Thilina Ratnayake
COMP 8505
A1

Thilina Ratnayake C8505 08 September 2015
A00802338 A1

Table of Contents

Thilina Ratnayake C8505 08 September 2015
A00802338 A1

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.

Thilina Ratnayake C8505 08 September 2015
A00802338 A1

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:
    a. 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.

    b. 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
    a. 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.
    b. 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).
    c. The Blackhat will packetize the message into the length of packets as specified.
    d. The CompromisedHost will PING the Blackhat the necessary amount of times to transfer the message.

3. Covert Messages From The Compromised Host
    a. After a control packet/ handshake, the Blackhat is always listening for PINGS on his/her system.
    b. 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:

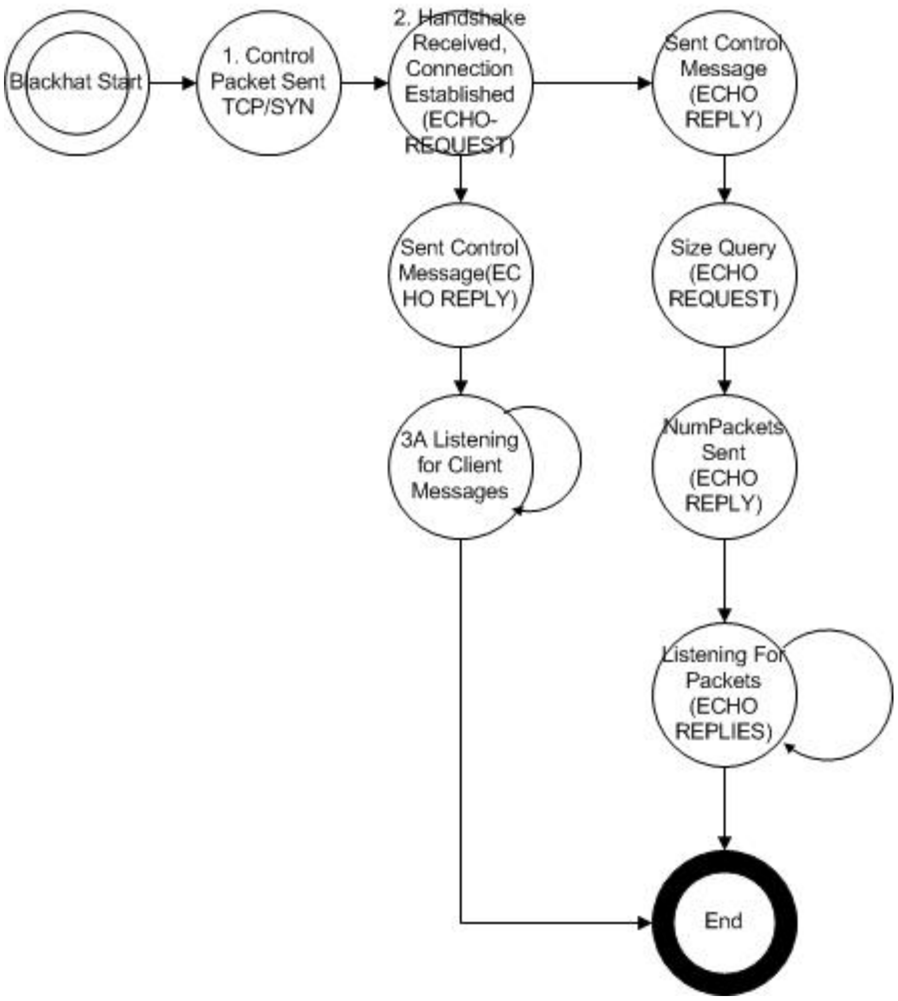
- **Reliability**: Every transmission should be preceded with a hail that explains how many packets are to be sent. Every packet should then carry an identification number and the other party should prompt for resending if any packets are missing.
- **Encryption:** All messages should be encrypted using 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:

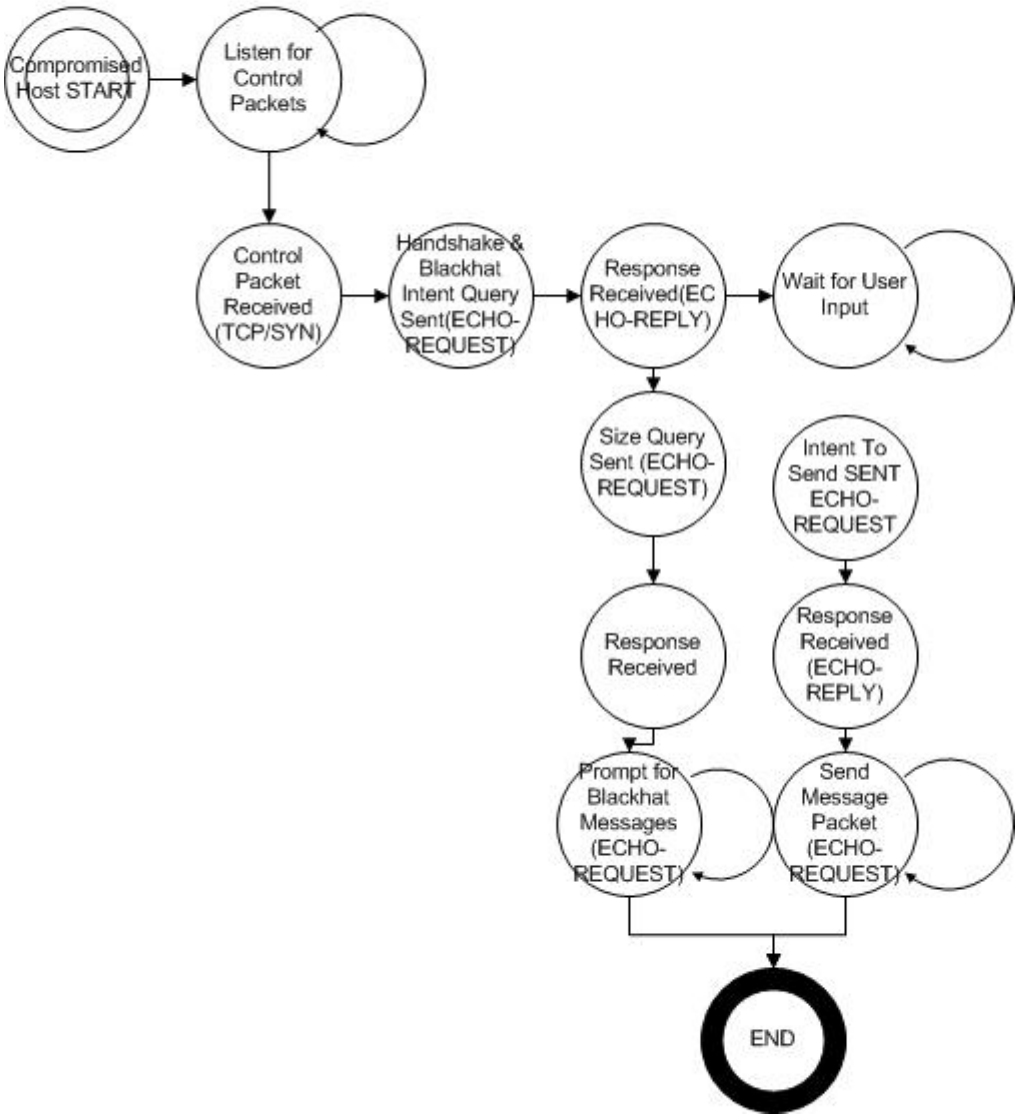
- Language: Python v2.7
- Packet crafting & sniffing library: Scapy v2.1.1

Diagrams

FSM1: Blackhat


Blackhat Start
1. Control Packet Sent TCP/SYN
2. Handshake Received, Connection Established (ECHO-REQUEST)
Sent Control Message (ECHO REPLY)
Sent Control Message(ECHO REPLY)
Size Query (ECHO REQUEST)
3A Listening for Client Messages
NumPackets Sent (ECHO REPLY)
Listening For Packets (ECHO REPLIES)
End

FSM2: CompromisedHost


Compromised Host START
Listen for Control Packets
Control Packet Received (TCP/SYN)
Handshake & Blackhat Intent Query Sent(ECHO-REQUEST)
Response Received(ECHO-REPLY)
Wait for User Input
Size Query Sent (ECHO-REQUEST)
Intent To Send SENT ECHO-REQUEST
Response Received
Response Received (ECHO-REPLY)
Prompt for Blackhat Messages (ECHO-REQUEST)
Send Message Packet (ECHO-REQUEST)
END

Pseudo Code:

BlackHat:

```
1.  Send Control Packet
      a. Craft the control packet
           i. Grab the current (Blackhat IP),  break it into bits.
          ii. Store 16 bits inside TCP ID, 16 bits inside TCP Sport
      b. Send the control packet
2.  Go into Listening Mode
      a. Listen for Handshake messages from Client
      b. On Handshake:
           i. IF you don't want to send, send message to client that you're ready
              to receive
          ii. 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
```

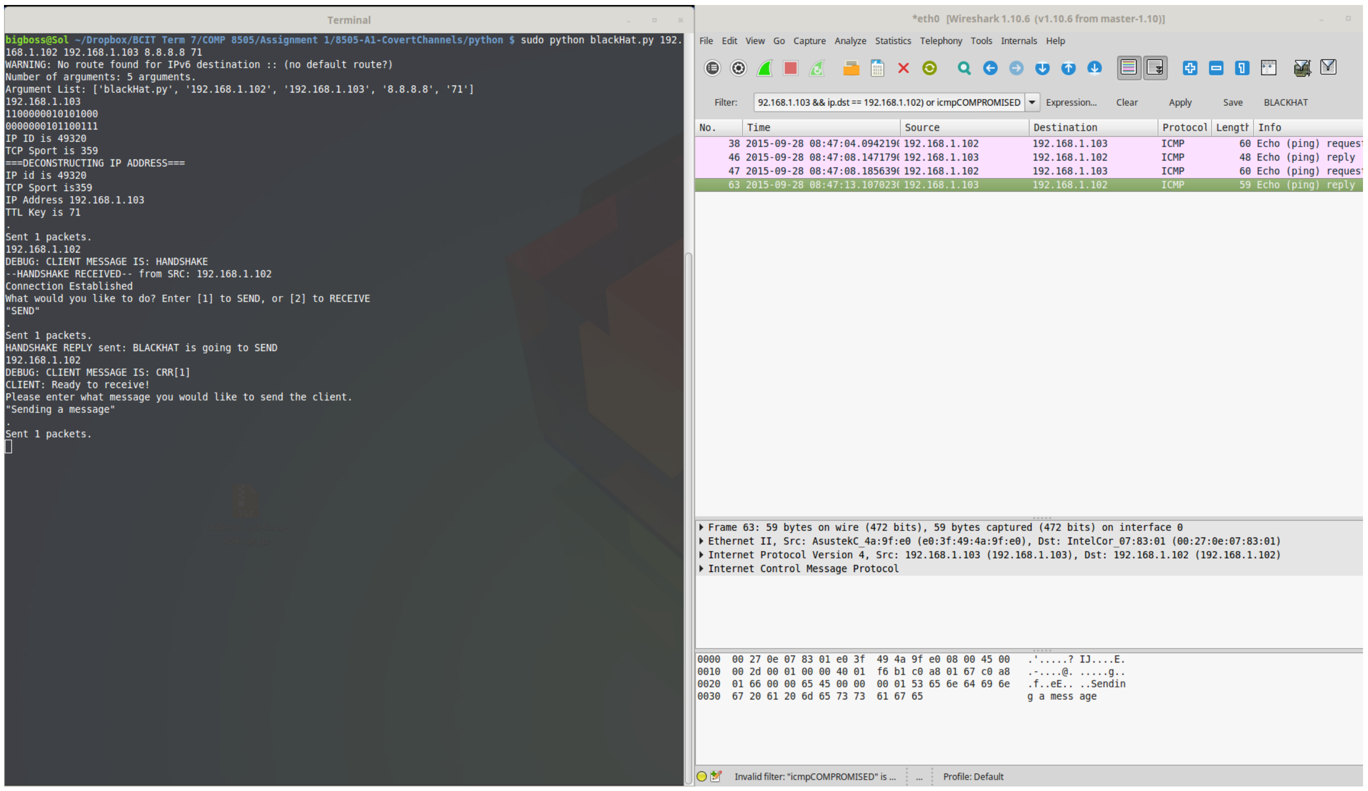
CompromisedHost:

```
1.  Listen for control packets
      a. Decipher the IP address of the control packet
           i. Combined IP ID field + TCP Sport field
      b. Send handshake complete message to Blackhat, ask if client wants to send
      c. IF Yes for sending:
           i. Query Blackhat on number of messages
          ii. Send a PING for number of messages
         iii. Combine all the responses to construct data
      d. 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:


Terminal
*eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save BLACKHAT

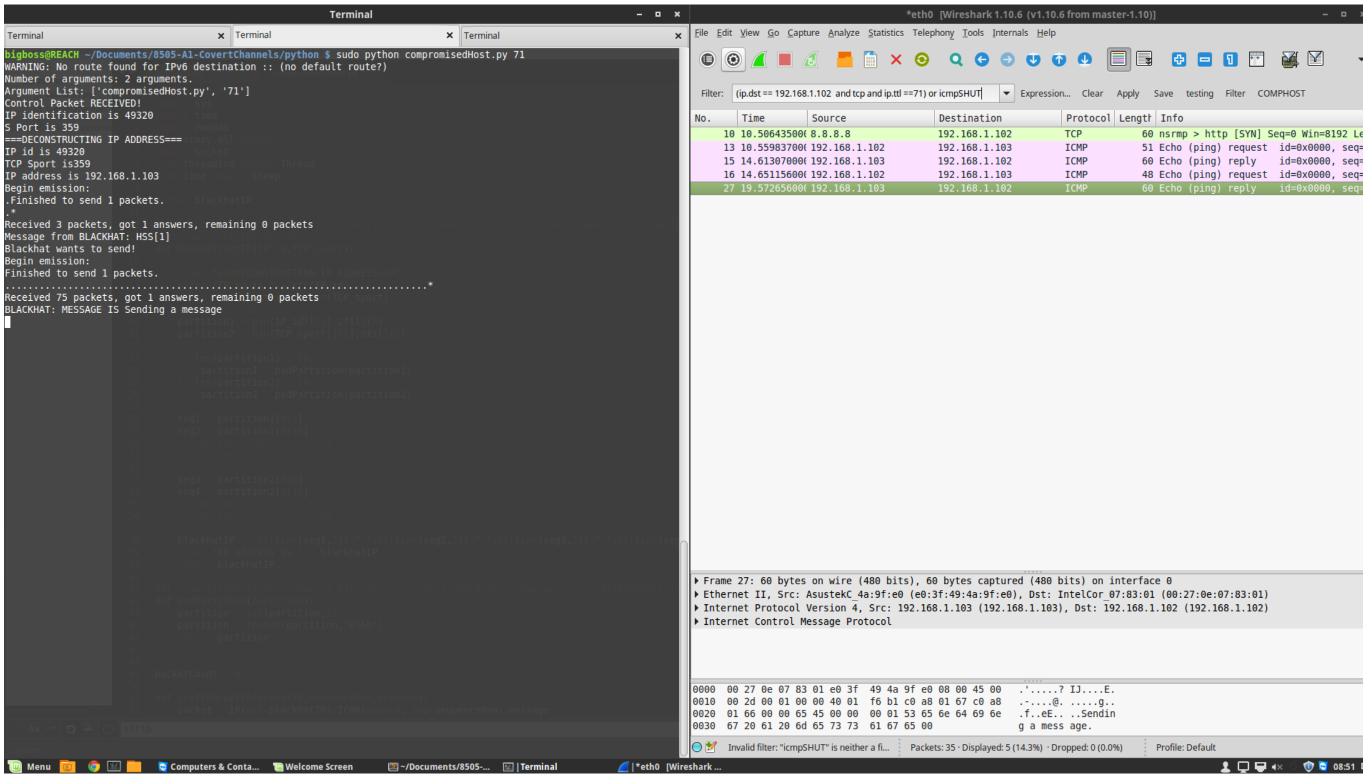
```
bigboss@Sol ~/Dropbox/BCIT Term 7/COMP 8505/Assignment 1/8505-A1-CovertChannels/python $ sudo python blackHat.py 192.
168.1.102 192.168.1.103 8.8.8.8 71
WARNING: No route found for IPv6 destination :: (no default route?)
Number of arguments: 5 arguments.
Argument List: ['blackHat.py', '192.168.1.102', '192.168.1.103', '8.8.8.8', '71']
192.168.1.103
1100000010101000
0000000101100111
IP ID is 49320
TCP Sport is 359
===DECONSTRUCTING IP ADDRESS===
IP id is 49320
TCP Sport is359
IP Address 192.168.1.103
TTL Key is 71
.
Sent 1 packets.
192.168.1.102
DEBUG: CLIENT MESSAGE IS: HANDSHAKE
--HANDSHAKE RECEIVED-- from SRC: 192.168.1.102
Connection Established
What would you like to do? Enter [1] to SEND, or [2] to RECEIVE
"SEND"
.
Sent 1 packets.
HANDSHAKE REPLY sent: BLACKHAT is going to SEND
192.168.1.102
DEBUG: CLIENT MESSAGE IS: CRR[1]
CLIENT: Ready to receive!
Please enter what message you would like to send the client.
"Sending a message"
.
Sent 1 packets.
```

Filter: 92.168.1.103 && ip.dst == 192.168.1.102) or icmpCOMPROMISED

```
No.   Time                        Source          Destination     Protocol Length Info
   38 2015-09-28 08:47:04.094219( 192.168.1.102   192.168.1.103   ICMP        60 Echo (ping) request
   46 2015-09-28 08:47:08.147179( 192.168.1.103   192.168.1.102   ICMP        48 Echo (ping) reply
   47 2015-09-28 08:47:08.185639( 192.168.1.102   192.168.1.103   ICMP        60 Echo (ping) request
   63 2015-09-28 08:47:13.107023( 192.168.1.103   192.168.1.102   ICMP        59 Echo (ping) reply

▸ Frame 63: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface 0
▸ Ethernet II, Src: AsustekC_4a:9f:e0 (e0:3f:49:4a:9f:e0), Dst: IntelCor_07:83:01 (00:27:0e:07:83:01)
▸ Internet Protocol Version 4, Src: 192.168.1.103 (192.168.1.103), Dst: 192.168.1.102 (192.168.1.102)
▸ Internet Control Message Protocol

0000  00 27 0e 07 83 01 e0 3f  49 4a 9f e0 08 00 45 00   .'.....? IJ....E.
0010  00 2d 00 01 00 00 40 01  f6 b1 c0 a8 01 67 c0 a8   .-....@. .....g..
0020  01 66 00 00 65 45 00 00  00 01 53 65 6e 64 69 6e   .f..eE.. ..Sendin
0030  67 20 61 20 6d 65 73 73  61 67 65                  g a mess age

Invalid filter: "icmpCOMPROMISED" is ...   ...   Profile: Default
```

Fig 1.1 Blackhat: Blackhat sending to CompromisedHost

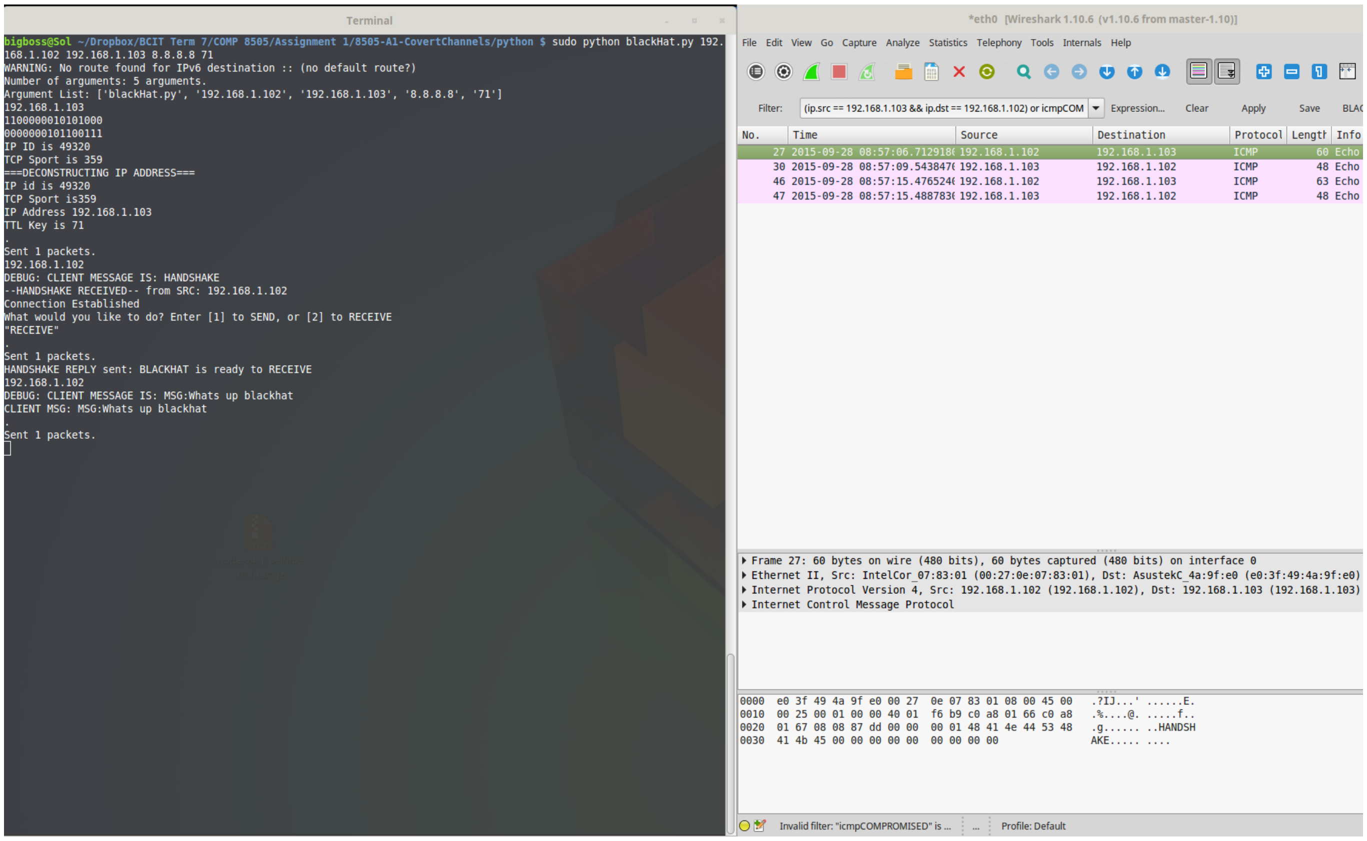
Thilina Ratnayake C8505 08 September 2015
A00802338 A1



```
bigboss@REACH ~/Documents/8505-A1-CovertChannels/python $ sudo python compromisedHost.py 71
WARNING: No route found for IPv6 destination :: (no default route?)
Number of arguments: 2 arguments.
Argument List: ['compromisedHost.py', '71']
Control Packet RECEIVED!
IP identification is 49320
S Port is 359
===DECONSTRUCTING IP ADDRESS===
IP id is 49320
TCP Sport is359
IP address is 192.168.1.103
Begin emission:
.Finished to send 1 packets.
.*
Received 3 packets, got 1 answers, remaining 0 packets
Message from BLACKHAT: HSS[1]
Blackhat wants to send!
Begin emission:
Finished to send 1 packets.
........................................................................*
Received 75 packets, got 1 answers, remaining 0 packets
BLACKHAT: MESSAGE IS Sending a message
```

```
*eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

Filter: (ip.dst == 192.168.1.102  and tcp and ip.ttl ==71) or icmpSHUT

No.  Time          Source          Destination     Protocol Length Info
  10 10.50643500(  8.8.8.8         192.168.1.102   TCP          60 nsrmp > http [SYN] Seq=0 Win=8192 Le
  13 10.55983700(  192.168.1.102   192.168.1.103   ICMP         51 Echo (ping) request  id=0x0000, seq=
  15 14.61307000(  192.168.1.103   192.168.1.102   ICMP         60 Echo (ping) reply    id=0x0000, seq=
  16 14.65115600(  192.168.1.102   192.168.1.103   ICMP         48 Echo (ping) request  id=0x0000, seq=
  27 19.57265600(  192.168.1.103   192.168.1.102   ICMP         60 Echo (ping) reply    id=0x0000, seq=

▸ Frame 27: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: AsustekC_4a:9f:e0 (e0:3f:49:4a:9f:e0), Dst: IntelCor_07:83:01 (00:27:0e:07:83:01)
▸ Internet Protocol Version 4, Src: 192.168.1.103 (192.168.1.103), Dst: 192.168.1.102 (192.168.1.102)
▸ Internet Control Message Protocol

0000  00 27 0e 07 83 01 e0 3f  49 4a 9f e0 08 00 45 00   .'.....? IJ....E.
0010  00 2d 00 01 00 00 40 01  f6 b1 c0 a8 01 67 c0 a8   .-....@. .....g..
0020  01 66 00 00 65 45 00 00  00 01 53 65 6e 64 69 6e   .f..eE.. ..Sendin
0030  67 20 61 20 6d 65 73 73  61 67 65 00               g a mess age.

Invalid filter: "icmpSHUT" is neither a fi...   Packets: 35 · Displayed: 5 (14.3%) · Dropped: 0 (0.0%)   Profile: Default
```

Figure1.2 Client: Blackhat sending to Client



```
bigboss@Sol ~/Dropbox/BCIT Term 7/COMP 8505/Assignment 1/8505-A1-CovertChannels/python $ sudo python blackHat.py 192.
168.1.102 192.168.1.103 8.8.8.8 71
WARNING: No route found for IPv6 destination :: (no default route?)
Number of arguments: 5 arguments.
Argument List: ['blackHat.py', '192.168.1.102', '192.168.1.103', '8.8.8.8', '71']
192.168.1.103
1100000010101000
0000000101100111
IP ID is 49320
TCP Sport is 359
===DECONSTRUCTING IP ADDRESS===
IP id is 49320
TCP Sport is359
IP Address 192.168.1.103
TTL Key is 71
.
Sent 1 packets.
192.168.1.102
DEBUG: CLIENT MESSAGE IS: HANDSHAKE
--HANDSHAKE RECEIVED-- from SRC: 192.168.1.102
Connection Established
What would you like to do? Enter [1] to SEND, or [2] to RECEIVE
"RECEIVE"
.
Sent 1 packets.
HANDSHAKE REPLY sent: BLACKHAT is ready to RECEIVE
192.168.1.102
DEBUG: CLIENT MESSAGE IS: MSG:Whats up blackhat
CLIENT MSG: MSG:Whats up blackhat
.
Sent 1 packets.
```

*eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

Filter: (ip.src == 192.168.1.103 && ip.dst == 192.168.1.102) or icmpCOM

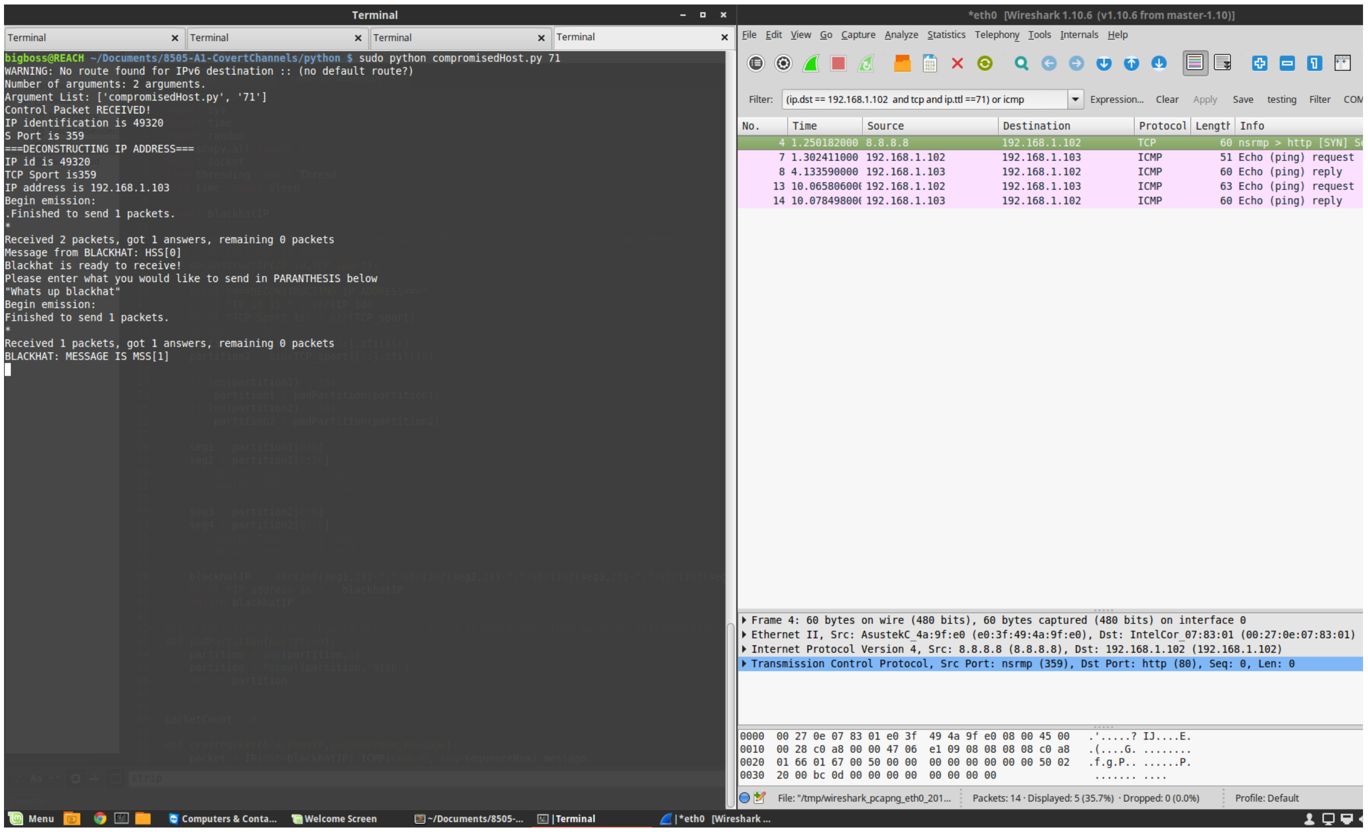
```
No.   Time                        Source           Destination      Protocol  Lengtł  Info
   27 2015-09-28 08:57:06.712918( 192.168.1.102    192.168.1.103    ICMP          60  Echo
   30 2015-09-28 08:57:09.543847( 192.168.1.103    192.168.1.102    ICMP          48  Echo
   46 2015-09-28 08:57:15.476524( 192.168.1.102    192.168.1.103    ICMP          63  Echo
   47 2015-09-28 08:57:15.488783( 192.168.1.103    192.168.1.102    ICMP          48  Echo
```

```
▸ Frame 27: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: IntelCor_07:83:01 (00:27:0e:07:83:01), Dst: AsustekC_4a:9f:e0 (e0:3f:49:4a:9f:e0)
▸ Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 192.168.1.103 (192.168.1.103)
▸ Internet Control Message Protocol
```

```
0000  e0 3f 49 4a 9f e0 00 27  0e 07 83 01 08 00 45 00   .?IJ...' ......E.
0010  00 25 00 01 00 00 40 01  f6 b9 c0 a8 01 66 c0 a8   .%....@. .....f..
0020  01 67 08 08 87 dd 00 00  00 01 48 41 4e 44 53 48   .g...... ..HANDSH
0030  41 4b 45 00 00 00 00 00  00 00 00 00               AKE..... ....
```

Invalid filter: "icmpCOMPROMISED" is ... Profile: Default

Fig 2.1. Blackhat: Client sending to Blackhat


Terminal
*eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save testing Filter

```
bigboss@REACH ~/Documents/8505-A1-CovertChannels/python $ sudo python compromisedHost.py 71
WARNING: No route found for IPv6 destination :: (no default route?)
Number of arguments: 2 arguments.
Argument List: ['compromisedHost.py', '71']
Control Packet RECEIVED!
IP identification is 49320
S Port is 359
===DECONSTRUCTING IP ADDRESS===
IP id is 49320
TCP Sport is359
IP address is 192.168.1.103
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Message from BLACKHAT: HSS[0]
Blackhat is ready to receive!
Please enter what you would like to send in PARANTHESIS below
"Whats up blackhat"
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
BLACKHAT: MESSAGE IS MSS[1]
```

```
Filter: (ip.dst == 192.168.1.102  and tcp and ip.ttl ==71) or icmp

No.  Time         Source         Destination    Protocol Length Info
  4  1.250182000  8.8.8.8        192.168.1.102  TCP      60 nsrmp > http [SYN] Se
  7  1.302411000  192.168.1.102  192.168.1.103  ICMP     51 Echo (ping) request
  8  4.133590000  192.168.1.103  192.168.1.102  ICMP     60 Echo (ping) reply
 13  10.06580600( 192.168.1.102  192.168.1.103  ICMP     63 Echo (ping) request
 14  10.07849800( 192.168.1.103  192.168.1.102  ICMP     60 Echo (ping) reply

▸ Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: AsustekC_4a:9f:e0 (e0:3f:49:4a:9f:e0), Dst: IntelCor_07:83:01 (00:27:0e:07:83:01)
▸ Internet Protocol Version 4, Src: 8.8.8.8 (8.8.8.8), Dst: 192.168.1.102 (192.168.1.102)
▸ Transmission Control Protocol, Src Port: nsrmp (359), Dst Port: http (80), Seq: 0, Len: 0

0000  00 27 0e 07 83 01 e0 3f  49 4a 9f e0 08 00 45 00   .'.....? IJ....E.
0010  00 28 c0 a8 00 00 47 06  e1 09 08 08 08 08 c0 a8   .(....G. ........
0020  01 66 01 67 00 50 00 00  00 00 00 00 00 00 50 02   .f.g.P.. ......P.
0030  20 00 bc 0d 00 00 00 00  00 00 00 00               ....... ....

File: "/tmp/wireshark_pcapng_eth0_201...   Packets: 14 · Displayed: 5 (35.7%) · Dropped: 0 (0.0%)   Profile: Default
```

Fig 2.2 Client: Client Sending To Blackhat