модель построения сети отличается от модели клиент-сервер, где взаимодействие идет с центральным сервером. На рисунке 1 а) изображены архитектура клиент-сервера и б) архитектура P2P. Типичным примером передачи файла в модели клиент-сервер является File Transfer Protocol (FTP), в котором программы клиента и сервера разделены: клиент инициирует передачу, а сервер отвечает на запросы.



а) б)

Рисунок 1 – Архитектура клиент-сервера и P2P

2.1 Базовые элементы P2P-сетей

2.1.1 Узел P2P-сети

Узел (Peer) — фундаментальный составляющие блок любой одноранговой сети. Каждый узел имеет уникальный идентификатор и принадлежит одной или нескольким группам. Он может взаимодействовать с другими узлами как в своей, так и в других группах. [2]

Виды узлов:

— **Простой узел**. Обеспечивает работу конечного пользователя, предоставляя ему сервисы других узлов и обеспечивая предоставление ресурсов пользовательского компьютера другим участникам сети.

— **Роутер**. Обеспечивает механизм взаимодействия между узлами, отделёнными от сети брандмауэрами или NAT-системами.

2.1.2 Группа узлов

Группа узлов — набор узлов, сформированный для решения общей задачи или достижения общей цели. Могут предоставлять членам своей группы такие наборы сервисов, которые недоступны узлам, входящим в другие группы.

Группы узлов могут разделяться по следующим признакам:

- приложение, ради которого они объединены в группу;
- требования безопасности;
- необходимость информации о статусе членов группы.

2.1.3 Сетевой транспорт

Конечные точки (Endpoints) — источники и приёмники любого массива данных передаваемых по сети.

Пайпы (Pipes) — однонаправленные, асинхронные виртуальные коммуникационные каналы, соединяющие две или более конечные точки.

Сообщения — контейнеры информации, которая передаётся через пайп от одной конечной точки до другой.

2.2 Маршрутизация

P2P-сети обычно реализуют некоторую форму виртуальной (логической) сети, наложенную поверх физической, то есть описывающей реальное расположение и связи между узлами, сети, где узлы образуют подмножество узлов в физической сети. Данные по-прежнему обмениваются непосредственно над базовой TCP/IP сетью, а на прикладном уровне узлы имеют возможность взаимодействовать друг с другом напрямую, с помощью логических связей. Наложение используется для индексации и обнаружения узлов, что позволяет системе P2P быть независимой от физической сети. На основании того, как узлы соединены друг с другом внутри сети, и как ресурсы индексированы и расположены, сети классифицируются на **неструктурированные** и **структурированные** (или как их **гибрид**).

2.2.1 Неструктурированные сети

Неструктурированная P2P сеть не формирует определенную структуру сети, а случайным образом соединяет узлы друг с другом. Так как не существует глобальной структуры формирования сети, неструктурированные сети легко организуются и доступны для локальных оптимизаций. Кроме того, поскольку

роль всех узлов в сети одинакова, неструктурированные сети являются весьма надежными в условиях, когда большое количество узлов часто подключаются к сети или отключаются от нее.

Однако, из-за отсутствия структуры, возникают некоторые ограничения. В частности, когда узел хочет найти нужный фрагмент данных в сети, поисковый запрос должен быть направлен через сеть, чтобы найти как можно больше узлов, которые обмениваются данными. Такой запрос вызывает очень высокое количество сигнального трафика в сети, требует высокой производительности, и не гарантирует, что поисковые запросы всегда будут решены.

2.2.2 Структурированные сети

В структурированных P2P сетях наложение организуется в определенную топологию, и протокол гарантирует, что любой узел может эффективно участвовать в поиске файла или ресурса, даже если ресурс использовался крайне редко.

Наиболее распространенный тип структурированных сетей P2P реализуется распределенными хэш-таблицами (DHT), в котором последовательное хеширование используется для привязки каждого файла к конкретному узлу. Это позволяет узлам искать ресурсы в сети, используя хэш-таблицы, хранящих пару ключ-значение, и любой участвующий узел может эффективно извлекать значение, связанное с заданным ключом.

Тем не менее, для эффективной маршрутизации трафика через сеть, узлы структурированной сети должны обладать списком соседей, которые удовлетворяют определенным критериям. Это делает их менее надежными в сетях с высоким уровнем оттока абонентов (т.е. с большим количеством узлов, часто подключающихся к сети или отключающихся от нее).

2.2.3 Гибридные модели

Гибридные модели представляют собой сочетание P2P сети и модели клиент-сервер. Гибридная модель должна иметь центральный сервер, который помогает узлам находить друг друга. Есть целый ряд гибридных моделей, которые находят компромисс между функциональностью, обеспечиваемой структурированной сетью модели клиент-сервер, и равенством узлов, обеспечиваемой чистыми одноранговыми неструктурированными сетями. В настоящее время гибридные модели имеют более высокую производительность, чем чисто неструк-

турированные или чисто структурированные сети.

2.3 Безопасность

Как и любой другой форме программного обеспечения, P2P-приложения могут содержать уязвимости. Особенно опасно для P2P программного обеспечения, является то, что P2P приложения действуют и в качестве серверов и в качестве клиентов, а это означает, что они могут быть более уязвимы для удаленных эксплоитов.

2.3.1 Маршрутизационные атаки

Поскольку каждый узел играет роль в маршрутизации трафика через сеть, злоумышленники могут выполнять различные «маршрутизационные атаки», или атаки отказа в обслуживании. Примеры распространенных атак маршрутизации включают в себя «неправильная маршрутизация поиска», когда вредоносные узлы преднамеренно пересылают запросы неправильно или возвращают ложные результаты, «неправильная маршрутизация обновления», когда вредоносные узлы изменяют таблицы маршрутизации соседних узлов, посылая им ложную информацию, и «неправильная маршрутизация разделения сети», когда новые узлы подключаются через вредоносный узел, который помещает новичков в разделе сети, заполненной другими вредоносными узлами.

2.3.2 Поврежденные данные и вредоносные программы

Распространенность вредоносных программ варьируется между различными протоколами одноранговых сетей. Исследования, анализирующие распространение вредоносных программ по сети P2P обнаружили, например, что 63% запросов на загрузку по сети Limewire содержали некоторую форму вредоносных программ, в то время как на OpenFT только 3% запросов содержали вредоносное программное обеспечение. Другое исследование анализа трафика в сети Kazaa обнаружили, что 15% от 500 000 отобранных файлов, были инфицированы одним или несколькими из 365 различных компьютерных вирусов.

Поврежденные данные также могут быть распределены по P2P-сети путем изменения файлов, которые уже были в сети. Например, в сети FastTrack, RIAA удалось внедрить фальшивые данные в текущий список загрузок и в уже загруженные файлы (в основном файлы MP3). Файлы, инфицированные вирусом RIAA были непригодны впоследствии и содержали вредоносный код.

Следовательно, P2P-сети сегодня внедрили огромное количество механизмов безопасности и проверки файлов. Современное хеширование, проверка данных и различные методы шифрования сделали большинство сетей, устойчивыми к практически любому типу атак, даже когда основные части соответствующей сети были заменены фальшивыми или нефункциональными узлами.

2.4 Отказоустойчивость и масштабируемость сети

Децентрализованность P2P сетей повышает их надежность, так как этот метод взаимодействия устраняет ошибку единой точки разрыва, присущую клиент-серверным моделям. С ростом числа узлов, объем трафика внутри системы увеличивается, масштаб сети также увеличивается, что приводит к уменьшению вероятности отказа. Если один узел перестанет функционировать должным образом, то система в целом все равно продолжит работу. В модели клиент-сервер, с ростом количества пользователей, уменьшается количество ресурсов выделяемых на одного пользователя, что приводит к риску возникновения ошибок.

2.5 Распределенное хранение и поиск

Возможность резервного копирования данных, восстановление и доступность приводят как и к преимуществам, так и к недостаткам P2P сетей. В централизованной сети, только системный администратор контролирует доступность файлов. Если администраторы решили больше не распространять файл, его достаточно удалить с серверов, и файл перестанет быть доступным для пользователей. Другим словами, клиент-серверные модели имеют возможность управлять доступностью файлов. В P2P сети, доступность контента определяется степенью его популярности, так как поиск идет по всем узлам, через которые файл проходил. То есть, в P2P сетях нет централизованного управления, как системный администратор в клиент-серверном варианте, а сами пользователи определяют уровень доступности файла.

3 Применение

В P2P сетях, пользователи передают и используют контент сети. Это означает, что в отличие от клиент-серверных сетей, скорость доступа к данным возрастает с увеличением числа пользователей, использующих этот контент. На этой идее построен протокол Bittorrent — пользователи скачавшие файл, становятся узлами и помогают другим пользователям скачать файл быстрее. Эта особенность является главным преимуществом P2P сетей.

Множество файлообменных систем, таких как Gnutella, G2 и eDonkey популяризовали P2P технологии:

- Пиринговые системы распространения контента.
- Пиринговые системы обслуживания, например повышение производительности, в частности Correli Caches.
- Публикация и распространение программного обеспечения (Linux, видеоигры).

В связи децентрализованностью доступа к данным в P2P сетях возникает проблема нарушения авторских прав. Компании, занимающиеся разработкой P2P приложений часто принимают участие в судебных конфликтах. Самые известные судебные дела это Grokster против RIAA и MGM Studios, Inc. против Grokster Ltd., где в обоих случаях технологии файлообменных систем признавались законными.

4 Способы обнаружения P2P трафика

4.1 Анализ портов

Многие P2P-приложения работают на определённых портах. Некоторые из таких указаны в таблице 1 [3].

Таблица 1 – Список наиболее известных портов, используемых P2P-протоколами

Протоколов	Номера TCP/UDP портов
BitTorrent	6881-6999
Direct Connect	411, 412, 1025-32000
eDonkey	2323, 3306, 4242, 4500, 4501, 4661-4674, 4677, 4678, 4711, 4712, 7778
FastTrack	1214, 1215, 1331, 1337, 1683, 4329
Yahoo	5000-50010, 5050, 5100
Napster	5555, 6257, 6666, 6677, 6688, 6699-6701
MSN	1863, 6891-6901
MP2P	10240-20480, 22321, 41170
Kazaa	1214
Gnutella	6346, 6347
ARES Galaxy	32285
AIM	1024-5000, 5190

Для реализации данного метода достаточно обнаружить в сетевом трафике соединения, использующие такие порты. Очевидно, что данный способ легко реализовать, однако он имеет недостатки. Во-первых, многие приложения могут использовать случайные порты, или же пользователь может сам выбрать номер порта. Во-вторых, такие порты могут использоваться не P2P-приложениями и наоборот, P2P-приложения могут использовать номера портов известных приложений, например, 80 или 443 порты — HTTP и HTTPS. Так, в работе [4] приведены результаты, которые показывают, что зачастую на основе данного метода можно определить лишь 30% P2P трафика.

4.2 Анализ сигнатур

Суть этого способа заключается в мониторинге трафика, проходящего через сеть, на предмет обнаружения определенных **сигнатур**, специфичных для P2P-приложений, в полезной нагрузке пакетов [5]. Сетевая сигнатура — набор данных, которые необходимо найти в трафике. Это могут быть IP-адреса, порты,

флаги (например, протокола TCP) и так далее. Многие современные коммерческие и свободно распространяемые решения для обнаружения P2P трафика основаны на этом методе. Например, система предотвращения вторжений Snort предоставляет возможность создавать набор правил, включающих информацию о характеристиках транспортного уровня и содержимом полезной нагрузки пакетов для различных приложений.

Особенности данного метода:

— Необходимо постоянное обновление базы сигнатур.
— Трафик зачастую зашифрован, что сильно затрудняет анализ.
— Поиск сигнатур на прикладном сетевом уровне очень ресурсоёмкий.

4.3 Эвристические предположения

4.3.1 TCP/UDP-эвристика

Если в течение интервала времени обнаружено, что пара адресов взаимодействует и по TCP-, и по UDP-протоколу, то они предположительно участвуют в P2P-обмене.

4.3.2 IP/Port-эвристика

Ещё одной особенностью P2P является тот факт, что при обращении к паре { IP-назначения, порт-назначения } количество адресов источников практически совпадает с количеством портов источников. Подобное поведение характерно, например, для сигнального взаимодействия с раздающим данные BitTorrent-клиентом. Однако стоит отметить, что такое поведение характерно не только для P2P.

5 Идентификация BitTorrent

В работе [6] предложен алгоритм, который основывается на четырёх критериях.

5.1 Подключенные IP-адреса

Первый критерий основан на IP/Port-эвристике. Хосты BitTorrent всегда подключены ко многим IP-адресам. Подразумевается, что под подключенными IP-адресами понимается, что они передали друг другу хотя бы по одному TCP-пакету. В BitTorrent это может быть необходимо для подключения к раздаче и передачи особенных сообщений (choke, have, keepalive). Причём каждый пир (участник) пытается поддерживать не менее 20 пиров, следовательно, каждый пир периодически отправляет несколько TCP-пакетов на один и тот же набор IP-адресов.

5.2 Передача данных

BitTorrent разбивает исходные файлы на небольшие части, поэтому пользователи могут скачивать разные файлы от разных пользователей. Это можно определить по значимому соотношению активных передач. Под активной передачей подразумевается хотя бы 5 больших TCP-пакетов, т.е. размер пакета должен быть примерно равен *MTU* (максимальная единица передачи). В Ethernet это около 1500 байт. Передача пакетов максимального размера необходима для того, чтобы их количество было минимальным для передачи файла.

Однако пиры BitTorrent не всегда одновременно обмениваются данными между собой. Это связано с *алгоритмом дросселирования* (*choke*). Этот алгоритм выбирает соседей, которым будут раздаваться или с которых будут скачиваться файлы. В любой момент времени пир загружает данные не более, чем с 4 пиров, которые обеспечивают самую высокую скорость загрузки.

5.3 Двусторонняя передача данных

Процесс выбора в алгоритме дросселирования приводит к двусторонней передаче данных. В отличие от BitTorrent, другие интернет-приложения обычно работают по схеме клиент-сервер, поэтому данные передаются только в одном направлении в определённый промежуток времени. Кроме того, в других протоколах P2P-обмена между элементами нет взаимного обмена, который заложен в

алгоритме дросселирования. Пирам в этих протоколах не нужно загружать свои фрагменты другим пирам, с которых они скачивают данные.

5.4 Изменение отношений

В алгоритме дросселирования все пиры в наборе сортируются каждые 10 секунд в порядке убывания скорости загрузки данных. После сортировки локальный пир будет раздавать данные только первым четырём пирам в отсортированном списке. Учитывая, что скорость передачи довольно динамична, выбранные пиры будут часто меняться. Таким образом, пара пиров может активно передавать данные друг другу, но потом внезапно может стать неактивной. В результате хост BitTorrent может быть идентифицирован по значимому соотношению изменений IP-отношений к активным передачам.

5.5 Алгоритм

На основании четырёх критериев создаются специальные метрики, которые рассчитываются каждые 30 секунд и сравниваются с пороговым значением, чтобы определить, является ли хост пиром BitTorrent. В данном алгоритме обрабатываются только TCP-пакеты.

5.5.1 Подключения

Подсчитывается число C — количество пиров, которые общались с хостом. Если это количество будет больше или равно порогу $C_{threshold}$, то хост будет идентифицирован как BitTorrent-хост.

$$C \geq C_{threshold}$$

5.5.2 Коэффициент активной передачи

Коэффициент активной передачи хоста R_{AT} — отношение числа активных подключений AT к общему числу подключений C. Если этот коэффициент больше или равен пороговому $R_{ATthreshold}$, то хост будет идентифицирован как BitTorrent-хост.

$$R_{AT} \geq R_{ATthreshold},$$

где $R_{AT} = \frac{AT}{C}$.

5.5.3 Двусторонние передачи данных

Измеряется количество подключений $BiAT$, по которым одновременно принимаются и отправляются данные. Если это число больше или равно пороговому $BiAT_{threshold}$, то хост будет идентифицирован как BitTorrent-хост.

$$BiAT \geq BiAT_{threshold}$$

5.5.4 Коэффициент изменений отношений

Коэффициент изменений отношений R_{RC} — отношение числа изменений отношений RC к числу активных передач AT. Если этот коэффициент больше или равен пороговому $R_{RCthreshold}$, то хост будет идентифицирован как BitTorrent-хост.

$$R_{RC} \geq R_{RCthreshold},$$

где $R_{RC} = \frac{RC}{AT}$.

5.5.5 Точность алгоритма

Точность данного алгоритма зависит от выбранных пороговых значений. Они могут быть получены эмпирическим путём.

6 Описание программы

В данной работе был разработан **сниффер** — анализатор сетевого трафика. Программа выводит на экран информацию о перехваченных пакетах таких сетевых протоколов как *IPv4*, *TCP* и *UDP*. Дополнительно последний вывод программы сохраняется в текстовые файлы. Программа обрабатывает трафик, проходящий через ту машину, на которой она запущена. Таким образом, в каждом перехваченном пакете будет фигурировать локальный IP-адрес машины. Важно отметить, что при анализе трафика с целью обнаружения P2P-активности данный IP-адрес игнорируется.

При запуске *window.py* создаётся **сокет** — программный интерфейс для обеспечения обмена данными между процессами. Через него проходит весь сетевой трафик на той виртуальной машине, на которой он находится.

```
conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
```

6.1 Интерфейс

На рисунке 2 можно увидеть вывод основной информации о *TCP/UDP* пакетах и список IP-адресов, которые взаимодействуют с локальным адресом через P2P.

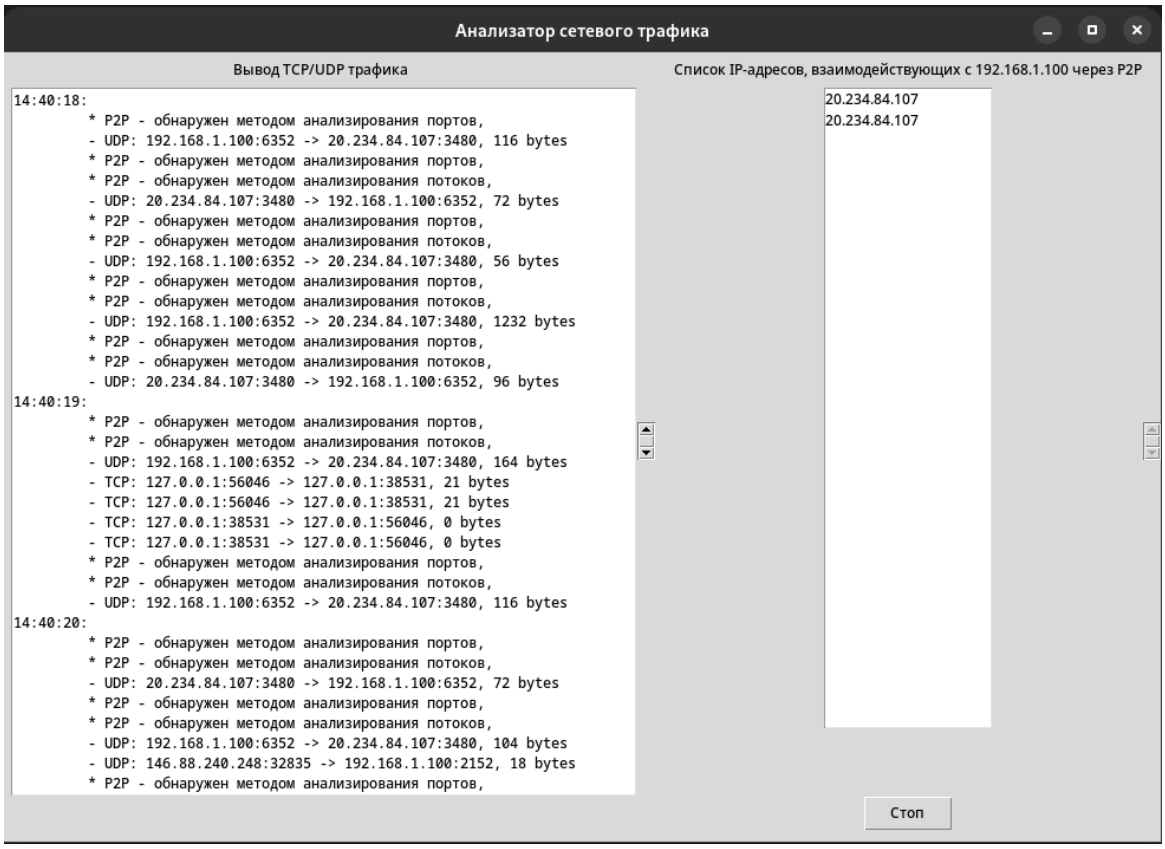


Рисунок 2 – Тестирование работы программы при запущенном звонке в Skype

6.2 Функция sniff

Функция *sniff* вызывает функцию *main* из *sniffer.py*, которая передаёт информацию о пакете для вывода на экран. Эта информация сохраняется в список *out*. Затем, если в *out* был обнаружен IP-адрес, который взаимодействует с локальным адресом машины через P2P, то в вывод дополнительно заносится строка, обозначающая способ обнаружения P2P-активности. Затем информация выводится на экран и сохраняется в текстовый файл *out.txt*.

```
def sniff(self):
    out = sniffer.main(conn)
    if out:
        for addr in sniffer.p2p_addrs:
            if len(out) > 1:
                if addr in out[1]:
                    out.insert(0, TAB_2 + 'P2P - обнаружен методом анализирования потоков,')
            elif addr in out[0]:
                out.insert(0, TAB_2 + 'P2P - обнаружен методом анализирования потоков,')

        for addr in sniffer.p2p_addrs1:
            if len(out) > 1:
                if addr in out[1]:
                    out.insert(0, TAB_2 + 'P2P - обнаружен методом анализирования портов,')
            elif addr in out[0]:
                out.insert(0, TAB_2 + 'P2P - обнаружен методом анализирования портов,')
        time = str(datetime.now().strftime('%H:%M:%S')) + ":"
        if time != self.last_time:
            out.insert(0, time)
        self.last_time = time

        for s in out:
            file.write(s + '\n')
            self.output.insert('end', s + '\n')
    root.after(100, self.sniff)  # сканирование каждые 0.1 сек
```

Функция *sniff* вызывает рекурсивно саму себя каждые 0.1 секунды, то есть сканирование происходит раз в 0.1 секунды. Эмпирически было установлено, что для анализирования трафика пары приложений, одно из которых взаимодействует через P2P достаточно сканировать раз в 0.2-0.3 секунды. Однако при более активном трафике сканирование следует проводить чаще, чтобы не было пакетов, которые не оказались бы перехваченными.

6.3 Функция main

Функция *main* обрабатывает информацию о пакете и сохраняет некоторые данные с помощью функции *save* в глобальные переменные. В множества *TCP_addrs* и *UDP_addrs* сохраняются IP-адреса, взаимодействующие по соответствующим протоколам. В множество *rejected* добавляются IP-адреса, которые работают на портах, перечисленных в списке *EXCEPTIONS*. Это необходимо, чтобы отсечь их при анализе потоков, поскольку их активность схожа с P2P-активностью. Например, это могут быть почтовые или *DNS* сервисы.

В словарь *dict_ipport* сохраняются пары вида { dest_ip + dest_port → объект класса IPPort }. В таких объектах сохраняется информация о различных адресах источника и различных портов источника для каждой пары адреса назначения { dest_ip + dest_port }.

```
def main(conn):
output = []
outline = ''

raw_data, addr = conn.recvfrom(65536)
dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)

# IPv4
if eth_proto == 8:
    (version, header_length, ttl, proto, src, dest, data) = ipv4_packet(data)

    # TCP
    if proto == 6:
        src_port, dest_port, sequence, ack, flag_urg, flag_ack, \
        flag_psh, flag_rst, flag_syn, flag_fin, data = tcp_segment(data)

        outline += TAB_1 + 'TCP: ' + src + ':' + str(src_port) + ' -> ' + dest + ':' + \
                   str(dest_port) + ', ' + str(len(data)) + ' bytes'
        output.append(outline)

        save(src, dest, src_port, dest_port)
        check_ports(src, dest, src_port, dest_port)

    # UDP
    elif proto == 17:
        src_port, dest_port, length, data = udp_segment(data)

        outline += TAB_1 + 'UDP: ' + src + ':' + str(src_port) + ' -> ' + dest + ':' + \
                   str(dest_port) + ', ' + str(len(data)) + ' bytes'
        output.append(outline)
```

```
            check_ports(src, dest, src_port, dest_port)
            save(src, dest, src_port, dest_port)

        check_intersection()

return output
```

6.4 Определение P2P трафика

6.4.1 Метод анализирования портов

С помощью функции *check_ports*, которая вызывается при каждом запуске *main*, проводится анализ портов. Если был обнаружен порт, который присутствует в списке *LIST_P2P*, то IP-адрес заносится в *p2p_addrs1*.

```
def check_ports(src, dest, src_port, dest_port):
if LIST_P2P.get(src_port, False) or LIST_P2P.get(dest_port, False):
    if src != UIP:
        p2p_addrs1.add(src)
    else:
        p2p_addrs1.add(dest)
```

В списке пар порт-приложение *LIST_P2P* находится информация об используемых портах некоторых P2P-приложений, а именно:

- BitTorrent;
- Direct Connect;
- eDonkey;
- FastTrack;
- Yahoo;
- Napster;
- Gnutella;
- AIM;
- Skype;
- Steam;
- Hamachi;
- Radmin VPN;

Конечно же, данный метод не даёт гарантии обнаружения и не позволяет однозначно идентифицировать приложение и тип данных, передающихся по P2P-сети.

6.4.2 Метод анализирования потоков

Данный метод реализуется в функции *find_p2p*:

```
def find_p2p():
    # 1 Заполнение p2p_addrs адресами,
    # взаимодействующими одновременно по TCP и UDP с учётом исключений
    inter = check_intersection()
    for addr in inter:
        if addr not in rejected and addr != UIP:
            p2p_addrs.add(addr)

    # 2 Заполнение p2p_addrs адресами, выбранными исходя из check_p2p с учётом исключений
    for ipport in dict_ipport:
        ipp = dict_ipport[ipport]
        ip = ipp.dst_ip
        port = ipp.dst_port
        if ipp.check_p2p() and check_exceptions(ip, port) and ip != UIP:
            p2p_addrs.add(ip)

    return p2p_addrs
```

Данная функция работает по следующему алгоритму [7]:

Шаг 1. Рассматриваются пары адресов, взаимодействующие одновременно по протоколам *TCP* и *UDP*. Если при этом *TCP/UDP*-порты не входят в список исключений *rejected*, то оба адреса заносятся в массив *p2p_addrs*.

Шаг 2. Для каждой пары адресов из *dict_ipport* проверяется, что IP-адрес не находится в списке исключений и проверяется условие — если массив различных адресов источника содержит более двух адресов, а разница между этим массивом и массивом различных портов источника меньше двух, то считается, что пара принимает участие в P2P-деятельности.

Эти действия проводятся каждые 15 секунд с момента запуска программы.

6.5 Тестирование программы

Для тестирования программы использовался *Skype*. Звонки в данном приложении работают с помощью P2P. На 2 рисунке показана работа программы во время активного аудио звонка в Skype. С помощью метода анализирования портов (в данном примере порт 3480) и потоков был обнаружен P2P-трафик.

ЗАКЛЮЧЕНИЕ

Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 P2P (Peer-to-Peer) [Электронный ресурс]. — URL: https://ru.bmstu.wiki/P2P_(Peer-to-Peer) (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.

2 P2P [Электронный ресурс]. — URL: https://glebradchenko.susu.ru/courses/bachelor/odp/2013/SUSU_Distr_11_P2P.pdf (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.

3 List of well-known ports used by various peer-to-peer (P2P) protocols. [Электронный ресурс]. — URL: https://www.researchgate.net/figure/List-of-well-known-ports-used-by-various-peer-to-peer-P2P-protocols_tbl1_347789431 (Дата обращения 25.05.2022). Загл. с экр. Яз. рус.

4 *Madhukar, A.* A longitudinal study of p2p traffic classification / A. Madhukar, C. Williamson // *Proc. of 14th IEEE International Symposium on Modeling, Analysis, and Simulation*. — 2012.

5 Обнаружение P2P трафика Подробнее: https://www.securitylab.ru/analytics/240496.ph [Электронный ресурс]. — URL: https://www.securitylab.ru/analytics/240496.php (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.

6 *Ngiwlay, W.* Bittorrent peer identification based on behaviors of a choke algorithm / W. Ngiwlay, C. Intanagonwiwat, Y. Teng-amnuay // *Association for Computing Machinery*. — 2008.

7 *Бредихин, С. В.* Диагностика p2p-АКТИВНОСТИ на основе анализа потоков netflow / С. В. Бредихин, Н. Г. Щербакова // *Проблемы информатики*. — 2006.