Yoshiaki Ryuzaki
Sukh Atwal

Assignment 4 - Covert Channels

Contents

Introduction

The goal of this assignment was to send a message from a client to a listening server through a covert channel. The application was written in Python 2.

To use the program, the following libraries are required:

- Scapy
- Netifaces
- Json

How to Use

To launch the application, please use Python 2.7.

The Server:

1. From the server directory, execute main.py by using the command “python main.py”
2. Choose the network interface you want to listen to

Optional Changes:

- To change the ports range to listen on, modify the main.py file ○ To change the signal to use, modify the listening.py file
- To change or create new port to character pair, modify the table.json file

The Client:

1. From the client directory, execute main.py by using the command “python main.py”
2. Enter the IP address of the server
3. Enter the message you want to send across the covert channel

Optional Flag: ○ Use -f to specify a file which has the messages you want to pass

Optional Changes:

○ To change or crate new port to character pairs, modify the table.json file ○ To change the signal to use, modify the sending.py file.

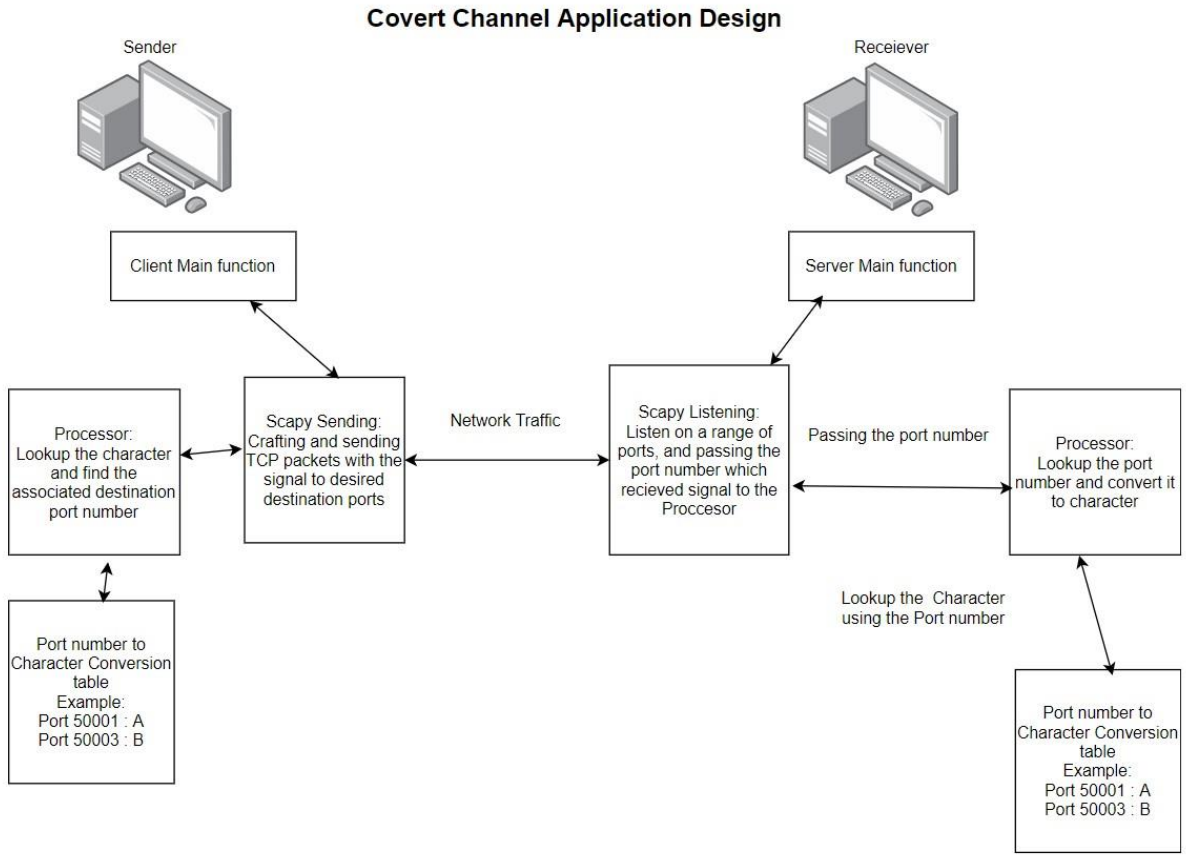
Analysis of Craig Rowland Program

The three-methods Craig mentioned in his paper are: Manipulation of IP Identification Field, TCP Sequence Number Field and the TCP Acknowledgement Field. One of the issues with Craig’s covert channel program was that there was no byte ordering functions used. He explained that this was because not including it would enable more realistic looking sequence numbers. Craig is also using an ASCII conversion, which could result in an easier time to decode the message if they know which field to look for. In our covert channel application, we are using our own conversion table. This makes it harder to know the message and we have the possibility to further encrypt it in the future.

Yoshiaki Ryuzaki
Sukh Atwal

Design

Application Overview



Our application is separated into two main parts: The client and the server.

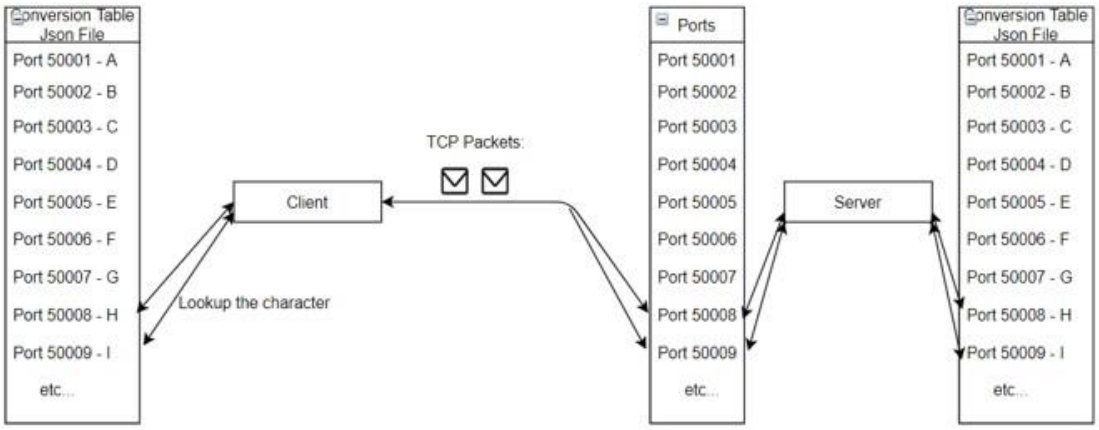
They both share the same port to character conversion table in a json format (Which pairs port numbers with characters). The server initiates the whole process starting by listening onto the range of ports which are specified in the conversion table. On the client machine, once the user passes a message to the client (Or specify a text file to use), the script will separate the message into characters and look up the associated destination port number of those characters from the conversion table.

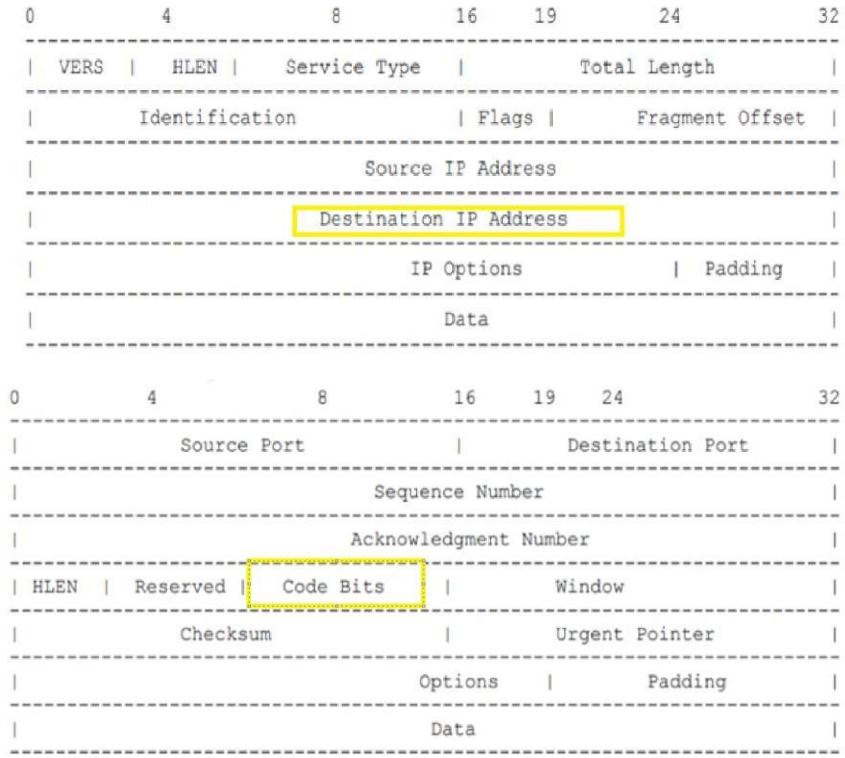
After the client's script gets all the associated destination port numbers, it'll generate TCP packets with those destination ports and a special signal (In our case, we used "40" or F for the TCP control field but other fields like HLEN can also be used). The client sends these TCP packets to the server.

When the server receives and finds these TCP packets, it records all the port numbers and search the port numbers in the conversion table to get the associated characters. Once it has all the associated characters, it combines these characters back together to get the original message that was sent. The server script will display the message on the screen also creates an output file with all the messages in the same directory.

Yoshiaki Ryuzaki
Sukh Atwal

Feature Overview





This is an example of how it works when the user sends the message, “HI”, from the client to the server.

The client-side script separates the message into character “H” and “I”, looking up the associated destination ports of these characters which would be 50008 and 50009 in this case. The client script would then send two TCP packets to port 50008 and 50009 to the server with the signal (Control flag “40” or “F” in this case).

When the server receives these packets, it sees the signal and then tries to find the associated characters using the received port numbers. At the end of it, the script puts the characters back to obtain the original message.

Yoshiaki Ryuzaki
Sukh Atwal

Testing

#	Description	Tools Used	Expectations	Actuality	Pass/Fail
1	Passing user inputted message to the server	Wireshark	User receives the message and no data will be detected by Wireshark	Server receives the message and no data can be found in the captured packets	Pass
2	Passing a text file to the server	Wireshark	User receives the message in the text file and no data will be detected by Wireshark	Server receives the message and no data can be found in the captured packets	Pass

Two machines were used for the test:

Client (Sender) with IP 192.168.1.74

Server (Receiver) with IP 192.168.1.75

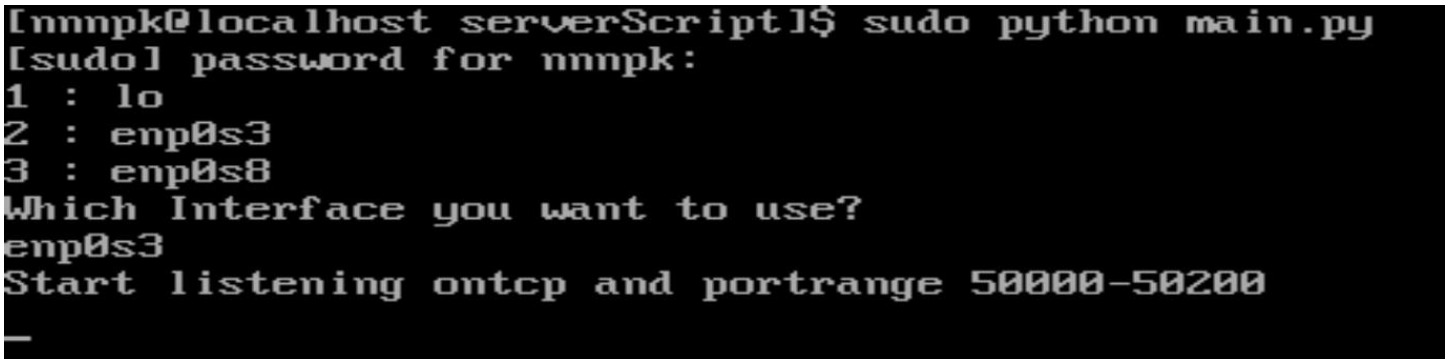
All packets are captured by using Wireshark on the 192.168.1.74 machine.

Test 1.

1. **Initial Setup**

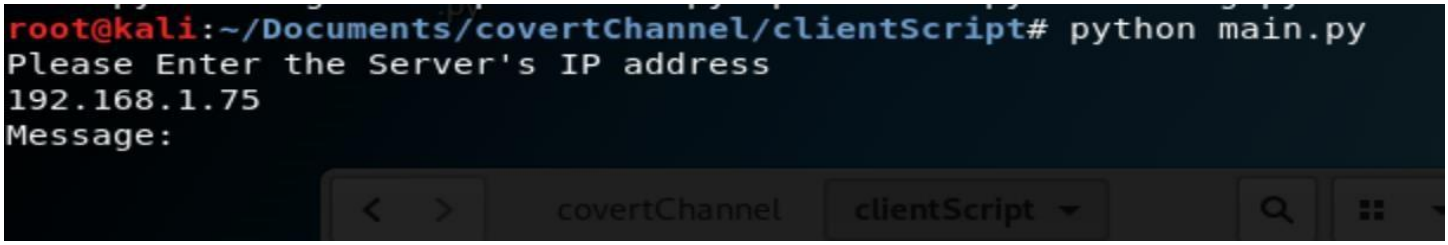
Yoshiaki Ryuzaki
Sukh Atwal
Server:

The server script will be listening on port 50000 to port 50200 of the enp0s3 network interface.



Client:

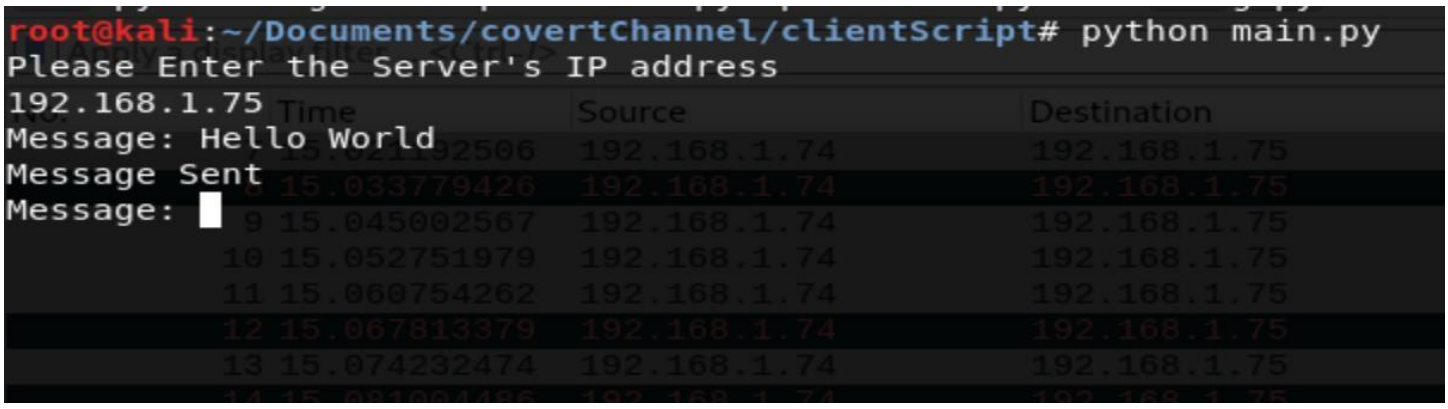
The client will send all the messages to 192.168.1.75. And the client will not use any specified text files, instead it will prompt the user to enter messages directly.



2. **Sending Message**

Client:

We can see the message “Hello World” has been sent by the server.



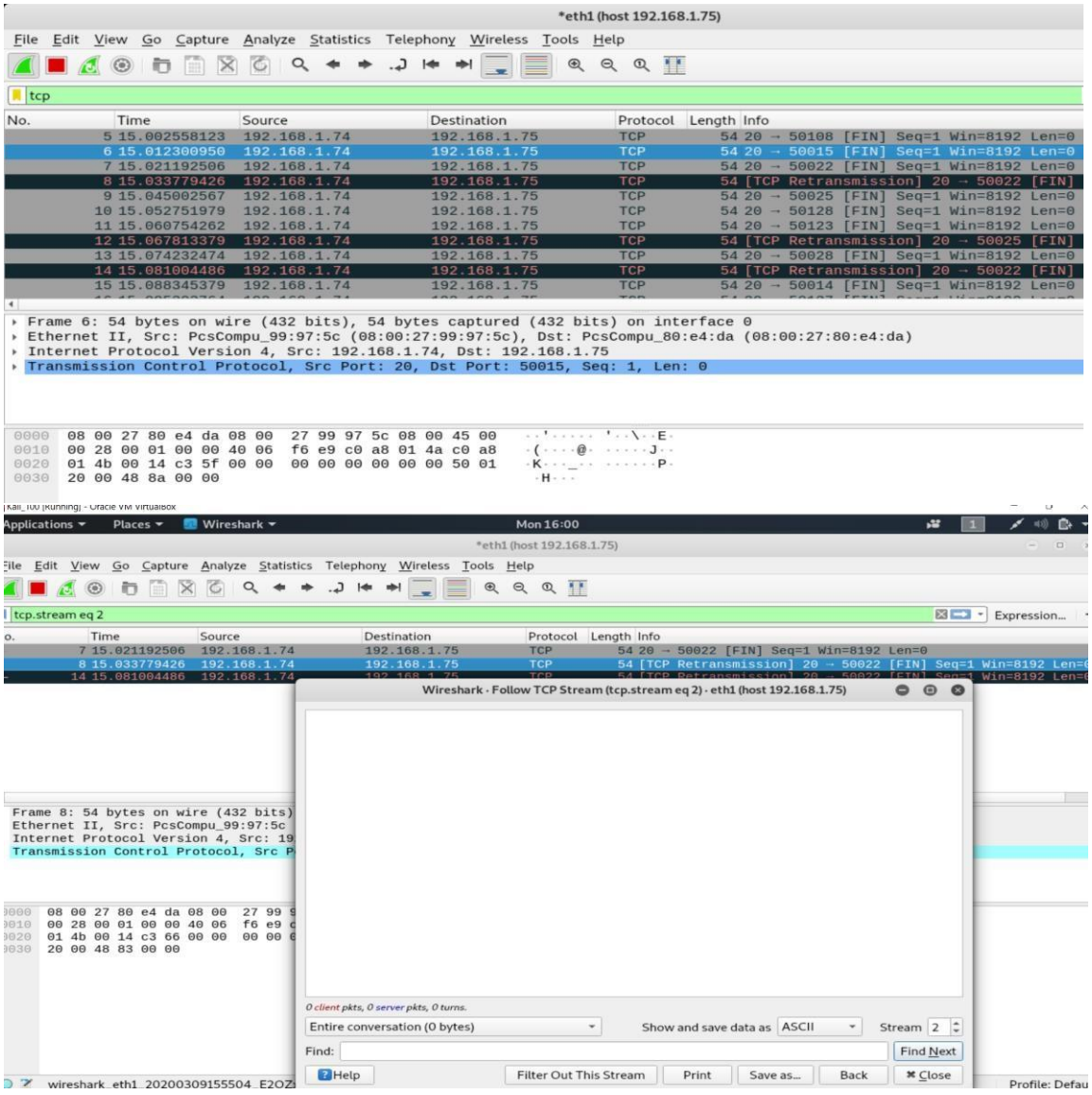
Server:

We can see the message “Hello world” has been received by the server.

```
[nmnpk@localhost serverScript]$ sudo python main.py
[sudo] password for nmnpk:
1 : lo
2 : enp0s3
3 : enp0s8
Which Interface you want to use?
enp0s3
Start listening ontcp and portrange 50000-50200
^[      Hello World
```

3. Checking captured packet

We can find the captured TCP packets, but not able to find any data from it.



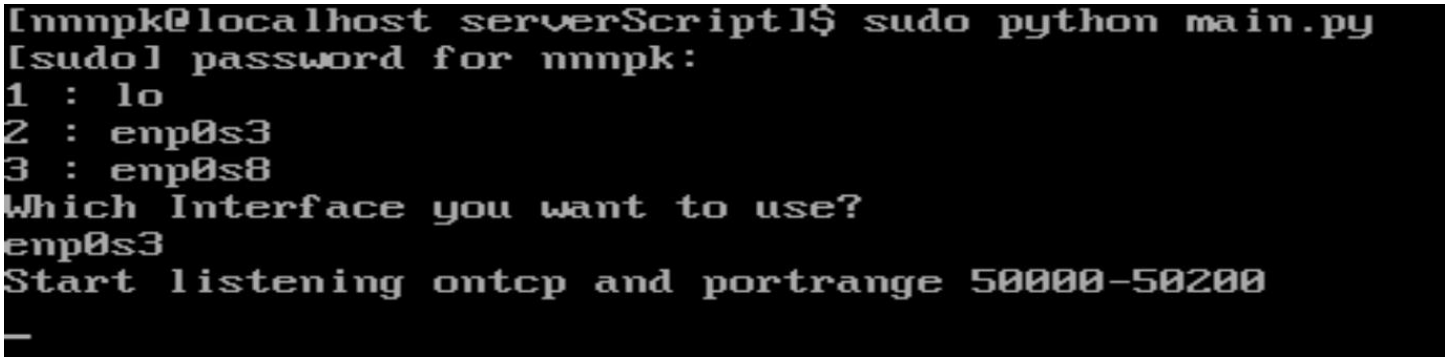
Yoshiaki Ryuzaki
Sukh Atwal

Test 2.

1. Initial Setup

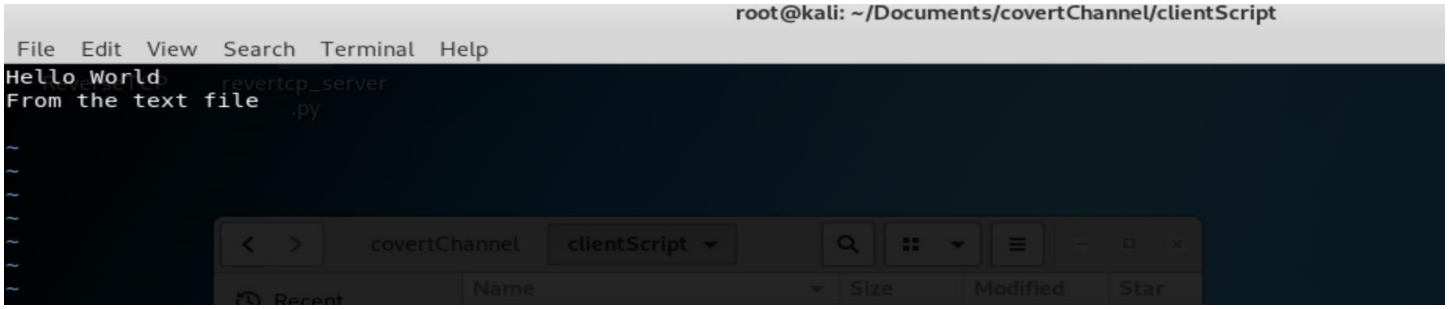
Server:

The server script will be listening on port 50000 to port 50200 of the enp0s3 network interface.

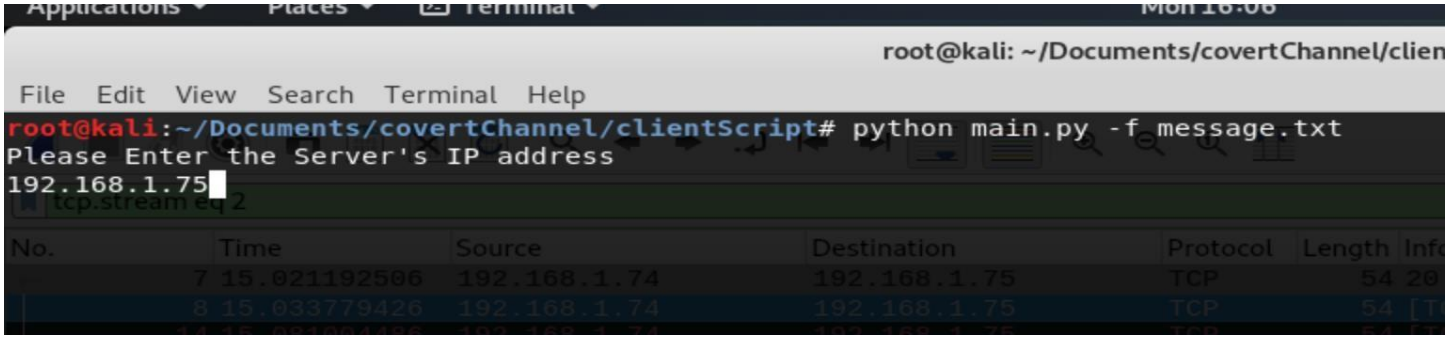


Client:

On the client, the message.txt file will be used this time:



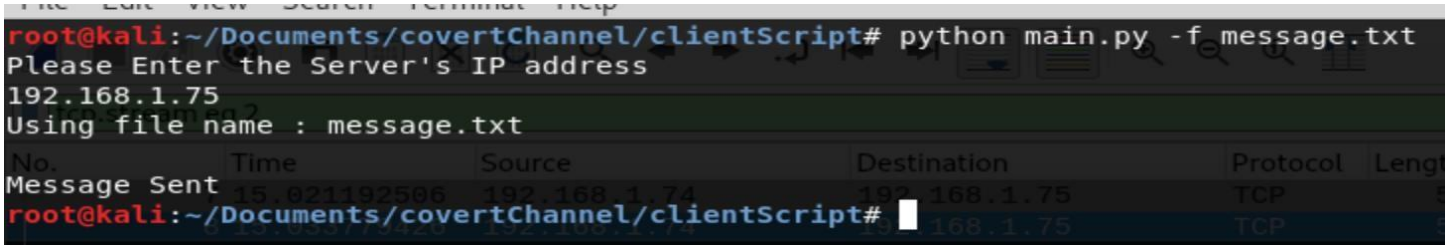
We will use “python main.py -f message.txt” command to pass the messages inside the message.txt file



2. **Sending Message**

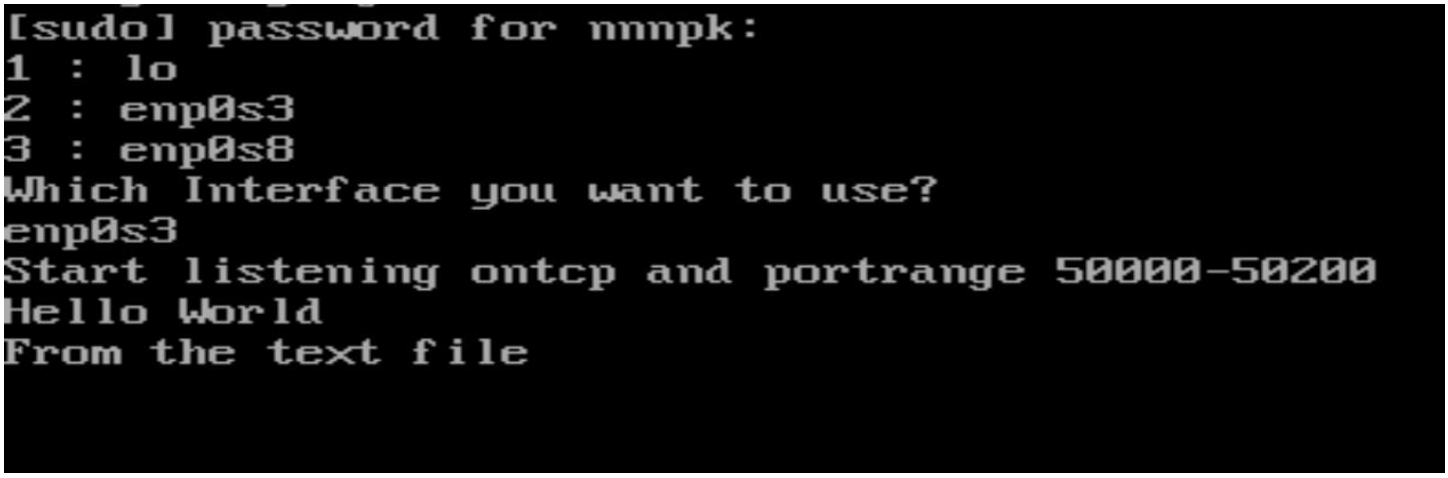
Client:

We can see the message has been sent by the client



Server:

And on the server, we can see that the server received the message



We can also check the output.txt file

3. **Checking captured packet**

We can find the captured TCP packets, but again not able to find any data from it.

Yoshiaki Ryuzaki
Sukh Atwal