\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ  Кафедра  «Криптология и кибербезопасность» |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Федеральное государственное автономное образовательное учреждение высшего образования*

**«Национальный исследовательский ядерный университет «МИФИ»»**

ЛАБОРАТОРНАЯ РАБОТА №1:

**«Реализация скрытого канала»**

Аверин Владислав

Группа: Б19-505

Март, 2023

1. **Развертывание демонстрационного стенда**

Для реализации скрытого канала использовались 3 виртуальные машины Kali Linux, созданными средством виртуализации VirualBox, каждая из которых имела интерфейс типа «сетевой мост» со статически заданными адресами из пула 192.168.2.1-254/24. Программный код, реализованный на первой машине (192.168.1.10) выступал в роли Клиента 1, посылающего UDP пакеты различной длины на машину-прокси (192.168.1.12) с постоянным интервалом времени. Код на машине-прокси принимает входящие пакеты и перенаправляет их на хост Клиента 2 (192.168.1.11), подменяя IP-адрес назначения со своего на адрес Клиента 2. Таким образом, было достигнуто минимальное сетевое взаимодействие двух узлов между собой через тривиальное средство защиты (прокси-сервер), которые создают легитимный канал связи (UDP потоки), в который будет встроен скрытый канал.

Помимо обычного перенаправления пакетов, машина-прокси считается «зараженной», и в ней присутствует код, реализующий скрытый канал на основе полей заголовка IP версии 4. Клиент 2 для простоты также считается «зараженным» и в нем имплементирован код для считывания данных передаваемых по скрытому каналу. Все исходные коды в стенде реализованы на языках C/C++ при помощи системной Linux-библиотеки <socket> и используют механизмы Raw Sockets (поэтому для каждой из программ требуется права суперпользователя). Мониторинг трафика производится посредством инструмента Wireshark. Таким образом, функциональная композиция стенда представлена следующим образом:

**Машина-Клиент1 (192.168.1.10)**: генерирует UDP трафик, работая с уровнями до IP включительно (IP + UDP + UDPPayload). Пакеты приходят на прокси на порт 9090. Запуск программы: sudo ./internal

**Прокси-сервер (192.168.1.12):** перенаправляет полученные UDP пакеты от Клиента 1 на Клиента 2 (в штатном режиме работы) на порт 9090. При активации закладки скрытого канала в зависимости от выбранного метода код начинает снифовать приходящий от Клиента 1 UDP трафик и встраивает в проходящие через хост-прокси пакеты скрытое сообщение и пересылая его также Клиенту 2, но на другой порт (9091). Т.е. скрытый канал модифицирует пакеты и дублирует их, но это сделано исключительно в целях лабораторного сравнения исходного и модифицированного скрытым каналом пакета. В действительности же пакеты бы не дублировались, а прокси отправлял от себя только модифицированный трафик.

Запуск программы: sudo ./proxy <*path/filename*> <*method*>

где *filename* – имя передаваемого объекта на прокси-машине, *method* – один из трех строковых аргументов: ‘QOS’, ‘TTL’ и ‘CHECKSUM’.

Передача файла в общем случае происходит в 3 этапа (в случае метода TTL в 4), из соображений того, что закладка и злоумышленник имеют некоторые договоренности, такие как: длина имени файла (в константах исходников данное значение равно 16 байт), а также тип переменной размера файла в байтах (стандартно задано 4 байта, т.е. максимально возможный передаваемый размер 2^32 = 4Гб). Данные параметры можно менять в зависимости от необходимости. Таким образом, передача по скрытому каналу производится следующим образом:

1. Первые 16 байт являются символами имени файла вместе с форматом; в случае меньшего размера передаются нуль-байты (0x00000000), воспринимаемые Си как нуль-терминальный символ (конец строки), и заполняющие оставшиеся символы в буфере-приемнике.
2. Следующие 4 байта – размер файла в байтах; передача производится порциально в порядке BigEndian. Т.е. передача битов от старшего разряда к младшему.
3. Следующие SIZE байт – содержимое файла, открытого в бинарном формате. SIZE определяется на шаге 2. После принятия SIZE байт передача объекта считается завершенной и на стороне злоумышленника, и на стороне закладки.

**Машина-Клиент 2 (192.168.1.11):** предназначена для мониторинга приходящего от Клиента 1 (через прокси) трафика посредством Wireshark, а также на ней внедрен код-злоумышленник, также снифующий порт скрытого канала и дешифрующий скрытое сообщение с тем же регламентом передачи, что описан выше.

Запуск программы: sudo ./external

Данная программа имитирует злоумышленника и создает по полученным данным на хосте Клиента 2 полученный файл.

**Некоторые замечания.**

Изначальная идея использования средств виртуализции в качестве реализации межсетевого взаимодействия заключалась в создании защищенного канала связи между Клиентом 1 и Клиентом 2, которые общались бы напрямую друг с другом без посредников. Но сама локальная сеть работала бы на основе статических маршрутов, маршрутизируя трафик между клиентами через реализованное средство защиты. Однако в результате работы с внутренней сетью VirtualBox такую топологию сети построить не удалось: задание статических маршрутов и разрешение транзитного трафика оказалось недостаточно, Клиент 1 и Клиент 2 все также продолжали общаться в обход узла прокси, так, будто подключены напрямую витой парой. Поэтому от идеи TCP – трафика было решено отказаться (т.к. производить handshake и получать ответы в таком случае через хост прокси невозможно). Для демонстрационного стенда достаточно реализации обычного UDP трафика, чтобы в него внедрить скрытый канал.

Кроме того, сначала в качестве кода Клиента 1 для флудинга использовалась программная утилита Ostinato, которая полностью удовлетворяла заданнным условиям лабораторной (плавающая длина пакетов, фиксированный интервал отправки), однако практика показала, что библиотека socket языка Си может не детектировать некоторые из таких автоматически собранных пакетов. Решить данную проблему не удалось, поэтому для генерации была написана собственная реализация.

Во всех демонстрациях для передачи использовались 2 файла: img.jpeg размером 9032 байта и текстовый файл sender.txt размером 637 байт с текстом: «The man in black fled across the desert, and the gunslinger followed.\nThe desert was the apotheosis of all deserts, huge, standing to the sky for what might have been parsecs in all directions. White; blinding; waterless; without feature save for the faint, cloudy haze of the mountains which sketched themselves on the horizon and the devil-grass which brought sweet dreams, nightmares, death.»



Рисунок 1 – файл img.jpeg

1. **Скрытый канал, основанный на поле QOS заголовка IPv4**

Данная реализация основана на сокрытии передаваемой информации в поле QOS, имеющего размер 1 байт. Последние 2 бита в данном поле являются битами ECN и могут служить для негласной передачи. Однако как было отмечено в условиях лабораторной работы, зачастую маршрутизаторы без должной настройки игнорируют все поле; таким образом, для передачи было использовано все 8 бит, т.е. передача производилась по одному байту в пакете.

Результат работы данного метода по передаче файла img.jpeg представлен ниже:

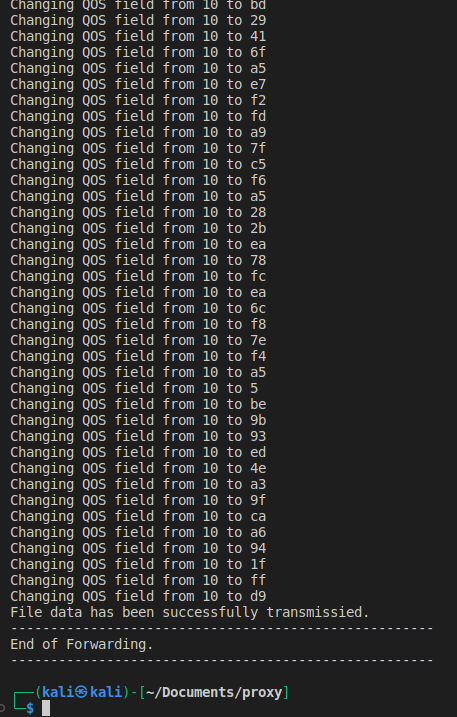
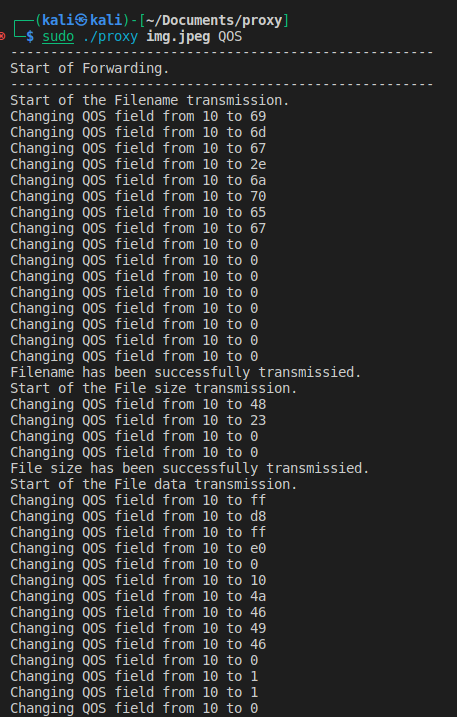


Рисунок 2 – результат работы программы-закладки

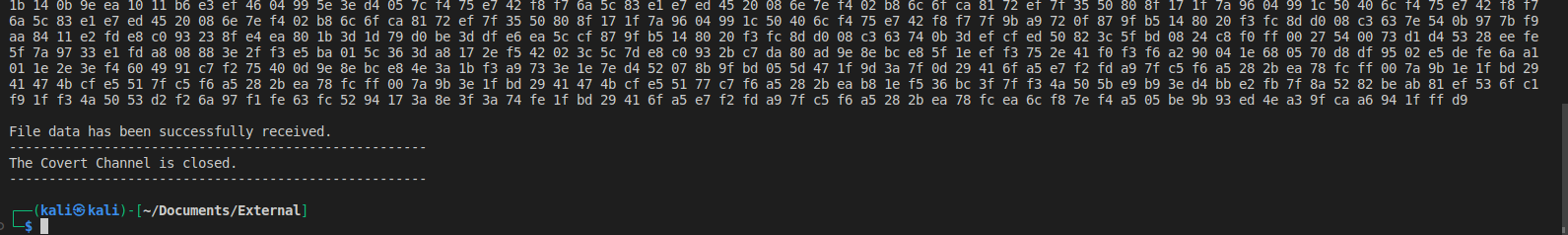
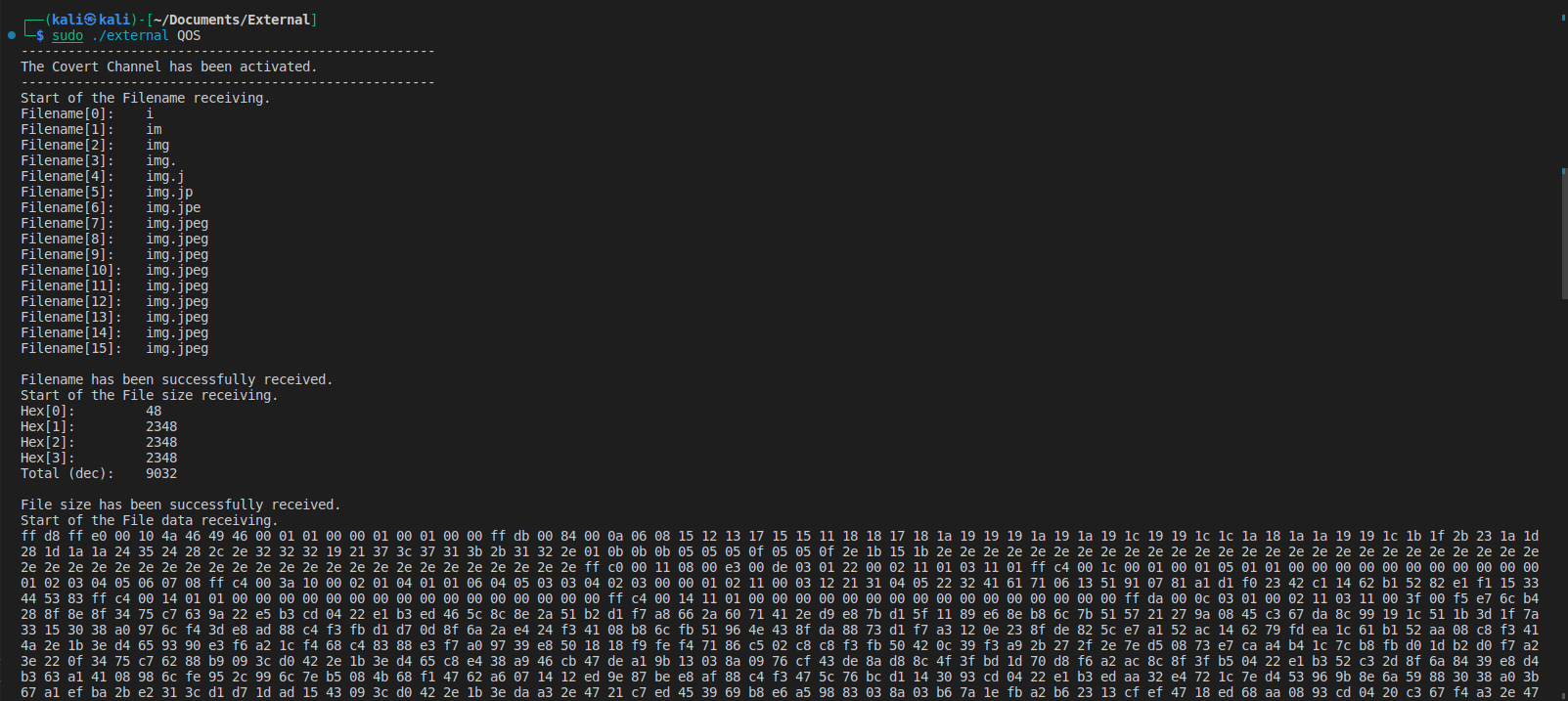


Рисунок 3 – результат работы программы злоумышленника

Далее представлен скриншот дампа Wireshark с примером модифицированного и исходного пакета для сравнения их сигнатуры:

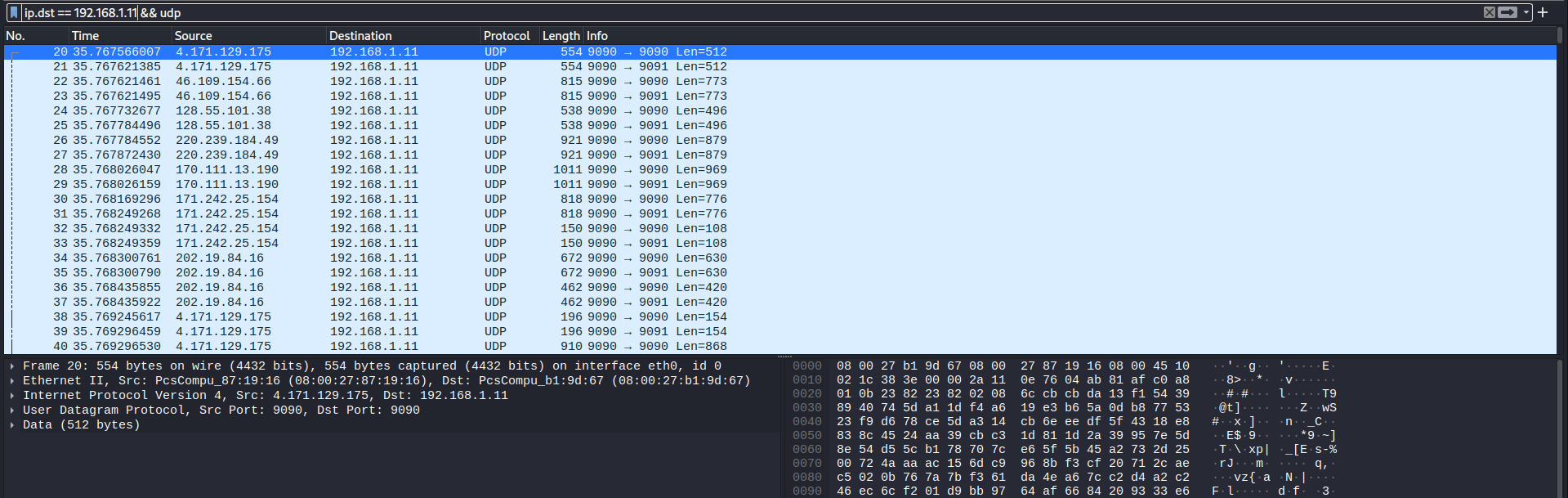


Рисунок 4 – фрагмент дампа Wireshark после передачи сообщения

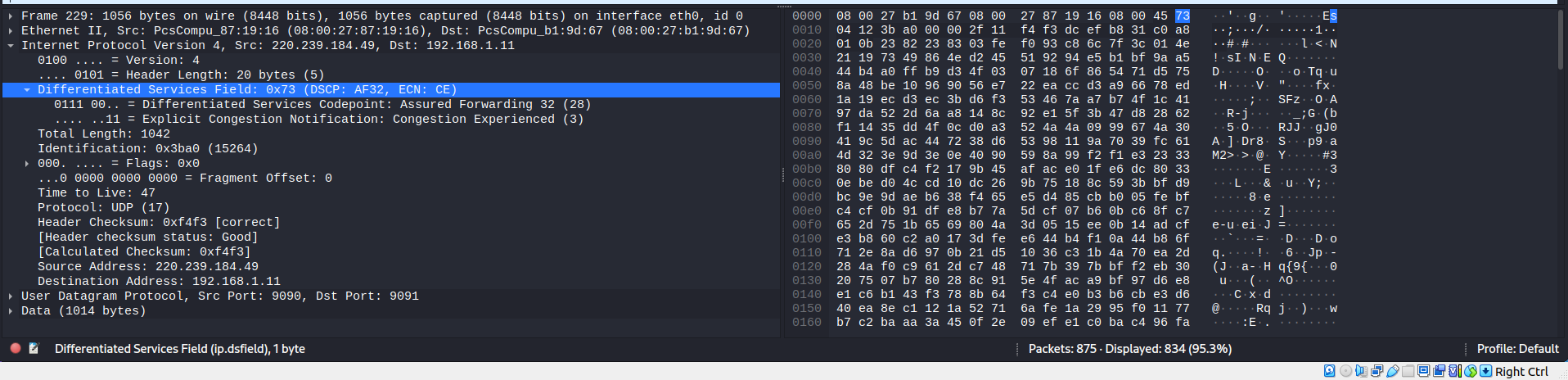
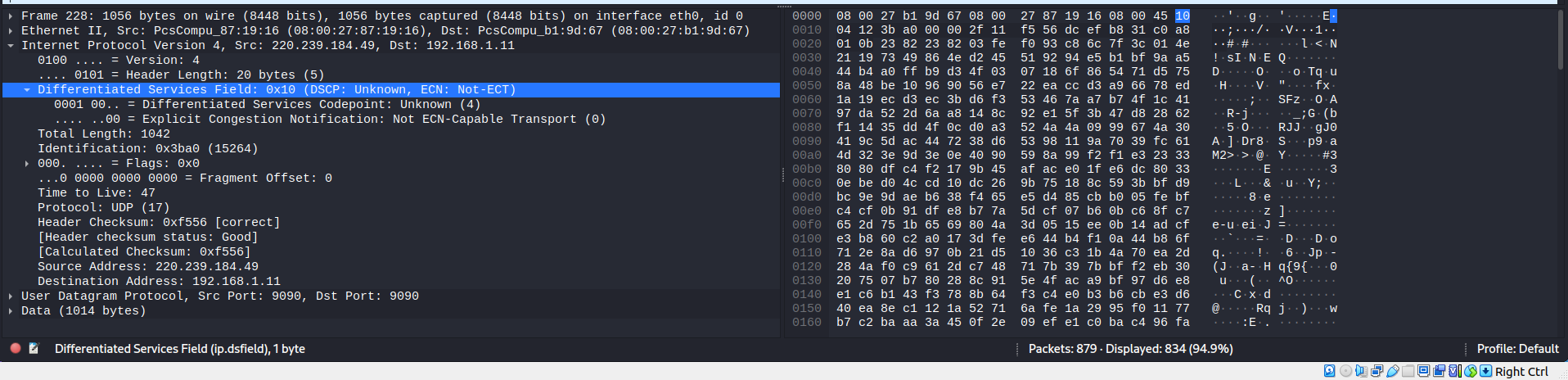


Рисунок 6 – исходный и модифицированный UDP пакеты

Как видно из рисунков выше, в модифицированном пакете, который передает по скрытому каналу байт 0x01110011 изменилось на данное значение поле QOS (с дефолтного заданного Клиентом 1 значения 0x00010000), а также поле Checksum и порт с 9090 на 9091 (данные изменения де-факто не нужны для корректной работы скрытого канала, однако их изменение помогало в отладке, поэтому было оставлено для наглядности)

1. **Скрытый канал, основанный на поле Checksum заголовка IPv4**

Для реализации скрытого канала на основе поля контрольной суммы для скрытой передачи использовался ее последний бит. В зависимости от четности очередного передаваемого бита и текущей суммы поле TTL увеличивалось на 1, чтобы достичь одинаковой четности. Таким образом, для передачи одного бита информации по скрытому каналу использовалось от 1 до 2 битов исходного пакета. Данное увеличение поля TTL не столь заметно и не является чем-то настолько невозможным, чтобы встроенные средства могли определить отклонения в модифицированных заголовках, однако специализированные средства для обнаружения скрытых каналов все равно смогут достаточно эффективно выявить данные аномалии, связанные с полем TTL.

Результат работы данного метода по передаче файла sender.txt (пропускная способность данного метода достаточно мала, при максимальной производительности UDP флудинга она все равно составляла 80 б/сек, поэтому изображение весом 9кб передавалось бы достаточно долго) представлен ниже:

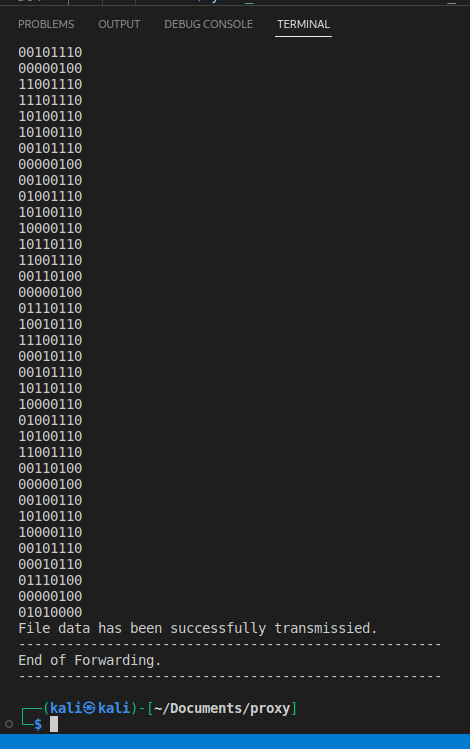
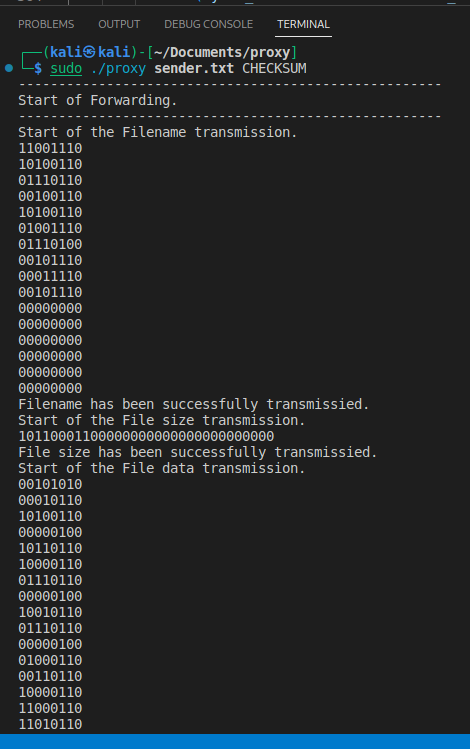


Рисунок 7 – Результат работы программы-закладки

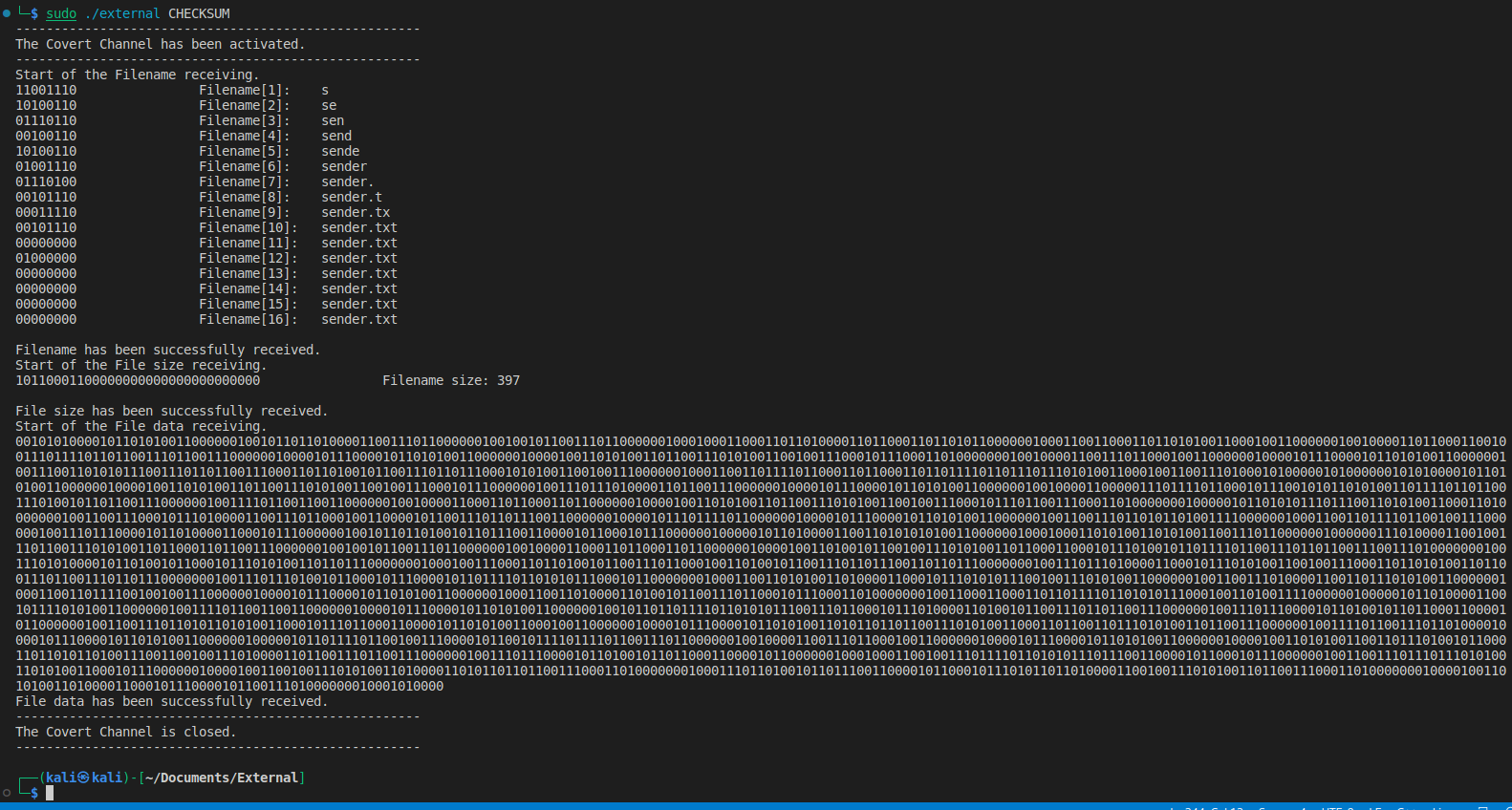


Рисунок 8 – Результат работы программы злоумышленника

Далее представлен скриншот дампа Wireshark с примером модифицированного и исходного пакета для сравнения их сигнатуры:

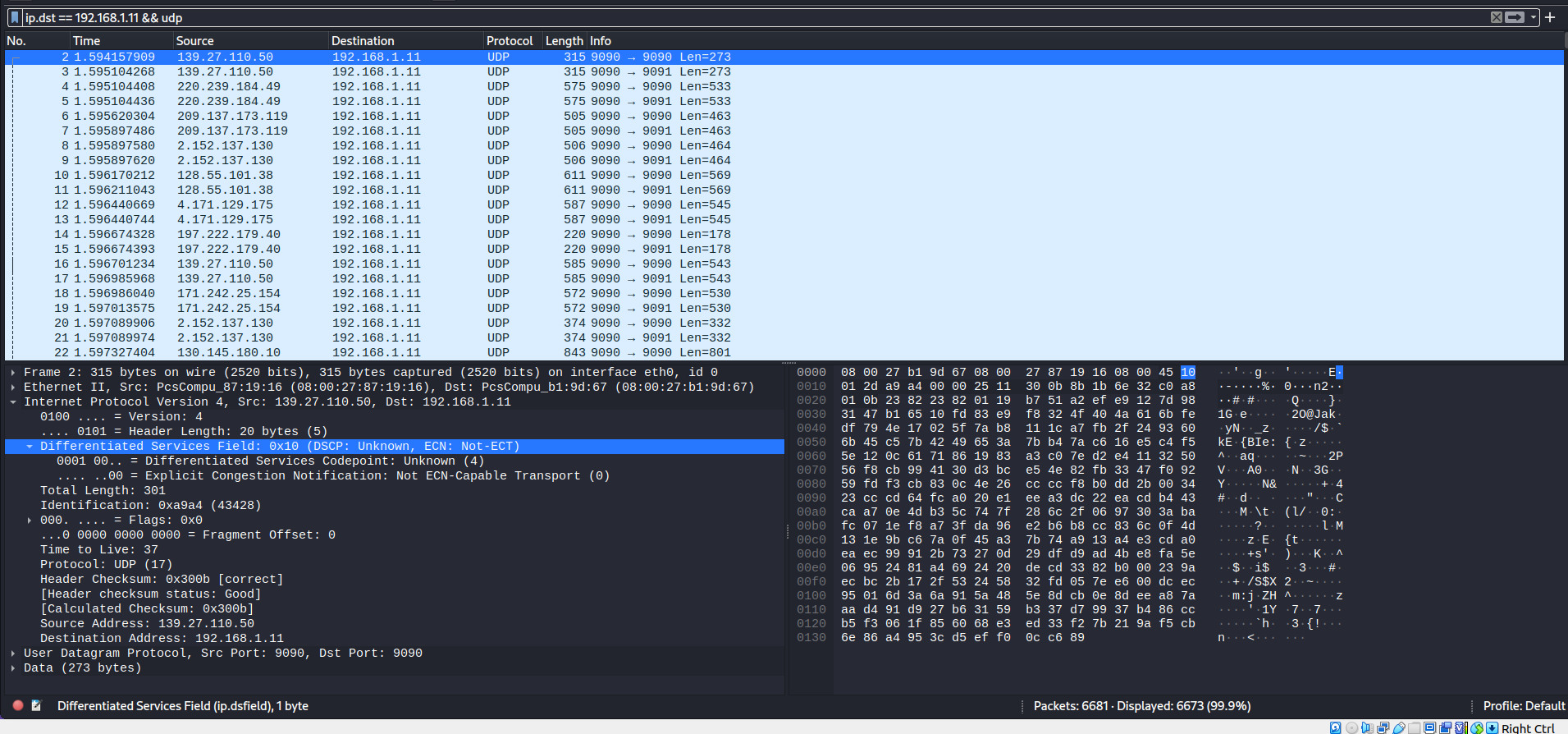


Рисунок 9 – фрагмент дампа Wireshark после передачи сообщения

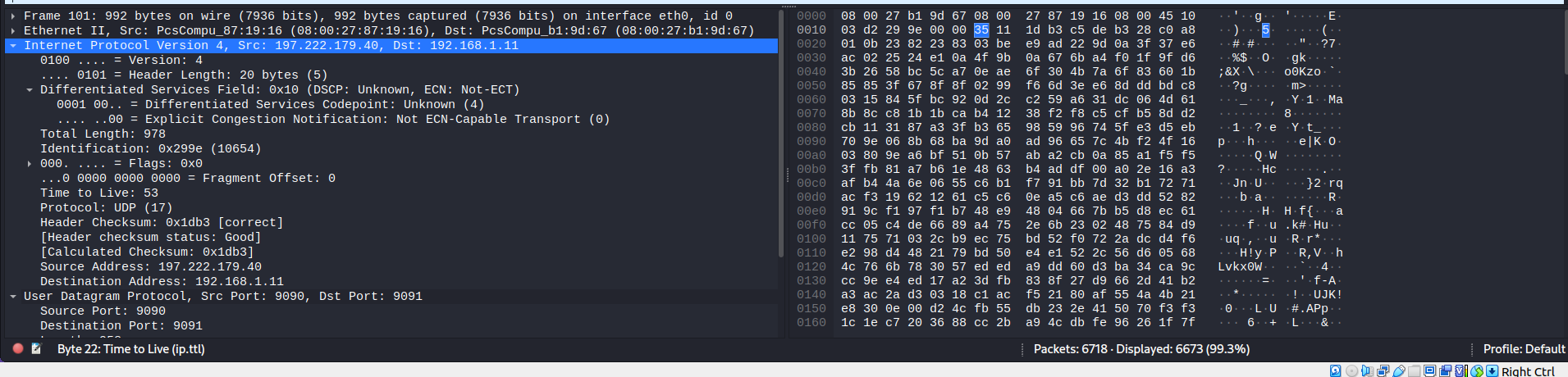
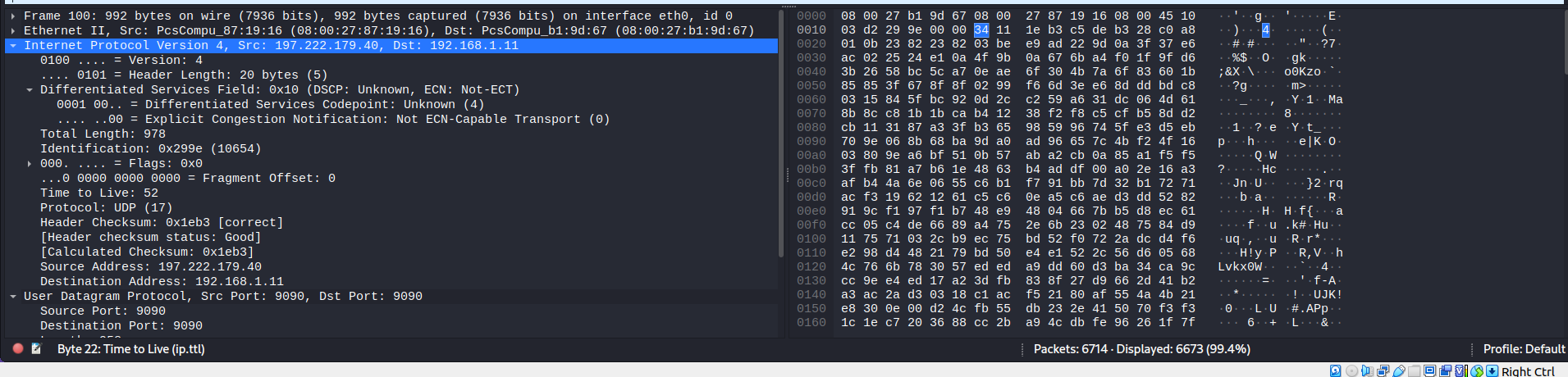


Рисунок 10 – исходный и модифицированный UDP пакеты

Примерная половина пакетов не отличались друг от друга (за исключением номера порта в UDP заголовке), что логично; вероятность того, что передаваемый бит и четность чексуммы пакета совпадут, равна 0,5. Однако как изображено на примере выше, в случае модификации пакета меняет поле TTL (в примере с 0x34 на 0x35) и значение контрольной суммы (в примере с 0x0383 на 0x0382; она инкрементировалась, из-за механизма вычисления контрольной суммы – взятия дополнения к 0xFFFF).

1. **Скрытый канал, основанный на поле TTL заголовка IPv4**

Для более приближенной к реальности ситуации модифицируем исходный код Клиента 1: теперь он генерирует N количество UDP потоков, каждый их который имеет свой собственный IP-адрес источника и константное значение поля TTL. Таким образом, имея некоторый пул значений данного поля программа-закладка и программа-злоумышленник анализируют данные потоки определенное количество времени в пассивном режиме (для упрощения демонстрации вместо времени используется анализ оговоренного заранее количества пакетов, после которого определяется среднее значение TTL. Потом скрытый канал переходит в активный режим, и программа-закладка начинает модифицировать при необходиомсти значение поля в зависимости от значения передаваемого бита: если он равен 0, и в случае, если исходное значение TTL не лежит в рамках [0, average\_tll], она меняет его на average\_tll; если же лежит, то ничего не делает. Аналогично для бита 1. Таким образом, принимающая сторона в виде программы-злоумышленника анализирует модифицированные пакеты и если TTL лежит в границах [0, average\_ttl], то декодирует его как 0, иначе как 1.

Результат работы данного метода по передаче файла sender.txt представлен ниже (пропускная способность данного метода также мала, т.к. пакет несет информацию лишь в 1 бит, поэтому использовался файл малого размера):

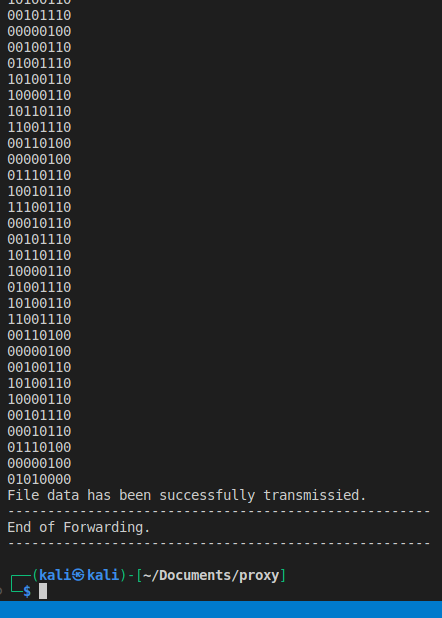
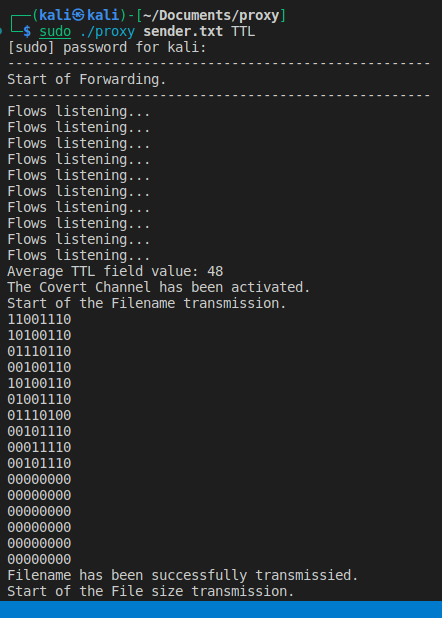


Рисунок 11 – Результат работы программы-закладки

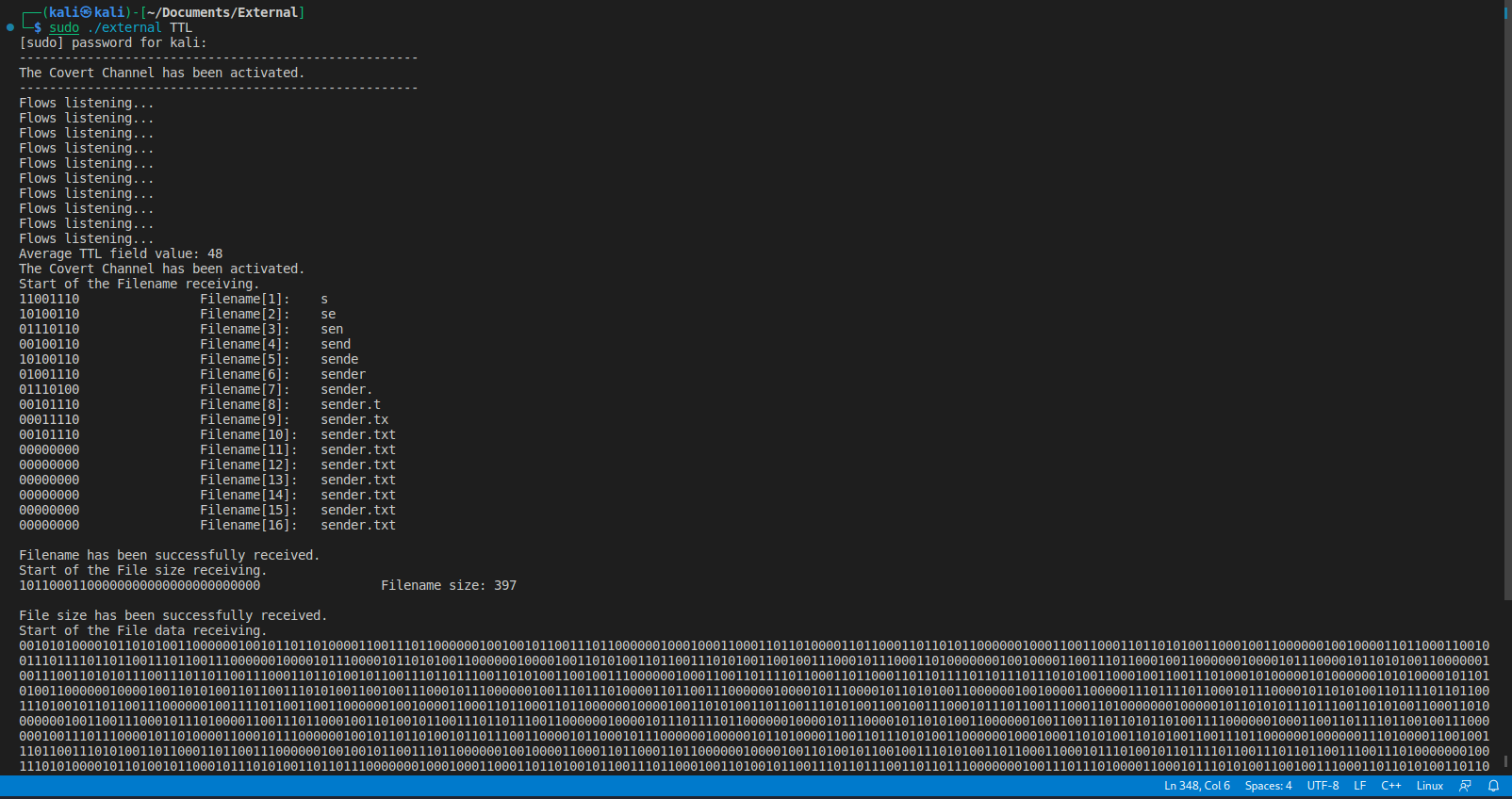


Рисунок 12 – Результат работы программы злоумышленника

Далее представлен скриншот дампа Wireshark с примером модифицированного и исходного пакета для сравнения их сигнатуры:

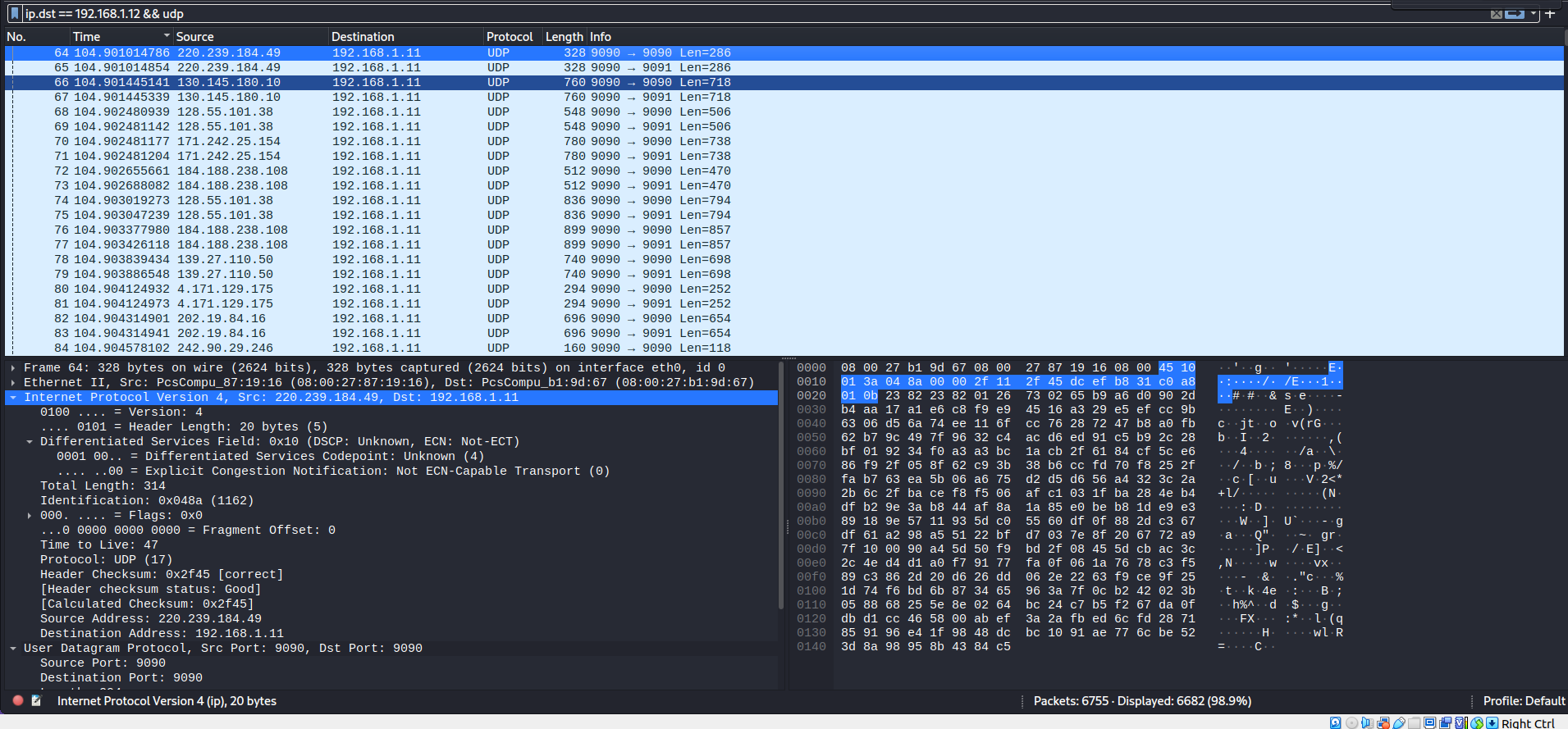


Рисунок 13 – фрагмент дампа Wireshark после передачи сообщения

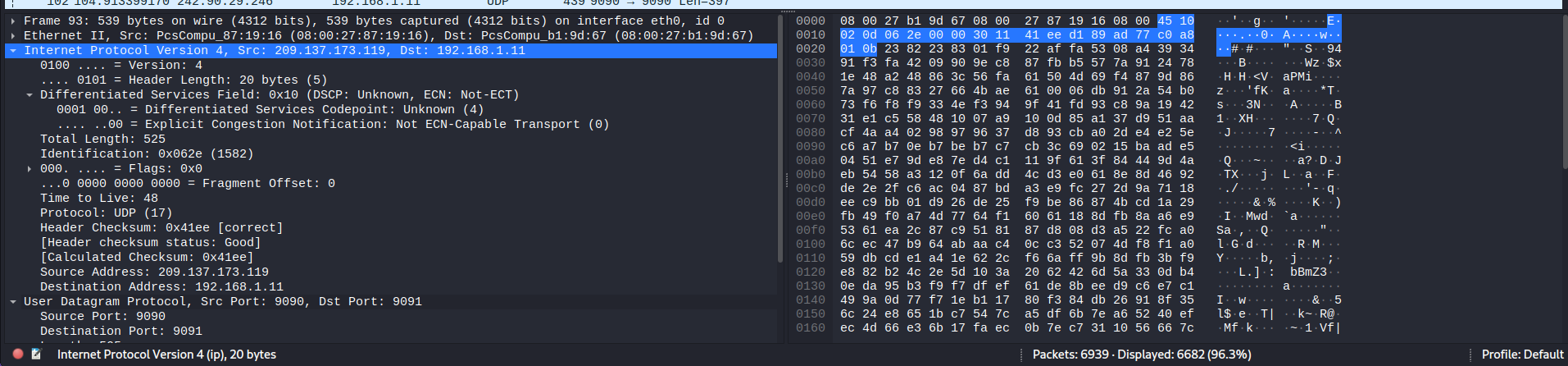
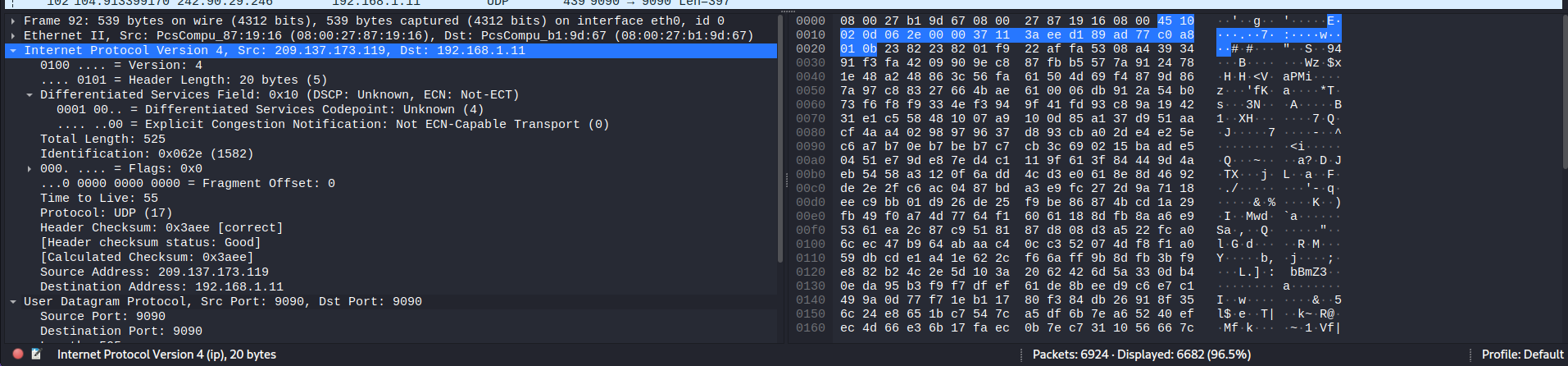


Рисунок 13 – исходный и модифицированный UDP пакеты

Здесь также присутствуют пакеты, не отличающиеся от исходных, в случае, если значение TTL удовлетворяет необходимому значению. Выше приведен пример пакета с измененными полями. Так, TTL поменяло свое значение с 0x37 (55) на 0x30 (48), что является приемлемым, т.к. обычно количество хопов в глобальной сети не превышает 32.

1. **Приложения – разработанные коды**

Исходный код доступен в открытом репозитории по ссылке: https://github.com/Infernalum/Covert-Channels