

# Assignment 4 - Covert Channels

## Contents

Introduction.....	1
How to Use .....	1
The Server: .....	1
The Client: .....	2
Analysis of Craig Rowland Program.....	2
Design .....	3
Application Overview .....	3
Feature Overview .....	4
Testing .....	5
Test 1 .....	5
Test 2 .....	8

## 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.

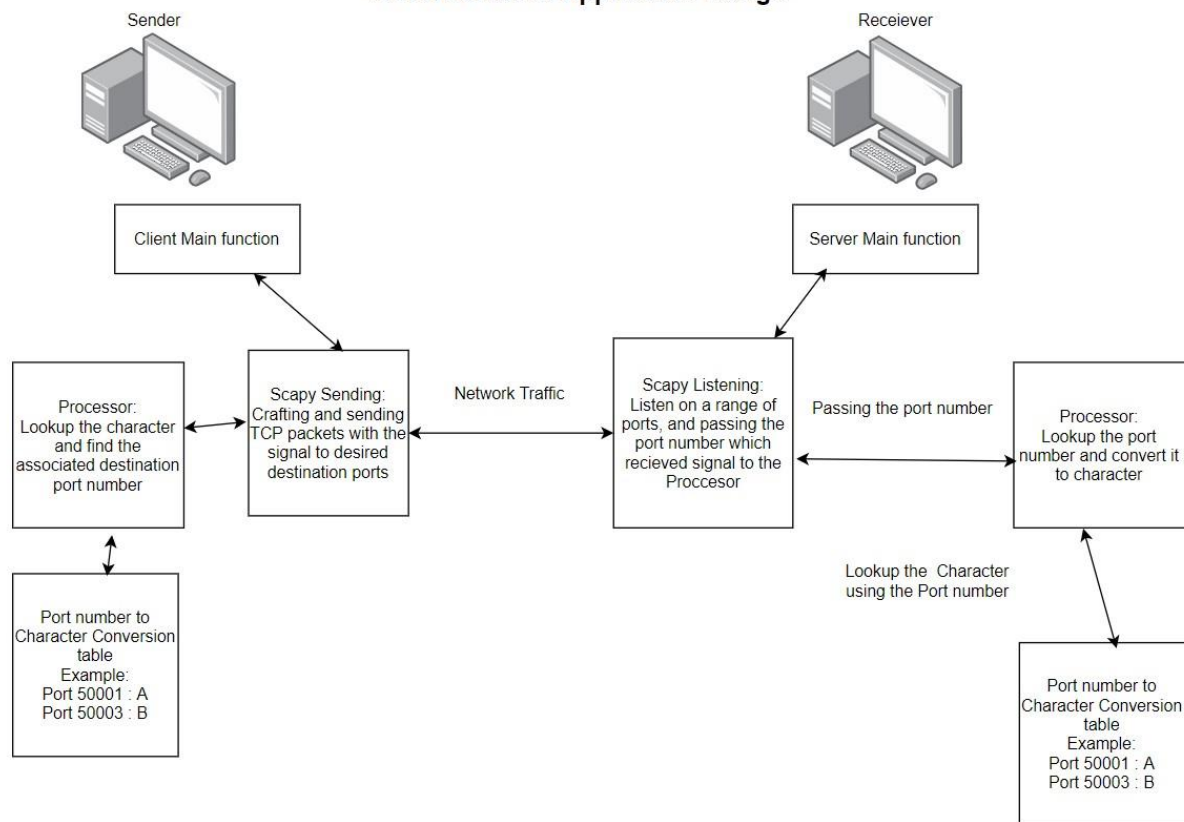
### Analysis of Craig Rowland Program

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.

## Design

### Application Overview

#### Covert Channel Application Design



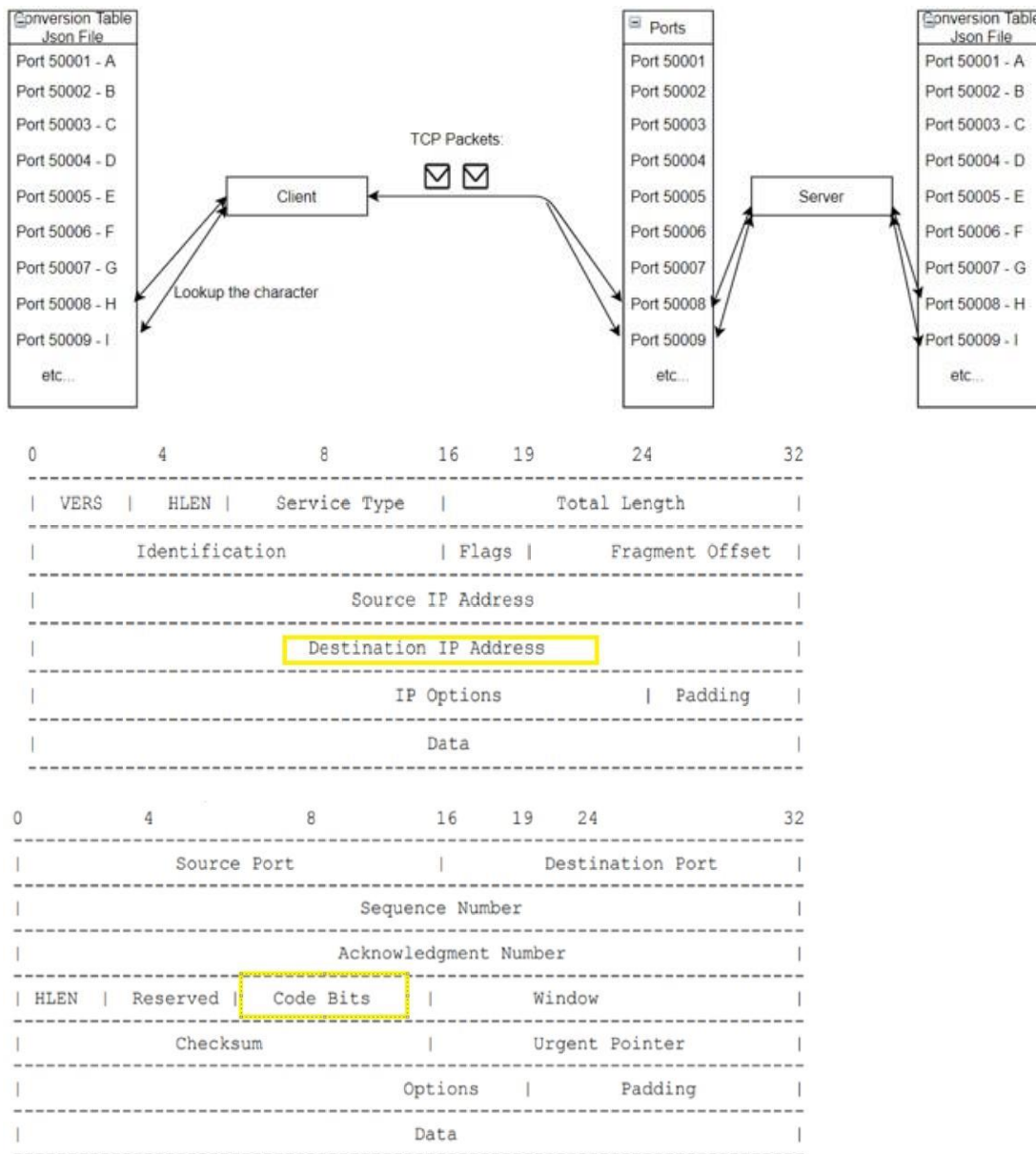
Our application is separated into two main parts: The client and the server.

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.

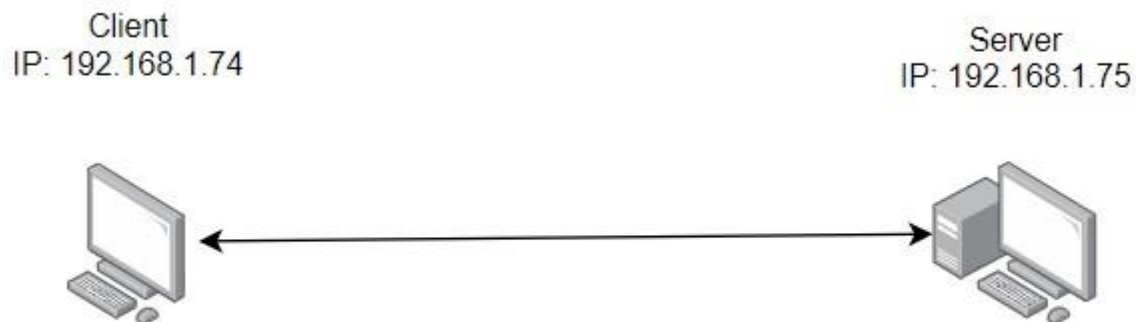
## Testing

#	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.

```
[nnnpk@localhost serverScript]$ sudo python main.py
[sudo] password for nnnpk:
1 : lo
2 : enp0s3
3 : enp0s8
Which Interface you want to use?
enp0s3
Start listening on tcp and portrange 50000-50200
_
```

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.

```
root@kali:~/Documents/covertChannel/clientScript# python main.py
Please Enter the Server's IP address
192.168.1.75
Message:
```

## 2. Sending Message

Client:

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

```
root@kali:~/Documents/covertChannel/clientScript# python main.py
Please Enter the Server's IP address
192.168.1.75
Message: Hello World
Message Sent
Message: 
```

Time	Source	Destination
15.033779426	192.168.1.74	192.168.1.75
15.045002567	192.168.1.74	192.168.1.75
15.052751979	192.168.1.74	192.168.1.75
15.060754262	192.168.1.74	192.168.1.75
15.067813379	192.168.1.74	192.168.1.75
15.074232474	192.168.1.74	192.168.1.75
15.081261490	192.168.1.74	192.168.1.75

Server:

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

Yoshiaki Ryuzaki  
Sukh Atwal

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

### 3. Checking captured packet

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

The image displays two screenshots of the Wireshark network protocol analyzer. The top screenshot shows a list of captured packets on the interface \*eth1 (host 192.168.1.75). The packets are all TCP segments from source 192.168.1.74 to destination 192.168.1.75. Notable entries include a FIN packet (seq 50015) and several retransmissions of a packet with seq 20. The bottom screenshot shows a detailed view of 'Frame 8: 54 bytes on wire (432 bits)', which is a TCP segment. The packet details pane shows Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The TCP segment has source port 19 and destination port 50015, with sequence number 1 and length 0. The packet bytes pane shows the raw data: 08 00 27 80 e4 da 08 00 27 99 97 5c 08 00 45 00 f6 e9 c0 a8 01 4a c0 a8 01 4b 00 14 c3 5f 00 00 00 00 00 00 50 01 20 00 48 8a 00 00. The 'Follow TCP Stream' window is also open, showing 'Entire conversation (0 bytes)' and 'Show and save data as ASCII'.



Yoshiaki Ryuzaki  
Sukh Atwal  
[Test 2.](#)

## 1. Initial Setup

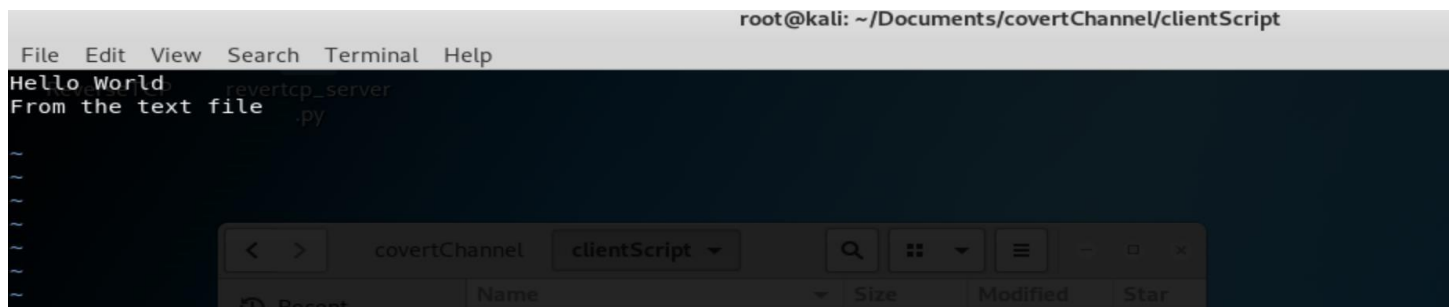
Server:

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

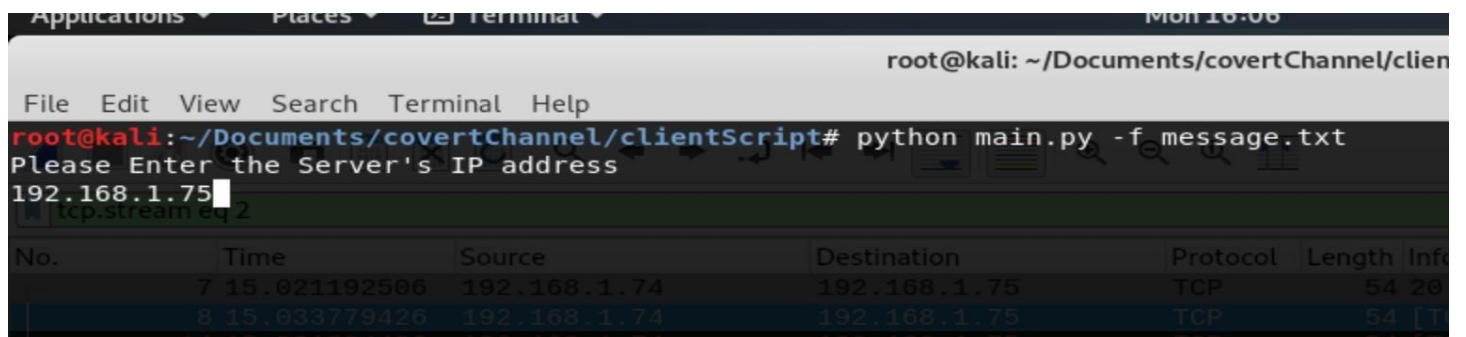
```
[nnmpk@localhost serverScript]$ sudo python main.py
[sudo] password for nnmpk:
1 : lo
2 : enp0s3
3 : enp0s8
Which Interface you want to use?
enp0s3
Start listening on tcp and portrange 50000-50200
_
```

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





Yoshiaki Ryuzaki

Sukh Atwal

## 2. Sending Message

Client:

We can see the message has been sent by the client

```
root@kali:~/Documents/covertChannel/clientScript# python main.py -f message.txt
Please Enter the Server's IP address
192.168.1.75
Using file name : message.txt
```

No.	Time	Source	Destination	Protocol	Length
Message Sent	15.021192506	192.168.1.74	192.168.1.75	TCP	8
	15.0333774426	192.168.1.74	192.168.1.75	TCP	8

```
root@kali:~/Documents/covertChannel/clientScript#
```

Server:

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

```
[sudo] password for nnnpk:
1 : lo
2 : enp0s3
3 : enp0s8
Which Interface you want to use?
enp0s3
Start listening on tcp and portrange 50000-50200
Hello World
From the text file
```

We can also check the output.txt file

```
[nnnpk@localhost serverScript]$ cat output.txt
Hello World
From the text file

[nnnpk@localhost serverScript]$
```

## 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

tcp

No.	Time	Source	Destination	Protocol	Length	Info
5	4.504025532	192.168.1.74	192.168.1.75	TCP	54	20 → 50108 [FIN] Seq=1 Win=8192 Len=0
6	4.511091291	192.168.1.74	192.168.1.75	TCP	54	20 → 50015 [FIN] Seq=1 Win=8192 Len=0
7	4.517994477	192.168.1.74	192.168.1.75	TCP	54	20 → 50022 [FIN] Seq=1 Win=8192 Len=0
8	4.528051041	192.168.1.74	192.168.1.75	TCP	54	[TCP Retransmission] 20 → 50022 [FIN] Seq=1 Win=8192 Len=0
9	4.534704554	192.168.1.74	192.168.1.75	TCP	54	20 → 50025 [FIN] Seq=1 Win=8192 Len=0
10	4.542342167	192.168.1.74	192.168.1.75	TCP	54	20 → 50128 [FIN] Seq=1 Win=8192 Len=0
11	4.550585792	192.168.1.74	192.168.1.75	TCP	54	20 → 50123 [FIN] Seq=1 Win=8192 Len=0
12	4.558664597	192.168.1.74	192.168.1.75	TCP	54	[TCP Retransmission] 20 → 50025 [FIN] Seq=1 Win=8192 Len=0
13	4.566610393	192.168.1.74	192.168.1.75	TCP	54	20 → 50028 [FIN] Seq=1 Win=8192 Len=0
14	4.574760931	192.168.1.74	192.168.1.75	TCP	54	[TCP Retransmission] 20 → 50022 [FIN] Seq=1 Win=8192 Len=0
15	4.581354642	192.168.1.74	192.168.1.75	TCP	54	20 → 50014 [FIN] Seq=1 Win=8192 Len=0

Frame 8: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0  
Ethernet II, Src: PcsCompu\_99:97:5c (08:00:27:99:97:5c), Dst: PcsCompu\_80:e4:da (08:00:27:80:e4:da)  
Internet Protocol Version 4, Src: 192.168.1.74, Dst: 192.168.1.75  
Transmission Control Protocol, Src Port: 20, Dst Port: 50022, Seq: 1, Len: 0

0000 08 00 27 80 e4 da 08 00 27 99 97 5c 08 00 45 00 ...E.  
0010 00 28 00 01 00 00 40 06 f6 e9 c0 a8 01 4a c0 a8 ...@...J..  
0020 01 4b 00 14 c3 66 00 00 00 00 00 00 50 01 ...K...f...P..  
0030 20 00 48 83 00 00 ...H...

tcp.stream eq 2

No.	Time	Source	Destination	Protocol	Length	Info
7	4.517994477	192.168.1.74	192.168.1.75	TCP	54	20 → 50022 [FIN] Seq=1 Win=8192 Len=0
8	4.528051041	192.168.1.74	192.168.1.75	TCP	54	[TCP Retransmission] 20 → 50022 [FIN] Seq=1 Win=8192 Len=0
14	4.574760931	192.168.1.74	192.168.1.75	TCP	54	[TCP Retransmission] 20 → 50022 [FIN] Seq=1 Win=8192 Len=0
33	4.713656949	192.168.1.74	192.168.1.75	TCP	54	20 → 50014 [FIN] Seq=1 Win=8192 Len=0

Wireshark · Follow TCP Stream (tcp.stream eq 2) · eth1 (host 192.168.1.75)

0 client pkts, 0 server pkts, 0 turns.

Entire conversation (0 bytes) Show and save data as ASCII Stream 2

Find: Find Next

Filter Out This Stream Print Save as Back Close