

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 RDP ТРАФИКА МЕТОДОМ АНАЛИЗА
ЕГО ПОВЕДЕНИЯ**

КУРСОВАЯ РАБОТА

студента 3 курса 331 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Научный руководитель
доцент

Гортинский А. В.

Заведующий кафедрой

Абросимов М. Б.

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Определение RDP	4
1.1 Безопасность протокола RDP.....	4
2 Принцип работы протокола RDP и анализ его поведения	5
3 Обнаружение сеанса удаленного управления с помощью программы.....	10
4 Демонстрация работы программы	18
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	24
Приложение А Код rdp-sniffer.py	26

ВВЕДЕНИЕ

Информация – это сведения об окружающем мире и протекающих в нём процессах, которые зафиксированы на каком-либо носителе. Благодаря протоколам удаленного доступа можно распоряжаться базами данных, информацией, которая хранится на другом устройстве. В недавнем прошлом большинство схем удаленного доступа характеризовалось высокой стоимостью, низкой производительностью, небольшой скоростью передачи данных, недостаточным уровнем защищенности передаваемой информации [1].

Сейчас, когда практически все предприятия перешли на дистанционный формат работы, компании выбирают протокол RDP, так как он прост в настройке и в использовании. Но далеко не все уделяют особое внимание безопасности собственных рабочих мест. Поэтому предприятия могут быть атакованы злоумышленниками.

В данной работе будут разобраны принцип работы RDP, анализ его поведения, а также методы обнаружения данного протокола.

1 Определение RDP

Протокол RDP (от англ. Remote Desktop Protocol — протокол удалённого рабочего стола) — патентованный протокол прикладного уровня компании Microsoft и приобретен ею у другой компании Polycom, который предоставляет пользователю графический интерфейс для подключения к другому компьютеру через сетевое соединение. Для этого пользователь запускает клиентское программное обеспечение RDP, а на другом компьютере должно быть запущено программное обеспечение сервера RDP [2].

Клиенты для подключения по RDP существуют для большинства версий Microsoft Windows, Linux, Unix, macOS, iOS, Android и других операционных систем. Стоит отметить, что RDP-серверы встроены в операционные системы Windows. По умолчанию подключения, созданные с помощью RDP, используют UDP-порт 3389 и TCP порт 3389, по которым осуществляется передача данных.

1.1 Безопасность протокола RDP

Как уже известно, что для операционной системы Windows постоянно выходят различные обновления, включая обновлений RDS (от англ. Remote Desktop Services — службы удаленных рабочих столов). В связи с этим возникают различные уязвимости при инициализации RDP-сессии. В основном они не связаны непосредственно с протоколом RDP, но касаются службы удаленных рабочих столов RDS и позволяют при успешной эксплуатации путем отправления специального запроса через RDP получить возможность выполнения произвольного кода на уязвимой системе, даже не проходя при этом процедуру проверки подлинности. Достаточно лишь иметь доступ к хосту или серверу с уязвимой системой Windows. Таким образом, любая система, доступная из сети Интернет, является уязвимой при отсутствии установленных последних обновлений безопасности Windows.

Если стоит задача защитить удаленный доступ, то, конечно, необходимо использовать надежный пароль, обновить свое программное обеспечение до последней версии, также можно использовать VPN подключение, чтобы получить IP-адрес виртуальной сети и добавить его в правило исключения брандмауэра RDP. Стоит отметить, что существует много разных способов, чтобы защитить подключение с помощью протокола RDP и более подробно это описано в документации Microsoft.

2 Принцип работы протокола RDP и анализ его поведения

Принцип работы RDP базируется на протоколе TCP. Соединение клиент-сервер происходит на транспортном уровне. После инициализации пользователь проходит аутентификацию. В случае успешного подтверждения сервер передает клиенту управление. Стоит отметить, что под понятием слова «клиент» подразумевается любое устройство (персональный компьютер, планшет или смартфон), а «сервер» — удаленный компьютер, к которому оно подключается.

Протокол RDP внутри себя поддерживает виртуальные каналы, через которые пользователю передаются дополнительные функции операционной системы, например, можно распечатать документ, воспроизвести видео или скопировать файл в буфер обмена.

Далее в работе будет описан процесс установки RDP-сессии, во время которой осуществляется захват трафика с помощью одной известной программы Wireshark. С помощью нее можно достаточно подробно рассмотреть структуру сообщений протоколов.

Для начала будет произведено подключение с помощью «Удаленного рабочего стола». Это средство представляет собой встроенную в Windows программу, предназначенную для удалённого доступа. При его использовании предполагается, что пользователь будет подключаться к одному компьютеру с другого устройства, находящегося в той же локальной сети. В качестве клиента и сервера будут выступать компьютеры с операционной системой Windows 10 Professional версии 21H2.

Для подключения к удаленному рабочему столу были заданы статические IP-адреса. Клиенту был присвоен статический IP-адрес 192.168.10.254, а серверу — 192.168.10.229, соответственно маска сети 255.255.255.0. После того, как были заданы IP-адреса, необходимо зайти в настройки Windows, чтобы включить возможность подключения к удаленному рабочему столу. Об этом более подробно описано в статьях [3] и [4]. Далее на сервере был произведен запуск анализа трафика с помощью приложения Wireshark. После подключения к удаленному компьютеру программа-анализатор трафика начала «захватывать» пакеты, как показано на рисунке 1, принадлежащие следующим протоколам:

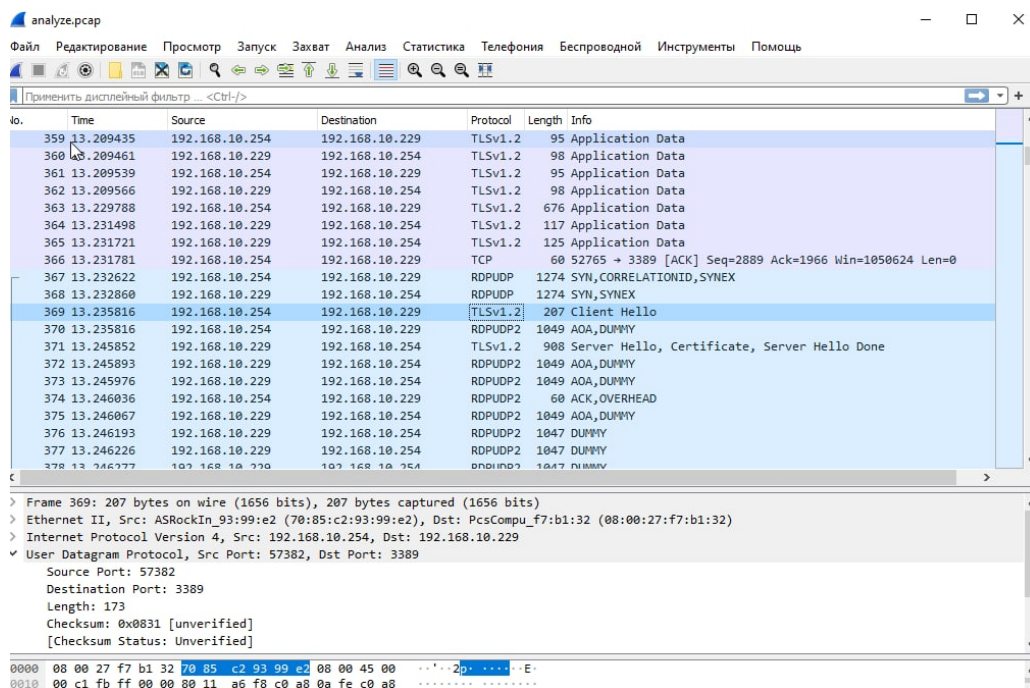


Рисунок 1 – Окно программы Wireshark после захвата трафика

- RDPUDP — протокол RDP, использующий для передачи данных UDP-протокол.
- RDPUDP2 также относится к протоколу RDP. Он был разработан для повышения производительности сетевого соединения по сравнению с соответствующим соединением RDP-UDP [8].
- TLSv1.2 — протокол защиты транспортного уровня, обеспечивающий защищенную передачу между узлами в сети интернет. В данном случае обеспечивает безопасность RDP-сессии.

Во время работы программы Wireshark было найдено достаточное количество пакетов, принадлежащих RDP, которые содержат в себе достаточно интересную информацию. Поэтому стоит рассказать о том, как происходит стандартный способ защиты RDP. Это можно представить в несколько этапов:

1. Клиент объявляет серверу о своем намерении использовать стандартный протокол RDP.
2. Сервер соглашается с этим и отправляет клиенту свой собственный открытый ключ, полученный при шифровании алгоритмом RSA, а также некоторую строку случайных байтов (обычно её называют «random сервером»), генерируемую сервером. На рисунке 2 можно увидеть запись random сервера.

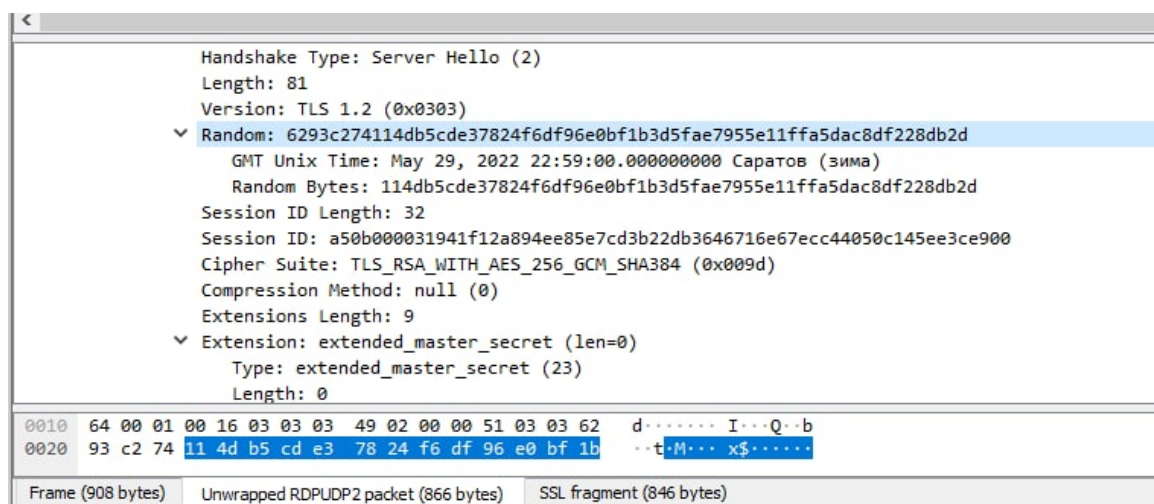


Рисунок 2 – Содержимое пакета, посылаемого от сервера клиенту (запись random сервера)

Совокупность открытого ключа и некоторая строка случайных байтов называется «сертификатом». Данная запись изображена на рисунке 3.

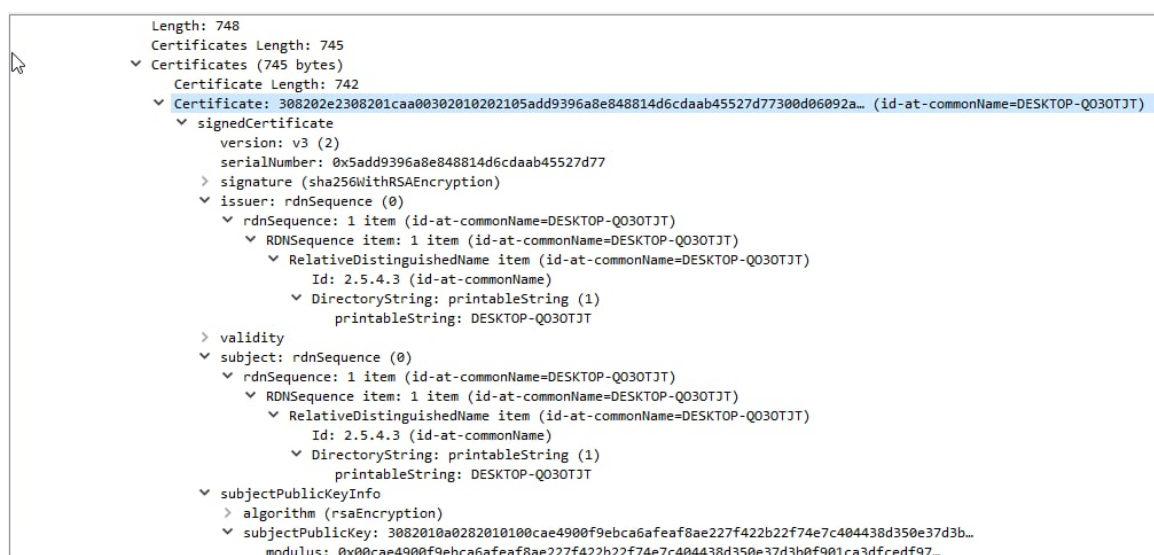


Рисунок 3 – Содержимое пакета, посылаемого от сервера клиенту (запись сертификата)

Сертификат подписывается службой терминалов, например, RDS, с использованием закрытого ключа для обеспечения подлинности.

3. Теперь клиент посылает некоторую строку случайных байтов, которая называется «premaster secret», показанная на рисунке 4.

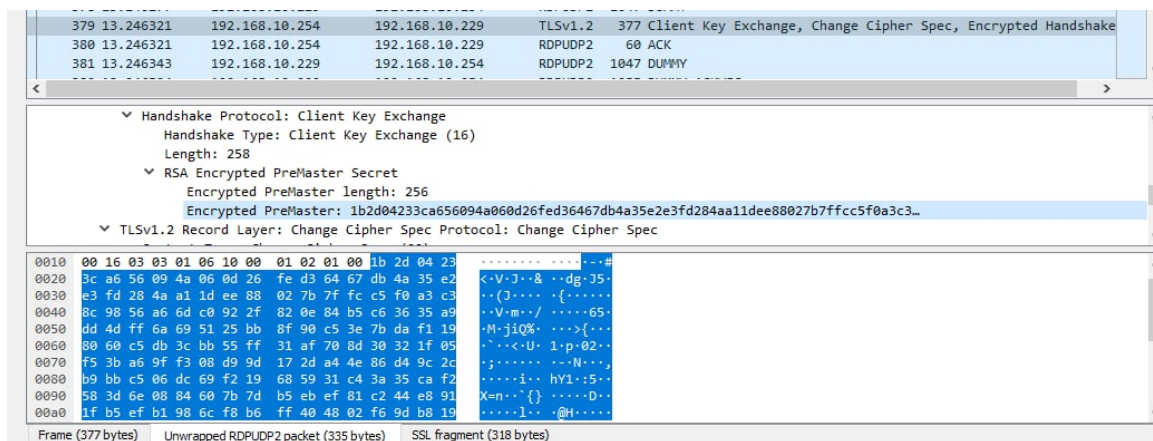


Рисунок 4 – Содержание пакета, посылаемого от клиента серверу (запись premaster secret)

Данная запись шифруется открытым ключом, которая может быть расшифрована сервером только с помощью закрытого ключа службы терминалов.

4. Сервер расшифровывает premaster secret с помощью собственного закрытого ключа.
5. В случае успеха клиент и сервер получают свои сеансовые ключи из random сервера и premaster secret. Далее они используются для симметричного шифрования остальной части сеанса.

После того как был произведен разбор RDP-сессии в некоторых пакетах можно заметить сообщения, принадлежащие протоколу TLS. На рисунке 5 видно, что в раскодированной последовательности чисел имеется IP-адрес сервера.

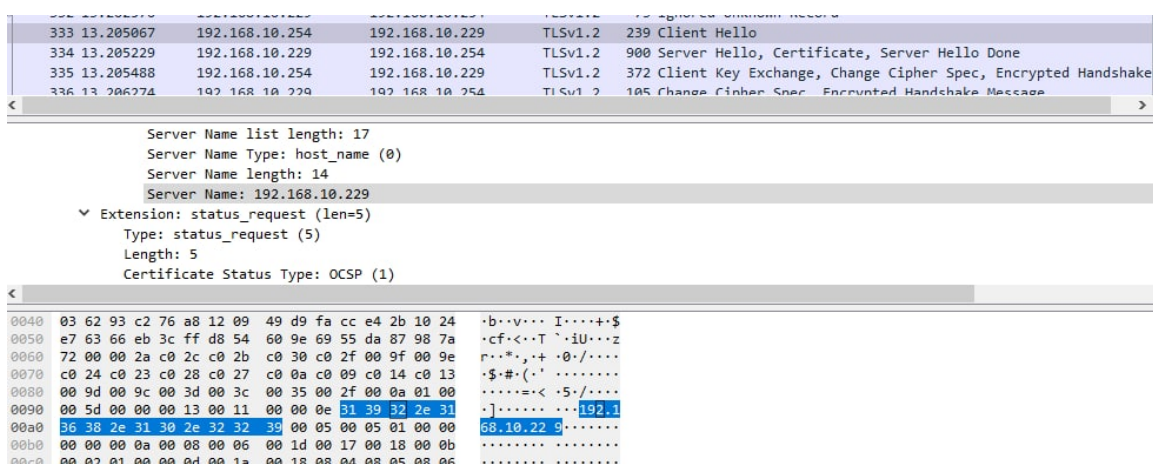


Рисунок 5 – Данные пакета, посылаемого клиентом

А следующим за ним идет пакет, в котором можно заметить имя сервера, как показано на рисунке 6.

334	13.205229	192.168.10.229	192.168.10.254	TLSv1.2	900 Server Hello, Certificate, Server Hello Done
335	13.205488	192.168.10.254	192.168.10.229	TLSv1.2	372 Client Key Exchange, Change Cipher Spec, Encrypted Handshake
336	13.206274	192.168.10.229	192.168.10.254	TLSv1.2	105 Change Cipher Spec, Encrypted Handshake Message

printableString: DESKTOP-Q030TJT				
> validity > subject: rdnSequence (0) > rdnSequence: 1 item (id-at-commonName=DESKTOP-Q030TJT) > RDNSequence item: 1 item (id-at-commonName=DESKTOP-Q030TJT) > RelativeDistinguishedName item (id-at-commonName=DESKTOP-Q030TJT) Id: 2.5.4.3 (id-at-commonName) > DirectoryString: printableString (1) printableString: DESKTOP-Q030TJT				

070	da 29 3c a3 07 28 45 d1 3f e3 50 74 17 4c 00 28	.)<..(E. ?Pt.L.(
080	31 14 00 9d 00 00 09 00 17 00 00 ff 01 00 01 00	1.....
090	0b 00 02 ec 00 02 e9 00 02 e6 30 82 02 e2 30 820..0.
0a0	01 ca a0 03 02 01 02 02 10 5a dd 93 96 a8 e8 48Z.....H
0b0	81 4d 6c da ab 45 52 7d 77 30 0d 06 09 2a 86 48	.Ml..ER} w0...*.H
0c0	86 f7 0d 01 01 0b 05 00 30 1a 31 18 30 16 06 030.1.0...
0d0	55 04 03 13 0f 44 45 53 4b 54 4f 50 2d 51 4f 33	U....DESKTOP-Q03
0e0	4f 54 4a 54 30 1e 17 0d 32 32 30 35 32 37 31 39	0TJ10...22052719
0f0	33 30 30 37 5a 17 0d 32 32 31 31 32 36 31 39 33	3007Z..2 21126193
100	30 30 37 5a 30 1a 31 18 30 16 06 03 55 04 03 13	007Z0.1.0...U...
110	0f 44 45 53 4b 54 4f 50 2d 51 4f 33 4f 54 4a 54	.DESKTOP-Q030TJT

Рисунок 6 – Данные пакета, посылаемого сервером

Такие пакеты были перехвачены в момент установки RDP-сессии, в то время, когда клиент успешно ввел пароль и его сертификат был одобрен сервером. Т.е. по данным сообщениям можно понять, что к устройству с IP-адресом 192.168.10.229 было совершено подключение через 3389-й порт. Значит в этот момент времени начиналась установка RDP-сессии.

Осталось понять, как обнаружить подключение к удаленному рабочему столу по протоколу RDP.

3 Обнаружение сеанса удаленного управления с помощью программы

Одним из методов выявления сообщений, передаваемых по сети, является сниффер — это программное обеспечение, которое анализирует входящий и исходящий трафик с компьютера, подключенного к интернету. Для данной работы была написана на языке Python программа, перехватывающая трафик сети, которая также может обнаружить подключение по протоколу RDP.

Данный сниффер принимает пакеты четвертой версии интернет-протокола, пакеты IPv4, содержащие в поле данных сообщения протоколов других уровней. В этом случае здесь будут рассматриваться сообщения протоколов транспортного уровня, в частности TCP и UDP.

При запуске программы появится запрос на выбор «прослушиваемого» порта. По умолчанию будет задан RDP-порт 3389. Далее, для того чтобы успешно перехватить пакет, необходимо установить неразборчивый режим на сетевой интерфейс, чтобы сетевая плата принимала все пакеты независимо от того, кому они адресованы. Данный выбор зависит от способа подключения устройства к сети. Например, в Linux есть виртуальный интерфейс («lo»), который ваш компьютер использует для связи с самим собой, также существуют интерфейсы относящиеся к проводному соединению («enp0s3») и беспроводному («wlp2s0»). В данном случае все устройства будут подключены к сети через Ethernet. Затем программа должна обратиться к файлу «white-list.log», в котором содержится информация об устройствах, находящиеся в одной локальной сети. Допустим в файл не был записан один компьютер, тогда в случае создания подключения по RDP-протоколу программа посчитает его неизвестным и запишет в файл «information.log», в котором будет содержаться вся информация о посторонних попытках создания RDP-сессии.

Функция `get_white_list()` формирует пары: имя и IP-адрес компьютера. К каждой такой паре сопоставляется пара записи имени и IP-адреса устройства в шестнадцатеричной системе счисления. Этот список нужен для анализа данных, хранящихся в TCP-сегменте.

```
# Получение списка верифицированных устройств
def get_white_list():
    f = open('white-list.log', 'r')
    while True:
        line = f.readline().replace('\n', '')
```

```

if '#' in line:
    continue
if not line:
    break
pos = line.find('::')
serv_name = line[:pos]
serv_ip = line[pos + 2:]
white_list[(serv_name, serv_ip)] = (convert_string(serv_name), convert_string(serv_ip))

# Форматирование строки в hex-код
def convert_string(string):
    s = ''
    for el in bytearray(string.encode('utf-8')):
        s += '\\\\' + str(hex(el))[1:]
    return s

```

Функция `write_to_file()` выполняет роль записи в файл «`information.log`» сообщений в зависимости от случая подключения по протоколу RDP. При демонстрации работы программы будет видно, какая информация в нем находится.

```

# Запись в файл
def write_to_file(tup, bl):
    try:
        time = str(datetime.datetime.now()).split('.')[0]
        with open('information.log', 'a+') as f:
            if bl:
                f.write('Было совершено подключение: ' + time)
                f.write( '\\nIP адрес неизвестного клиента: ' + str(tup[0]) +
                    ' MAC-адрес: ' + tup[1] )
                f.write( '\\nПодключение к ПК ' + tup[2] + ' от порта ' + str(tup[3]) +
                    ' к ' + str(tup[4]) + '\\n' )
            else:
                f.write('Время попытки подключения: ' + time)
                f.write( '\\nIP адрес неизвестного клиента: ' + str(tup[0]) +
                    ' MAC-адрес: ' + tup[1] )
                f.write( '\\nПодключение осуществлялось к ПК ' + tup[2] + ' от порта ' +
                    str(tup[3]) + ' к ' + str(tup[4]) + '\\n' )
        f.close()
    except:
        pass

```

Для анализа трафика в сети был создан сокет — программный интерфейс для обеспечения обмена данными между процессами. В силу заданных параметров он получал пакеты, представленные в виде некоторой последовательности чисел, записанных в шестнадцатеричной системе счисления.

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

Чтобы раскодировать данную последовательность чисел, необходимо обратиться к структуре пакета. Для начала нужно рассмотреть кадр Ethernet, представленный на рисунке 7.



Рисунок 7 – Структура Ethernet кадра

Стоит отметить, что в данном заголовке нужно раскодировать 14 байт, 12 из которых MAC-адреса получателя и отправителя и 2 байта, идентифицирующие протокол сетевого уровня. К примеру 0x0800 – IPv4, 0x86DD – IPv6 и т.д. С помощью функций `get_ethernet_frame()` и `get_mac_addr()` производится получение всех необходимых данных заголовка Ethernet.

```
# Получение ethernet-кадра
def get_ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)

# Получение MAC-адреса
def get_mac_addr(mac_bytes):
    mac_str = ''
    for el in mac_bytes:
        mac_str += format(el, '02x').upper() + ':'
    return mac_str[:len(mac_str) - 1]
```

После получения информации об Ethernet кадре идет раскодирование интернет-протокола. В данной работе будут рассматриваться пакеты протокола IPv4, так как для обнаружения RDP-сессии этого вполне достаточно. Поэтому необходимо рассмотреть IPv4-заголовок, который показан на рисунке 8.

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса				16 бит Общая длина																			
		PR	D	T	R				3 бита Флаги			13 бит Смещение фрагмента													
16 бит Идентификатор пакета										D			M												
8 бит Время жизни		8 бит Протокол верхнего уровня				16 бит Контрольная сумма																			
32 бита IP-адрес источника																									
32 бита IP-адрес назначения																									
Параметры и выравнивание																									

Рисунок 8 – Структура IPv4-заголовка

Обычно длина заголовка IP равна 20 байт, т.е. пять 32-битных слов, однако при увеличении объема служебной информации эта длина может быть увеличена за счет использования дополнительных байт в поле параметров и выравниваний. Благодаря полю, где содержится длина заголовка, можно правильно раскодировать оставшуюся последовательность байт. С помощью функций `get_ipv4_data()` и `ipv4_dec()` будут получена информация о времени жизни текущего пакета, о номере транспортного протокола, об IP-адресах отправителя и получателя.

```
# Получение IPv4-заголовка
def get_ipv4_data(data):
    version_header_length = data[0]
    header_length = (version_header_length & 15) * 4
    ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
    return ttl, proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]

# Получение IP-адреса формата X.X.X.X
def ipv4_dec(ip_bytes):
    ip_str = ''
    for el in ip_bytes:
        ip_str += str(el) + '.'
    return ip_str[:-1]
```

После того, как был получен номер транспортного протокола, можно раскодировать их данные. Как уже упоминалось ранее, в качестве транспортных

протоколов будут рассматриваться TCP и UDP протоколы. На рисунке 9 изображена структура tcp-заголовка.

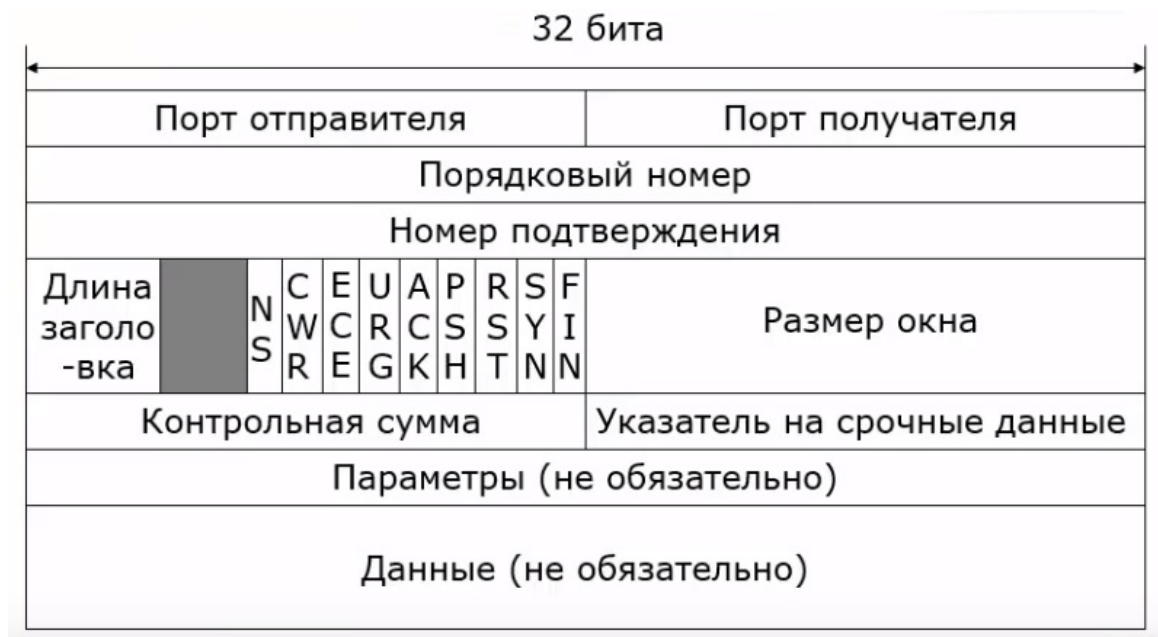


Рисунок 9 – Структура TCP-заголовка

С помощью функции `get_tcp_segment()` производится получение информации, содержащейся в TCP-протоколе. Получается, теперь программе известны порт получателя, порт отправителя, порядковый номер, номер подтверждения и флаги. Однако самая ценная информация для данной работы — это порты и данные, которые содержатся в TCP- и UDP-заголовках.

```
# Получение TCP-сегмента данных
def get_tcp_segment(data):
    src_port, dest_port, sequence, ack, some_block = struct.unpack('!HHLLH', data[:14])
    return src_port, dest_port, sequence, ack, data[(some_block >> 12) * 4:]
```

Аналогично получается раскодирование данных UDP-заголовка с помощью функции `get_udp_segment()`.

```
# Получение UDP-сегмента данных
def get_udp_segment(data):
    src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
    return src_port, dest_port, size, data[8:]
```

Далее необходимо рассмотреть данные, полученные после обработки TCP- и UDP-заголовков. В функции `scan_port()` проверяется порт очередного пакета.

Если порт совпадает с портом, заданным в начале программы (по умолчанию он равен 3389), то значит осуществляется попытка подключения к удаленному рабочему столу устройства, находящегося в текущей локальной сети. Также здесь осуществляется проверка известных IP-адресов, взятых из файла white-list.log.

```
# Проверка порта по-умолчанию
def scan_port(src_ipv4, dest_ipv4, src_mac, src_port, dest_port):
    if dest_port == def_port:
        fl = False
        for key in white_list.keys():
            if key[1] == src_ipv4:
                fl = True
                break
        if not fl:
            for key in white_list.keys():
                if key[1] == dest_ipv4:
                    tup = (src_ipv4, src_mac)
                    if tup not in black_list:
                        black_list.append(tup)
                        write_to_file((src_ipv4, src_mac, key[0], src_port, dest_port), False)
```

В функции scan_inf() производится анализ данных сегмента TCP, а format_data() — функция для корректного представления полученных данных.

```
# Проверка данных TCP-сегмента
def scan_inf(r_data, src_ipv4, dest_ipv4, src_mac, dest_mac, dest_port, src_port):
    global Current_object
    global black_list
    global Packet_cnt
    data = format_data(r_data)
    flag = False
    for key in white_list.keys():
        if key[1] == src_ipv4:
            flag = True
            break
    if not flag:
        for key, value in white_list.items():
            if value[1] in data and key[1] == dest_ipv4:
                Current_object = (key[0], key[1], value[0])
                tup = (src_ipv4, src_mac)
                if tup not in black_list:
                    black_list.append(tup)
                    write_to_file(( src_ipv4, src_mac, Current_object[0]
                                   , src_port, dest_port ), False)
    if Current_object:
        if Current_object[2] in data:
            for key in white_list.keys():
```

```

        if key[1] == src_ipv4:
            write_to_file(( dest_ipv4, dest_mac, Current_object[0]
                           , src_port, dest_port ), True)

            break
        Current_object = ''
    else:
        Packet_cnt += 1
        if Packet_cnt > 100:
            Packet_cnt = 0
            Current_object = ''

```

После вызова функции `get_tcp_segment()` могут быть получены данные, которые уже относятся к прикладному уровню. И такую последовательность чисел можно раскодировать IP-адрес и имя компьютера, показанных на рисунках 5 и 6.

Практически все выше перечисленные функции содержатся в `start_to_listen()`, где производится вывод в консоль информации о получаемых пакетах, делаются вызовы функций для раскодирования последовательности чисел.

```

# Перехват трафика и вывод информации в консоль
def start_to_listen(interface):
    global Current_object
    global Cur_number
    os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
    server = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
    # server.bind((socket.if_indextoname(interface), 0))
    while True:
        # Получение пакетов в виде набора hex-чисел
        raw_data, _ = server.recvfrom(65565)
        dest_mac, src_mac, protocol = get_ethernet_frame(raw_data)

        # Если это интернет-протокол четвертой версии
        if protocol == 8:
            print(f'-----Пакет N{Cur_number}-----')
            Cur_number += 1
            print('Ethernet кадр: ')
            print('MAC-адрес отправителя: ' + str(src_mac), 'MAC-адрес получателя: ' + str(dest_mac))
            ttl, proto, src_ipv4, dest_ipv4, data_ipv4 = get_ipv4_data(raw_data[14:])
            print('IPv4 заголовок:')
            print( 'TTL: ' + str(ttl)
                  , 'Номер протокола: ' + str(proto)
                  , 'IP-адрес отправителя: ' + str(src_ipv4)
                  , 'IP-адрес получателя: ' + str(dest_ipv4))

            # Если это UDP-протокол
            if proto == 17:

```



```

src_port_udp, dest_port_udp, size, data_udp = get_udp_segment(data_ipv4)
print('UDP заголовок:')
print( 'Порт отправителя: ' + str(src_port_udp), 'Порт получателя: ' +
      str(dest_port_udp), 'Размер: ' + str(size) )

scan_port(src_ipv4, dest_ipv4, src_mac, src_port_udp, dest_port_udp)
# Если это TCP-протокол
if proto == 6:
    src_port_tcp, dest_port_tcp, sequence, ack, data_tcp = get_tcp_segment(data_ipv4)
    print('TCP заголовок:')
    print( 'Порт отправителя: ' + str(src_port_tcp)
          , 'Порт получателя: ' + str(dest_port_tcp)
          , 'Порядковый номер: ' + str(sequence)
          , 'Номер подтверждения: ' + str(ack) )

    scan_port(src_ipv4, dest_ipv4, src_mac, src_port_tcp, dest_port_tcp)

th_inf = threading.Thread(target=scan_inf, args=[ data_tcp
                                                , src_ipv4
                                                , dest_ipv4
                                                , src_mac
                                                , dest_mac
                                                , dest_port_tcp
                                                , src_port_tcp ])

th_inf.start()
keyboard.add_hotkey('Space', wait_key)

```

После описания всех главных функций можно перейти к проверке корректности работы программы.

4 Демонстрация работы программы

Допустим к локальной сети подключено несколько устройств. Для тестирования данной программы было запущено четыре виртуальных машины:

- компьютер №1 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.229
- компьютер №2 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.254
- компьютер №3 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.21
- компьютер №4 с операционной системой Linux Ubuntu версии 22.04 LTS IP-адресом 192.168.10.107

В ходе работы был создан файл white-list.log в который записаны имена компьютеров и их IP-адреса. Содержимое файла показано на рисунке 10.

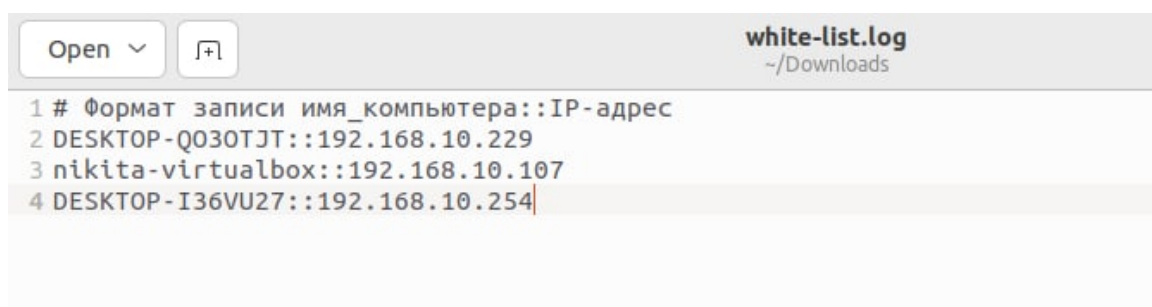


Рисунок 10 – Содержимое файла white-list.log

Стоит отметить, что информация о компьютере №3 не добавлена специально, так как его будут считать неизвестным устройством, остальные, информация о которых записана в файл, являются известными соответственно.

Теперь необходимо провести несколько тестирований данной программы:

1. Пусть компьютер №3 выполнит подключение к компьютеру №1, где в это время на компьютере №4 будет уже работать программа rdp-sniffer.py.

Из рисунка 11 видно, как программа запрашивает порт по умолчанию, а затем предоставляет выбор сетевого интерфейса.

```
nikita@nikita-VirtualBox: ~/Downloads
nikita@nikita-VirtualBox:~/Downloads$ sudo python3 rdp-sniffer.py
[sudo] password for nikita:

Запуск программы...

Хотите поменять RDP порт для анализа трафика? (по умолчанию 3389)
Если да, то нажмите 1, иначе - 0

Выберите сетевой интерфейс, нажав соответствующую цифру:
[(1, 'lo'), (2, 'enp0s3')]
2
```

Рисунок 11 – Вид консоли при запуске программы

После осуществления подключения к удаленному рабочему столу в консоли начали появляться записи о пакетах, которые относятся к протоколу RDP, как показано на рисунке 12.

```
nikita@nikita-VirtualBox: ~/Downloads

-----Пакет N482-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883107 Номер подтверждения: 0
-----Пакет N483-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:F7:B1:32 MAC-адрес получателя: 08:00:27:D8:18:8D
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.229 IP-адрес получателя: 192.168.10.21
TCP заголовок:
Порт отправителя: 3389 Порт получателя: 61088 Порядковый номер: 460425162 Номер подтверждения: 3471883108
-----Пакет N484-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883108 Номер подтверждения: 460425163
-----Пакет N485-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883108 Номер подтверждения: 460425163
```

Рисунок 12 – Вид консоли при работе программы

Видно, что пакеты доставляются через TCP-порт 3389 протокола RDP.

Теперь осталось проверить появились ли какие-нибудь записи в файле information.log.

На рисунке 13 изображены 3 записи, одна из которых была сделана из-за обнаружения 3389-го порта, а две записи — при раскодировании данных протокола TLS. При анализе трафика с помощью программы Wireshark также наблюдались повторяющиеся пакеты протокола TLS, в которых содержалась практически одна и та же информация. Скорее всего это связано с тем, что устройства в процессе обмена данными «договорились» между собой, например, об изменении шифра или порта. Поэтому такого рода пакеты приходится отправлять повторно. Так как программа отслеживает все такие пакеты, то в файл было сделано две записи в разные моменты времени.

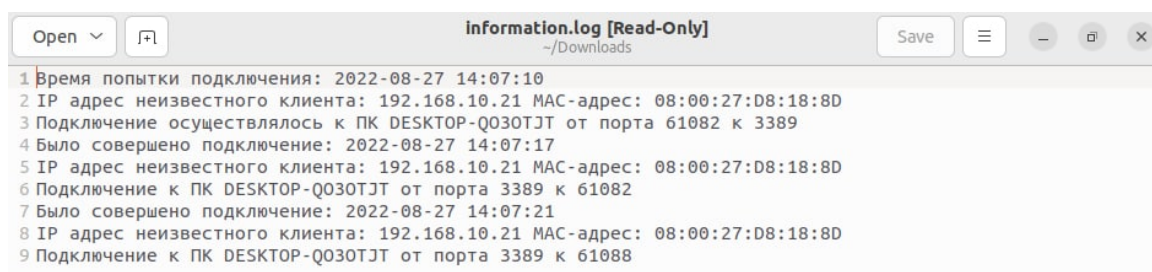


Рисунок 13 – Содержимое файла information.log после подключения по порту 3389

Таким образом благодаря файлу information.log можно узнать время установки RDP-сессии, IP-адрес и MAC-адрес неизвестного клиента

2. Допустим будет совершено подключение по другому RDP порту или в программе ввести совсем другой порт, по которому будет проводится анализ пакетов. Пусть Компьютер №3 будет подключаться к компьютеру №1 по порту, например 13389, как показано на рисунке 14.

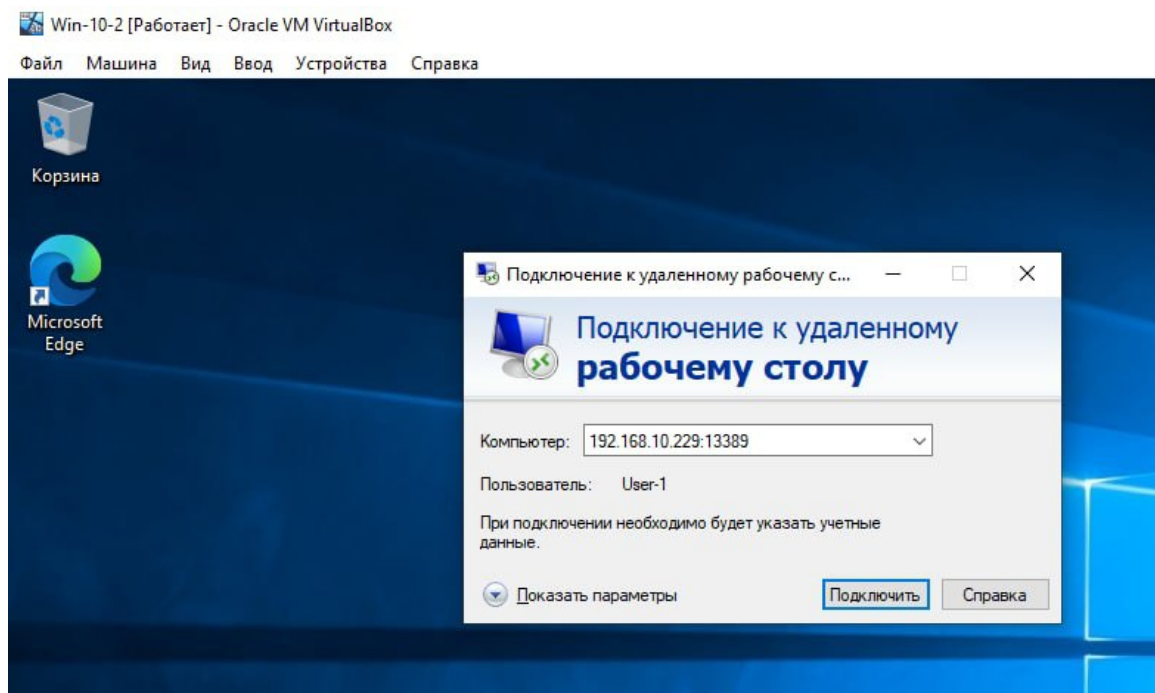


Рисунок 14 – Подключение к компьютеру №1 по другому порту

Тогда при подключении к удаленному рабочему столу в файле information.log появятся следующие записи, которые изображены на рисунке 15.

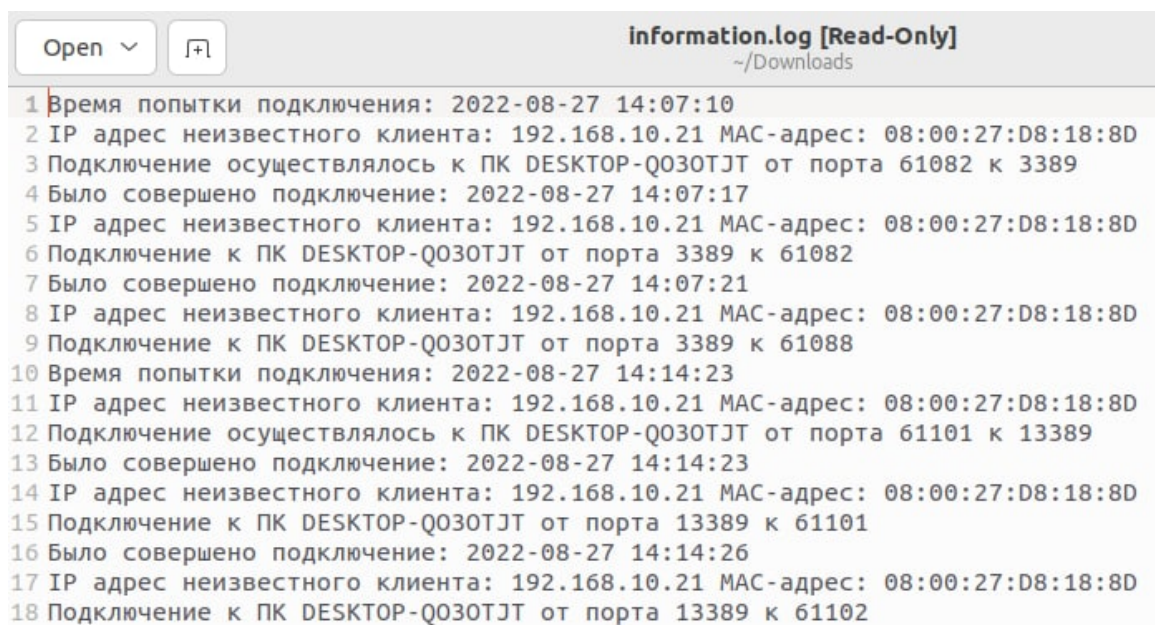
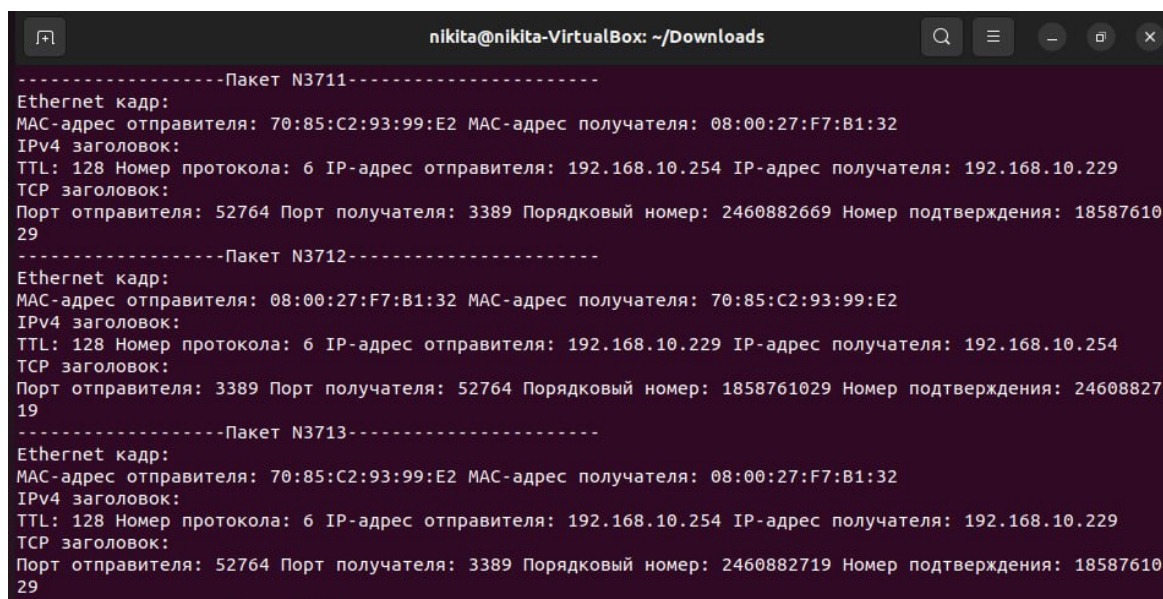


Рисунок 15 – Содержимое файла information.log после подключения по порту 13389

Получается программа установила то, что RDP-сессия произошла, опираясь на данные, которые содержатся в сообщениях протокола TLS.

3. Теперь осталось проверить подключение по протоколу RDP компьютера

№2 к компьютеру №1. При установке RDP-сессии программа отслеживает пакеты, как показано на рисунке 16.



```
nikita@nikita-VirtualBox: ~/Downloads
-----Пакет N3711-----
Ethernet кадр:
MAC-адрес отправителя: 70:85:C2:93:99:E2 MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.254 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 52764 Порт получателя: 3389 Порядковый номер: 2460882669 Номер подтверждения: 1858761029
-----Пакет N3712-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:F7:B1:32 MAC-адрес получателя: 70:85:C2:93:99:E2
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.229 IP-адрес получателя: 192.168.10.254
TCP заголовок:
Порт отправителя: 3389 Порт получателя: 52764 Порядковый номер: 1858761029 Номер подтверждения: 2460882719
-----Пакет N3713-----
Ethernet кадр:
MAC-адрес отправителя: 70:85:C2:93:99:E2 MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.254 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 52764 Порт получателя: 3389 Порядковый номер: 2460882719 Номер подтверждения: 1858761029
```

Рисунок 16 – Вид консоли при подключении к удаленному рабочему столу

Стоит отметить, что в файл `information.log` записи не были сделаны. Компьютер №2 считается верифицированным устройством, так как информация о нем содержится в файле `white-list.log`. Это сделано для того чтобы можно было точно определять несанкционированные подключения к удаленному рабочему столу.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы были разобраны методы обнаружения подключения к удаленному рабочему столу по протоколу RDP, где с помощью различных программ удалось рассмотреть принцип работы RDP-протокола. Стоит отметить, что RDP далеко не самый защищенный протокол. Хотя корпорация Microsoft регулярно выпускает обновления для своего программного обеспечения. Однако RDP-сессия становится уязвимой из-за упущений в безопасности, например из-за некорректной конфигурации сервисов или установки устаревших обновлений системы. В таком случае злоумышленник может использовать такие просчеты в своих целях. А людям, ответственным за безопасность компьютерной сети, остается только придумывать новые методы обнаружения и предотвращения несанкционированных подключений к удаленному рабочему столу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Книга Ибе О.С. «Компьютерные сети и службы удаленного доступа» / пер. с англ. - Москва, издательство: «ДМК Пресс», Яз. рус.
- 2 Удалённый рабочий стол RDP: как включить и как подключиться по RDP [Электронный ресурс] / URL: <https://hackware.ru/?p=11835> (дата обращения 03.05.2022), Яз. рус.
- 3 How to use remote desktop [Электронный ресурс] / URL: <https://support.microsoft.com/en-us/windows/how-to-use-remote-desktop-5fe128d5-8fb1-7a23-3b8a-41e636865e8c> (дата обращения 27.05.2022), Яз. англ.
- 4 Статья «Как исправить ошибку удаленного рабочего стола не удастся подключиться к удаленному компьютеру» [Электронный ресурс] / URL: <https://okdk.ru/kak-ispravit-oshibku-udalennogo-rabochego-stola-ne-udaetsya-podkljuchitsya-k-udalennomu-kompjuteru/> (дата обращения 27.05.2022), Яз. рус.
- 5 Документация Remote Utilities «RDP» [Электронный ресурс] / URL: <https://www.remoteutilities.com/support/docs/rdp/> (дата обращения 27.05.2022), Яз. англ.
- 6 Документация по стандартным библиотекам языка Python [Электронный ресурс] / URL: <https://docs.python.org/3/library/socket.html> (дата обращения 25.06.2022), Яз. англ.
- 7 Статья «Интерактивная система просмотра системных руководств (man-ов)» [Электронный ресурс] / URL: <https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=socket&category=2> (дата обращения 25.06.2022), Яз. англ.
- 8 Документация Microsoft «Протоколы» [Электронный ресурс] / URL: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpeudp2/d8bf9a56-90f3-4608-8f98-9600ed69876b (дата обращения 28.05.2022), Яз. рус.
- 9 Статья «Wireshark Tutorial: Decrypting RDP Traffic» [Электронный ресурс] / URL: <https://unit42-paloaltonetworks-com.translate.goog/wireshark->

tutorial-decrypting-rdp-traffic/?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=op,wapp (дата обращения 28.05.2022), Яз. англ.

ПРИЛОЖЕНИЕ А

Код rdp-sniffer.py

```
import socket, threading, struct
import os, datetime, sys
import keyboard

def_port = 3389 # порт по умолчанию
white_list = {} # список верифицированных устройств
black_list = [] # список неизвестных устройств
Current_object = '' # Текущий неизвестный объект
Cur_number = 1 # Счетчик всех перехваченных пакетов
Packet_cnt = 0 # Счетчик пакетов

# Получение ethernet-кадра
def get_ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)

# Получение MAC-адреса
def get_mac_addr(mac_bytes):
    mac_str = ''
    for el in mac_bytes:
        mac_str += format(el, '02x').upper() + ':'
    return mac_str[:len(mac_str) - 1]

# Получение IPv4-заголовка
def get_ipv4_data(data):
    version_header_length = data[0]
    header_length = (version_header_length & 15) * 4
    ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
    return ttl, proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]

# Получение IP-адреса формата X.X.X.X
def ipv4_dec(ip_bytes):
    ip_str = ''
    for el in ip_bytes:
        ip_str += str(el) + '.'
    return ip_str[:-1]

# Получение UDP-сегмента данных
def get_udp_segment(data):
    src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
```

```

    return src_port, dest_port, size, data[8:]

# Получение TCP-сегмента данных
def get_tcp_segment(data):
    src_port, dest_port, sequence, ack, some_block = struct.unpack('!HHLLH', data[:14])
    return src_port, dest_port, sequence, ack, data[(some_block >> 12) * 4:]

# Проверка порта по-умолчанию
def scan_port(src_ipv4, dest_ipv4, src_mac, src_port, dest_port):
    if dest_port == def_port:
        fl = False
        for key in white_list.keys():
            if key[1] == src_ipv4:
                fl = True
                break
        if not fl:
            for key in white_list.keys():
                if key[1] == dest_ipv4:
                    tup = (src_ipv4, src_mac)
                    if tup not in black_list:
                        black_list.append(tup)
                        write_to_file((src_ipv4, src_mac, key[0], src_port, dest_port), False)

# Форматирование данных для корректного представления
def format_data(data):
    if isinstance(data, bytes):
        data = ''.join(r'\x{:02x}'.format(el) for el in data)
    return data

# Проверка данных TCP-сегмента
def scan_inf(r_data, src_ipv4, dest_ipv4, src_mac, dest_mac, dest_port, src_port):
    global Current_object
    global black_list
    global Packet_cnt
    data = format_data(r_data)
    flag = False
    for key in white_list.keys():
        if key[1] == src_ipv4:
            flag = True
            break
    if not flag:
        for key, value in white_list.items():
            if value[1] in data and key[1] == dest_ipv4:
                Current_object = (key[0], key[1], value[0])

```

```

        tup = (src_ipv4, src_mac)
        if tup not in black_list:
            black_list.append(tup)
            write_to_file(( src_ipv4, src_mac, Current_object[0]
                           , src_port, dest_port ), False)
    if Current_object:
        if Current_object[2] in data:
            for key in white_list.keys():
                if key[1] == src_ipv4:
                    write_to_file(( dest_ipv4, dest_mac, Current_object[0]
                                   , src_port, dest_port ), True)
                    break
            Current_object = ''
    else:
        Packet_cnt += 1
        if Packet_cnt > 100:
            Packet_cnt = 0
            Current_object = ''

# Перехват трафика и вывод информации в консоль
def start_to_listen(interface):
    global Current_object
    global Cur_number
    os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
    server = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
    # server.bind((socket.if_indextoname(interface), 0))
    while True:
        # Получение пакетов в виде набора hex-чисел
        raw_data, _ = server.recvfrom(65565)
        dest_mac, src_mac, protocol = get_ethernet_frame(raw_data)

        # Если это интернет-протокол четвертой версии
        if protocol == 8:
            print(f'-----Пакет N{Cur_number}-----')
            Cur_number += 1
            print('Ethernet кадр: ')
            print('MAC-адрес отправителя: ' + str(src_mac), 'MAC-адрес получателя: ' + str(dest_mac))
            ttl, proto, src_ipv4, dest_ipv4, data_ipv4 = get_ipv4_data(raw_data[14:])
            print('IPv4 заголовок:')
            print( 'TTL: ' + str(ttl)
                  , 'Номер протокола: ' + str(proto)
                  , 'IP-адрес отправителя: ' + str(src_ipv4)
                  , 'IP-адрес получателя: ' + str(dest_ipv4))
            # Если это UDP-протокол
            if proto == 17:
                src_port_udp, dest_port_udp, size, data_udp = get_udp_segment(data_ipv4)
                print('UDP заголовок:')

```

```

print( 'Порт отправителя: ' + str(src_port_udp), 'Порт получателя: ' +
      str(dest_port_udp), 'Размер: ' + str(size) )

scan_port(src_ip4, dest_ip4, src_mac, src_port_udp, dest_port_udp)
# Если это TCP-протокол
if proto == 6:
    src_port_tcp, dest_port_tcp, sequence, ack, data_tcp = get_tcp_segment(data_ip4)
    print('TCP заголовок:')
    print( 'Порт отправителя: ' + str(src_port_tcp)
          , 'Порт получателя: ' + str(dest_port_tcp)
          , 'Порядковый номер: ' + str(sequence)
          , 'Номер подтверждения: ' + str(ack) )

    scan_port(src_ip4, dest_ip4, src_mac, src_port_tcp, dest_port_tcp)

    th_inf = threading.Thread(target=scan_inf, args=[ data_tcp
                                                    , src_ip4
                                                    , dest_ip4
                                                    , src_mac
                                                    , dest_mac
                                                    , dest_port_tcp
                                                    , src_port_tcp ])

    th_inf.start()
    keyboard.add_hotkey('Space', wait_key)

# Выход из программы
def wait_key():
    print('\nЗавершение работы программы...\n')
    os._exit(0)

# Запись в файл
def write_to_file(tup, bl):
    try:
        time = str(datetime.datetime.now()).split('.')[0]
        with open('information.log', 'a+') as f:
            if bl:
                f.write('Было совершено подключение: ' + time)
                f.write( '\nIP адрес неизвестного клиента: ' + str(tup[0]) +
                        ' MAC-адрес: ' + tup[1] )
                f.write( '\nПодключение к ПК ' + tup[2] + ' от порта ' + str(tup[3]) +
                        ' к ' + str(tup[4]) + '\n' )
            else:
                f.write('Время попытки подключения: ' + time)
                f.write( '\nIP адрес неизвестного клиента: ' + str(tup[0]) +
                        ' MAC-адрес: ' + tup[1] )
                f.write( '\nПодключение осуществлялось к ПК ' + tup[2] + ' от порта ' +

```

```

        str(tup[3]) + ' к ' + str(tup[4]) + '\n' )
    f.close()
except:
    pass

# Форматирование строки в hex-код
def convert_string(string):
    s = ''
    for el in bytearray(string.encode('utf-8')):
        s += '\\\\' + str(hex(el))[1:]
    return s

# Получение списка верифицированных устройств
def get_white_list():
    f = open('white-list.log', 'r')
    while True:
        line = f.readline().replace('\n', '')
        if '#' in line:
            continue
        if not line:
            break
        pos = line.find('::')
        serv_name = line[:pos]
        serv_ip = line[pos + 2:]
        white_list[(serv_name, serv_ip)] = (convert_string(serv_name), convert_string(serv_ip))

# Осуществление запуска программы
if __name__ == '__main__':
    print('\nЗапуск программы...\n')
    print('Хотите поменять RDP порт для анализа трафика? (по умолчанию 3389)')
    print('Если да, то нажмите 1, иначе - 0')
    b1 = input()
    if b1 == '1':
        print('Введите номер прослушиваемого RDP порта: ')
        def_port = int(input())

    print('Выберите сетевой интерфейс, нажав соответствующую цифру:')
    print(socket.if_nameindex())
    interface = int(input())
    try:
        get_white_list()
    except:
        print('Файл white-list.log не обнаружен')
        exit()
    else:

```

```
start_to_listen(interface)
```