**Thilina Ratnayake**

**COMP 8505**

**A1**

Table of Contents

1. Storing In Packet Header Fields 4

1.1 Uncommon TCP Sequence. 4

1.2 Irregular Increment of IP Identification Number. 4

1.3 Frequent Change of Source Ports. 4

1.4 Maximum Message Size. 5

1.5 Lack Of Encryption 5

2. Using a Bounce Server 5

3. Solutions For A Better Covert Channel. 5

3.1 Firewall evasion 5

3.2 No RSTs 6

3.3 Unsuspicious means 6

3.2 Covering The Trail 6

3.4 Encrypting The Messages 6

Assumptions: 7

Key-Features: 7

Decisions Made: 7

Design: 8

Languages & Libraries Used: 9

Summary:

Covert Channels are a means of conveying information between two parties in a manner that does not arouse normal avenues of detection and surveillance. Craig Rowland’s code is a good proof of concept but has some inherent weaknesses, which makes it inefficient and detectable. Those weaknesses are criticized in the paper and in response, a solution has been designed and implement. The solution involves sending only one TCP packet and the client pulling for messages via ICMP Ping messages and hiding the initial IP in IP.

Critique of Craig Rowland’s Covert Channels

In Craig Rowland’s implementation of covert channels found in covert\_tcp.c, there are a number of weaknesses that undermine that ability of the channel to be truly covert. While Rowland deals with three different implementations of his covert channel, I will first critique the first (stuffing packets in header fields) in detail, as my application will be using a similar strategy, and briefly critique the third “bounce-server” implementation. Lastly, I will discuss a solution for many of these issues, which will ultimately become the framework for my covert channel.

# 1. Storing In Packet Header Fields

Rowlands first method deals with stuffing each character of a given message in ASCII encoding within the IP identification field. This method has five weaknesses that make this channel inefficient and detectable.

## 1.1 Uncommon TCP Sequence.

The first weakness is the uncommon TCP sequence. Every SYN request from the client is replied with a RST due to the use of raw sockets. Specifically, because the server is listening on an unbound socket which prevents a successful 3-way-handshake from occurring. For any analyst observing traffic, even merely filtering the traffic between these two IP’s; this activity would show-up as a red flag as a SYN-RST sequence does not fit with usual network traffic patterns.

## 1.2 Irregular Increment of IP Identification Number.

The second weakness lies within the IP layer identification field. As outlined in RFC 4413:

“[T]he IP ID value will increment by one for each packet in the stream, except at wrap around[…]”.   
While it should be incrementing with every transmission; Due to the identification number depending on the ASCII value of the message being sent, the increment and decrement in IP identification numbers produced by Rowland’s code are unusual when compared against the RFC.

## 1.3 Frequent Change of Source Ports.

Thirdly, the port numbers from the client are changing with every request and in a short interval. Again, this is due to the way in which Rowland “encodes” the data within the packet, which leaves the port dependant on the ASCII value of the message. This behaviour is very uncharacteristic of a client server connection, which usually has a client connecting from the same port, or ports within a similar range.

## 1.4 Maximum Message Size.

A weakness in both application and detection is the fact that the maximum message size for every transmission is one byte per packet. This is highly inefficient and also increases the chance of detection, as a trail of many packets will exist between both parties.

## 1.5 Lack Of Encryption

The messages consist of the ASCII value of each character in the message which is essentially the same as sending a message plaintext. A keen analyst would be able to observe all the packets and easily reconstruct them to find out what the original message is.

# 2. Using a Bounce Server

Rowland’s third implementation of his channel involves using an intermediary server to relay his messages to the server. Some inherent weaknesses of this are the following.

At the target:

* Many SYN/ACKS are coming in with no initial SYNs. This would instantly get dropped by a stateful firewall.
* The source and destination ports are the same on the packet.

At the bounce server:

* Every SYN/ACK issued by the bounce server is met with a reset.

# 3. Solutions For A Better Covert Channel.

## 3.1 Firewall evasion

While many stateful firewalls will be able to stop many covert channels, the problem is that Rowland’s code will be curtailed even by a stateless firewall with some merely decent firewall rules. Specifically those that watch for multiple incoming SYNs, which are always replied to with RSTs. Ideally, one would try to avoid sending a SYN altogether using TCP, however it would be best to limit the amount of TCP packets sent to a host (as they prompt for watching for 3-way handshakes). One workaround is using a single TCP packet as a messenger; such by sending only one SYN message (which can be considered usual as if one was simply trying to access a web server on a network). Then one could hide an IP in that message and then finally; have the compromised system use the IP to send and receive data to.

## 3.2 No RSTs

The first weakness of Rowland’s channel was every SYN being met with a RST due to going to an unbound socket. While this is setting is not ideal for a connection oriented protocol such as TCP, it is perfectly normal for a protocol such as ICMP. Therefore, one solution is to implement a covert channel that sends on ICMP.

## 3.3 Unsuspicious means

On a locked down system, any incoming connection could be considered suspicious. However, many overlook the outgoing connections, especially with something like pings. Therefore an unsuspicious means of communication that could be used is the ICMP ping mechanism. A ping could be sent out from the compromised host through the firewall and incoming messages can be hidden in the response.

## 3.2 Covering The Trail

Rowland’s code makes no attempt to hide the originating address of the packet. The receiver sees exactly where the packet comes from and every ensuing packet connects to the attacker directly, which rises suspicious even more, leaving the attacker vulnerable. One way to hide an IP address is to never have it appear over the wire. Specifically one could break the IP down into bits and hide it into multiple header fields to be reconstructed at the compromised host. After the IP has been reconstructed at the compromised host, he can choose to ping out to that IP which seems normal in terms of examining network traffic.

## 3.4 Encrypting The Messages

Whereas Rowland only stores the ASCII value of a character in the packet, it would be wise to employ some sort of encryption to ensure that a message cannot be reconstructed by what’s on the wire. Something as simple as a pre-determined shift of values can be enough to encrypt the message.

In conclusion, Craig Rowland’s code is a good proof-of-concept however it has many problems dealing with both evasion and efficiency. A more detailed solution can be found in the next section of this document, which explains the design of my covert channel.

Design Work

The covert channels that I have attempted to implement will allow for bi-directional communication while evading most non-stateful firewalls and employing features such as encryption and reliability and the application layer.

For my implementation of the covert channels, a few assumptions and terms must be established:

# Assumptions:

1. There are two parties involved. A client “Blackhat” who will be sending messages via covert channels to a client “compromised Host” running a backdoor.
2. It is assumed that the client is behind a non-stateful firewall allowing connections on port 80 & outgoing ICMP pings and ping replies.
3. It is assumed that the client will be running the backdoor with root permissions.

# Key-Features:

* The Blackhat can send a message to a Compromised host.
* The Compromised host will be able to send messages to the Blackhat.

# Decisions Made:

* One of the primary flaws of Rowland’s code was the SYN/RSTs. Since the primary mechanism of carrying data for Rowland was in the TCP packets, this makes it very suspicious. Therefore, the decision was made to minimize the use of TCP packets. A TCP packet will only be used once to send a control packet.
* A second key flaw was the RST, which is due to the messages leading to unbound sockets. To circumvent this, the stateless protocol ICMP has been chosen.
* Any incoming data is inherently suspicious, whether it be a TCP SYN/PSH or a UDP packet. However, outgoing data is usually not as equally scrutinized. Therefore the existing outgoing PING/REPLY mechanism using ICMP is a perfect means to send data.

# Design:

The covert channel will work in two phases:

1. Control Packet & Handshake:
   1. The Blackhat will imitate an initial connection to a webserver on the clients machine by sending a TCP SYN to port 80. The source address of this packet will be spoofed to cover the trail to the Blackhat. This packet will contain the IP address of the Blackhat’s system stored into the IP identification and TCP source ports. The Blackhat will then begin listening for connections from the compromised host.
   2. The compromised host, after a variable amount of time, will then call home to the Blackhat with a PING message. This PING message will complete the handshake that tells the Blackhat that a connection has been established. This handshake also asks whether the Blackhat has anything to send.
2. Covert Messages Into The Compromised Host
   1. A Blackhat will prompt the CompromisedHost that it’s time to send by sending him/her a new control packet/handshake. Once this packet has been sent, the initial PING from the compromised host will ask if Blackhat wishes to send data.
   2. When the Blackhat replies yes, the compromised host will query as to how many packets to expect from the Blackhat (and how many times to PING the Blackhat to facilitate the length of the transfer).
   3. The Blackhat will packetize the message into the length of packets as specified.
   4. The CompromisedHost will PING the Blackhat the necessary amount of times to transfer the message.
3. Covert Messages From The Compromised Host
   1. After a control packet/ handshake, the Blackhat is always listening for PINGS on his/her system.
   2. Once a CompromisedHost wants to send, they will first send a control message to the Blackhat explaining that they are about to send a user determined amount of messages.

Features:

* **IP masking**: The IP address of the Blackhat is never sent over the wire. It is hidden in the IP Identification and TCP Source Port fields to be reconstructed at the client.
* **Bi-Directional:** Blackhat and CompromisedHost can both send messages to each other.

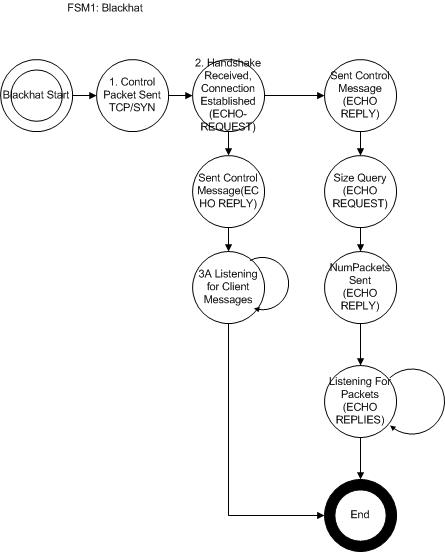
Features For Future Implementation:

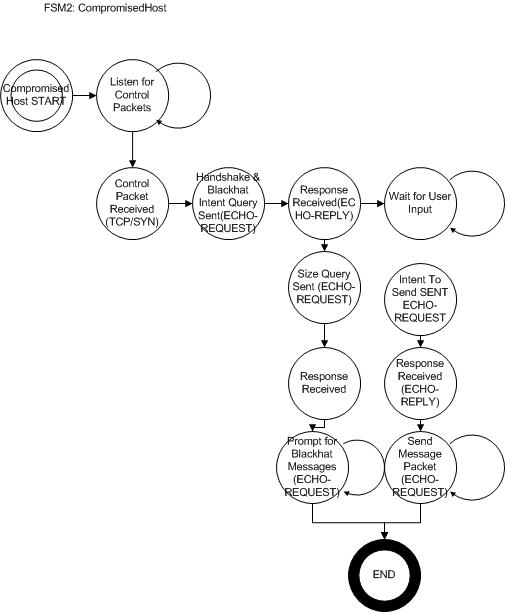
* **Reliability**: Every transmission should be preceded with a hail that explains how many packets are to be sent. Every packet should then carry an identification number and the other party should prompt for resending if any packets are missing.
* **Encryption:** All messages should be encrypted using some sort of algorithm. Python’s crypto library has support for DES. The initial key can also be hidden into some IP/TCP field in the initial control packet.

# Languages & Libraries Used:

* Language: Python v2.7
* Packet crafting & sniffing library: Scapy v2.1.1

Diagrams





Pseudo Code:

**BlackHat:**

1. Send Control Packet
   1. Craft the control packet
      1. Grab the current (Blackhat IP), break it into bits.
      2. Store 16 bits inside TCP ID, 16 bits inside TCP Sport
   2. Send the control packet
2. Go into Listening Mode
   1. Listen for Handshake messages from Client
   2. On Handshake:
      1. IF you don’t want to send, send message to client that you’re ready to receive
      2. IF you want to send
         1. Packetize message by dividing by the Maximum Transmission Unit
         2. Send control message that you want to send (code = 8 ECHO REQUEST)
         3. Answer clients query of how many packets will be required (code = ECHO REPLY)
         4. Wait for clients pings and insert each packet to the response

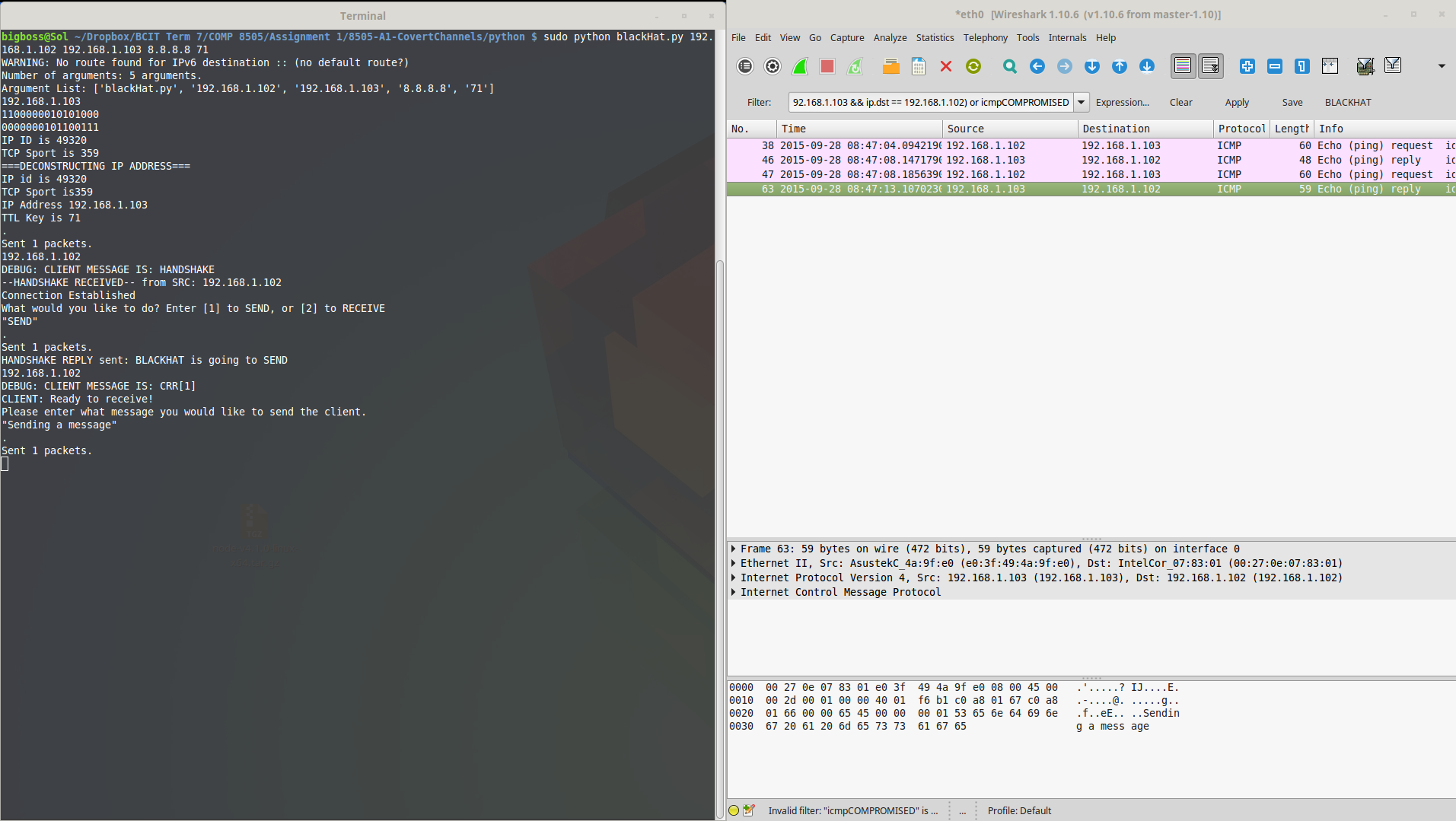
**CompromisedHost:**

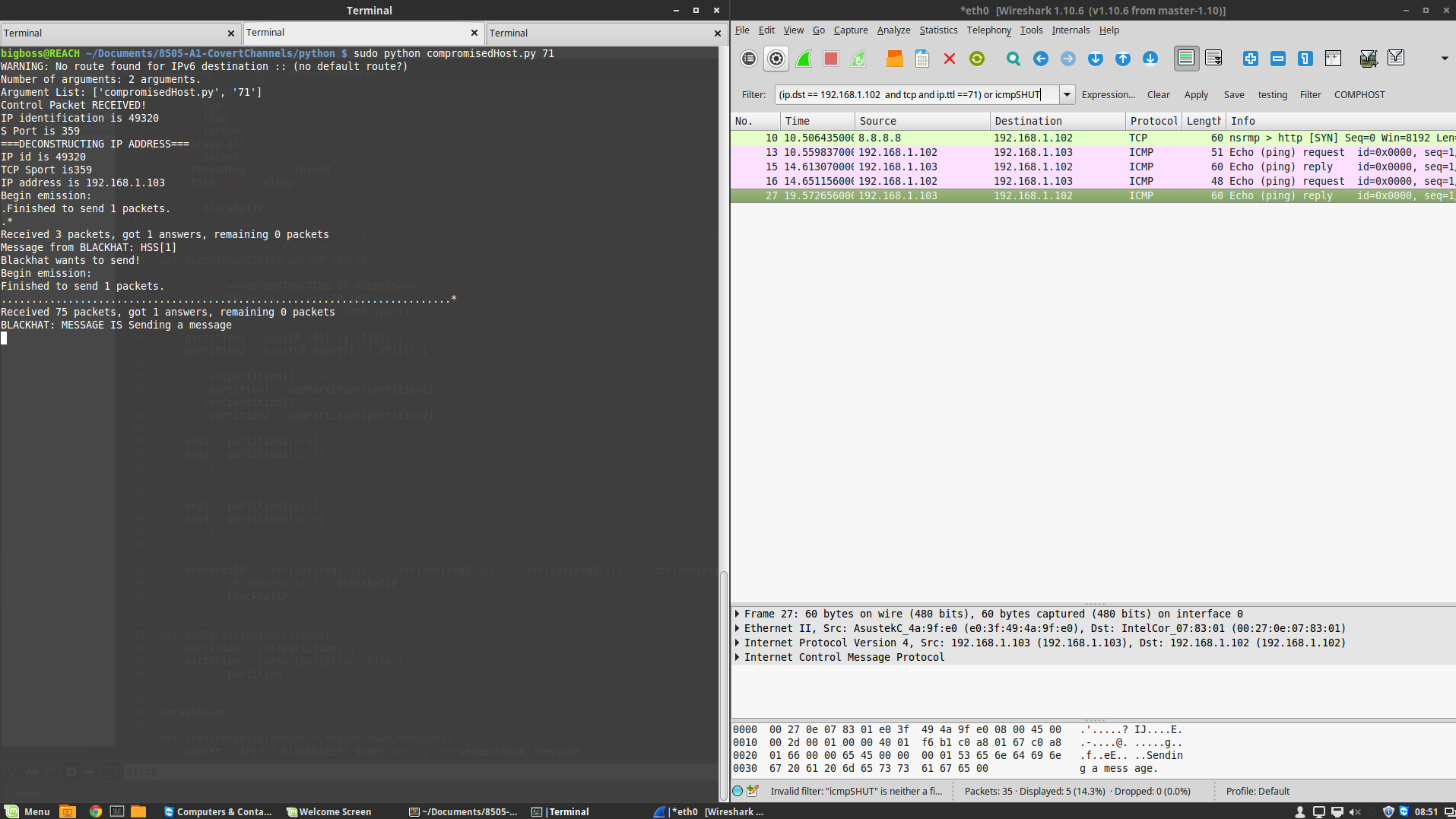
1. Listen for control packets
   1. Decipher the IP address of the control packet
      1. Combined IP ID field + TCP Sport field
   2. Send handshake complete message to Blackhat, ask if client wants to send
   3. IF Yes for sending:
      1. Query Blackhat on number of messages
      2. Send a PING for number of messages
      3. Combine all the responses to construct data
   4. IF No: Listen for control packets and input from user to send.
2. Go into Listening Mode

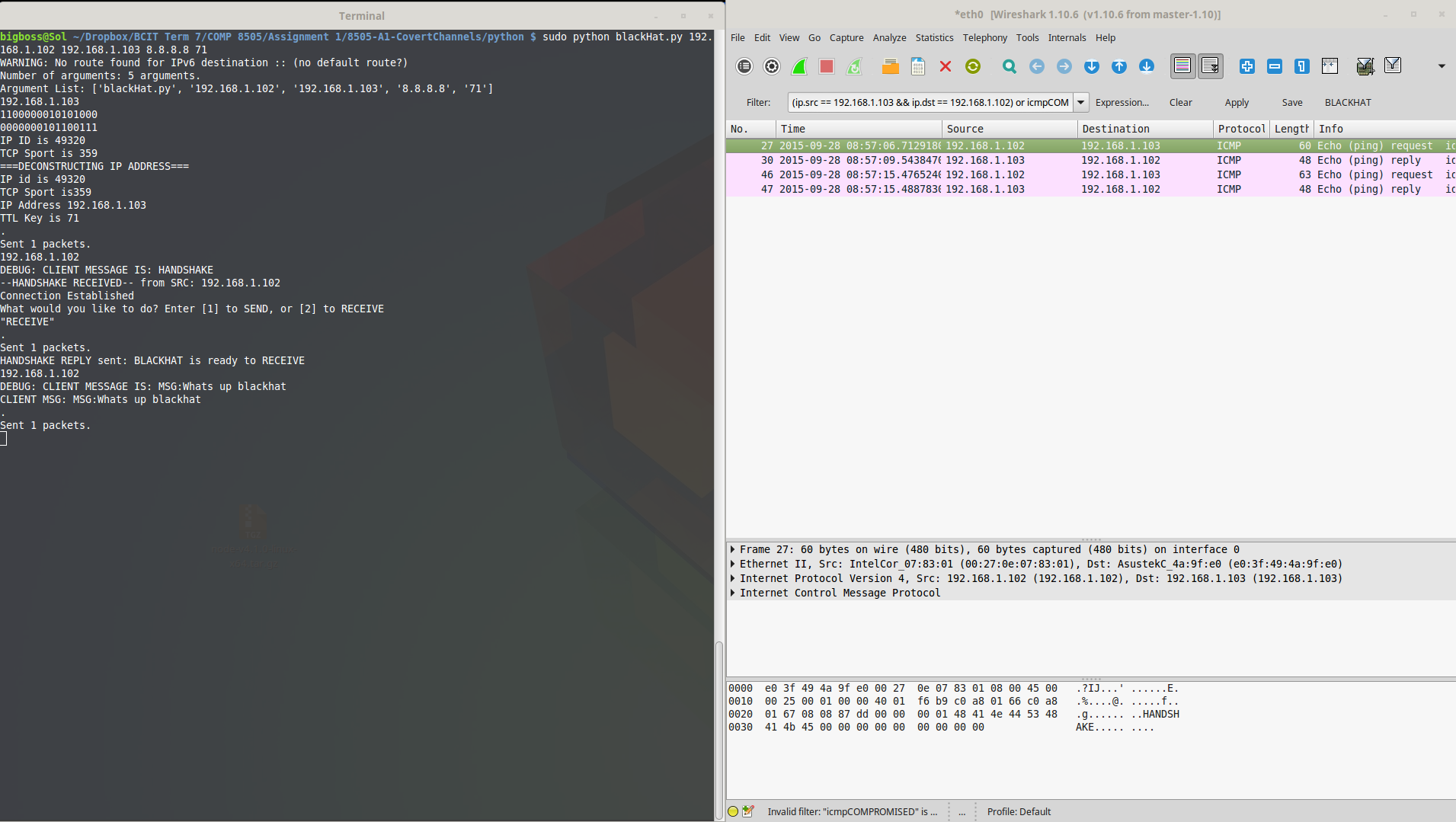
Testing & Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Name** | **Resources** | **Result** | **Figures** |
| 1. | Blackhat able to send a message to Client. | Blackhat.py  CompromisedHost.py | PASS | 1.1, 1.2 |
| 2. | Client able to send messages to Blackhat | Blackhat.py  CompromisedHost.py | PASS | 2.1, 2.2 |
| 3. | Blackhat able to send packets through Firewall that only allows packets through port 80 & outgoing ICMP | Blackhat.py  CompromisedHost.py | UNTESTED |  |
| 4. | Messages don’t raise any SNORT alerts | Blackhat.py  CompromisedHost.py | UNTESTED |  |

**Figures:**

Fig 1.1 Blackhat: Blackhat sending to CompromisedHost

Figure1.2 Client: Blackhat sending to Client

Fig 2.1. Blackhat: Client sending to Blackhat

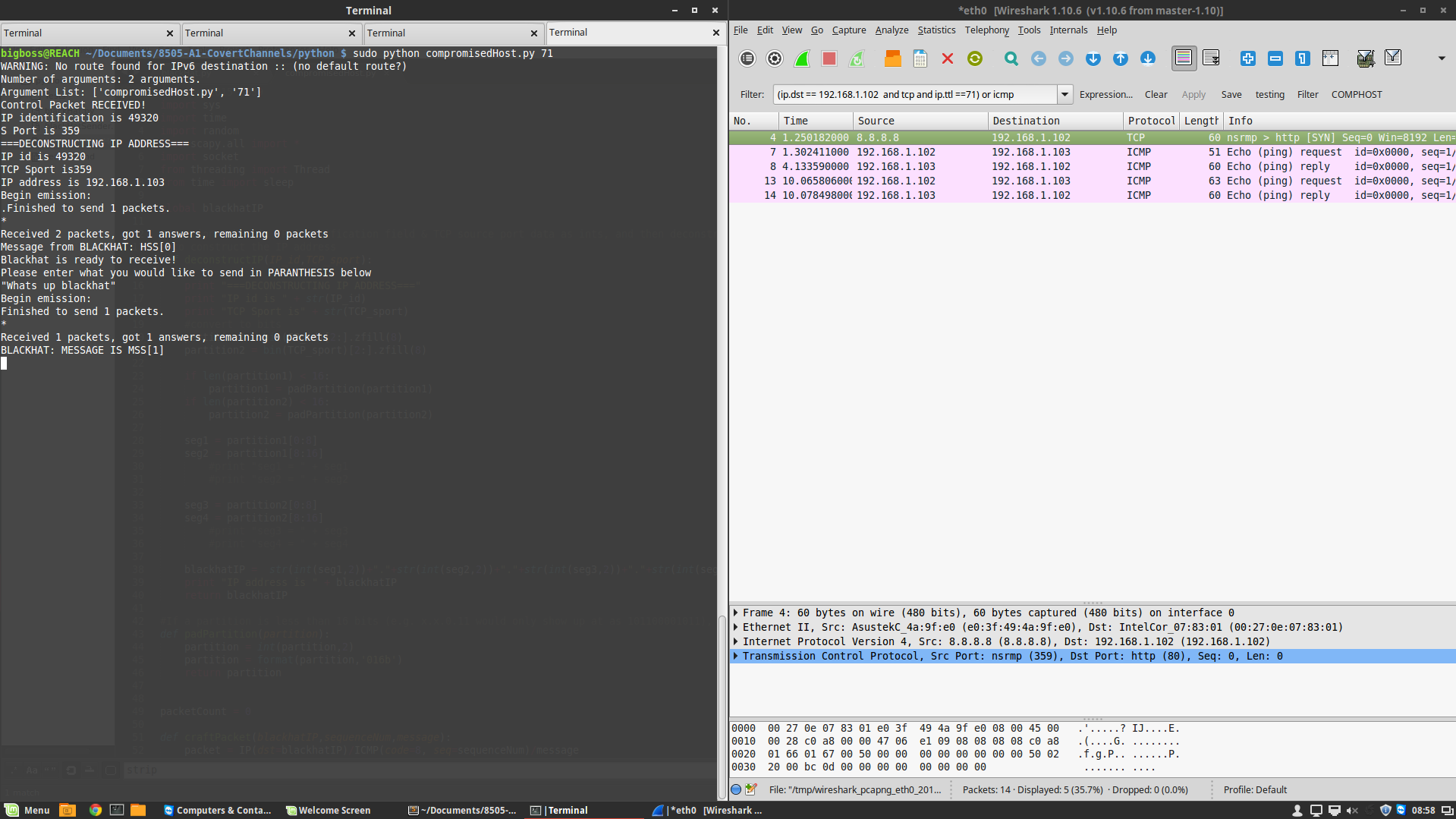


Fig 2.2 Client: Client Sending To Blackhat