

Thilina Ratnayake  
A00802338

C8505  
A3

17 October 2015

**Thilina Ratnayake**  
**COMP 8505**  
**A3**

# Table of Contents

---

## Table of Contents

<b>INTRODUCTION</b>	<b>3</b>
<b>FEATURES:</b>	<b>3</b>
<b>PRACTICAL APPLICATION</b>	<b>3</b>
<b>USAGE</b>	<b>4</b>
<b>REQUIREMENTS:</b>	<b>4</b>
<b>SENDING &amp; RECEIVING DATA</b>	<b>4</b>
<b>RESULTS</b>	<b>5</b>
<b>DIAGRAMS</b>	<b>6</b>
<b>CODE LISTINGS</b>	<b>6</b>
<b>PSUEDOCODE</b>	<b>7</b>
BLACKHAT.PY:	7
CLIENT.PY:	7
<b>TESTS</b>	<b>7</b>
<b>FIGURES</b>	<b>8</b>

## Introduction

Backsniffer is a covert communication suite that allows communication between an attacker and a backdoor application on a target's compromised machine.

Backsniffer contains two modules:

1. Blackhat.py – This is the module that sends commands to the target and waits for replies.
2. Client.py – The actual “Back-Door” that can be run on the client machine.

### Features:

Some features of the Backsniffer suite include:

- **Firewall Evasion:**  
Commands are able to get through to the target machine even with a running firewall due to the use of raw-sockets to sniff for packets.
- **Process Masking:**  
The back-door module running on the client's machine can camouflage itself by changing the name of its process. This allows it to remain invisible through usual detection methods such as running 'ps aux'.
- **Authentication**  
By checking for a pre-determined TTL and destination port, there are two layers of authentication to ensure that the backdoor only picks up messages that are meant for it.
- **AES 256 Bit Encryption**  
All messages sent between the client and backdoor are encrypted using AES 256 bit encryption to mitigate any chance of easy discovery via packet captures.

### Practical Application

A typical scenario for this application would be uploading and executing the client.py application on a target machine via an entry vector of one's choice (social-engineering etc)

Once run, the backdoor will disguise its-self with a process name as set by the attacker so as to evade detection on any process lists. After it has masked its process name, it will listen on raw sockets for packets from the attackers, which match a specific signature. The signature is a combination of 2 facts, the IP header's TTL and destination port it is being received on. As mentioned above, this ensures that messages get through any personal firewall that is running on the compromised system.

Packets then get decrypted using a pre-shared decryption key producing the command which is executed and sent back using the same encryption sequence. This allows an attacker to virtually have a remote shell on a compromised system.

## Usage

### Requirements:

1. Backsniffer requires that the following python libraries be installed
  - a. PyCrypto
  - b. Setproctitle
  - c. Scapy
2. This can all be installed by running the shell script:
  - a. `sh startup.sh`

### Sending & Receiving Data

1. Starting the backdoor.

On the target machine, enter command: `python client.py 80 71 012345689abcdef abcdefghijklmnop [KWorker2:0]`

The first two arguments that we are listening for packets that have the characteristics: (1) incoming to port 80 and (2) have a TTL of 71. This is the criteria to specify that the packets are from the attacker.

Secondly, we then enter in our pre-shared encryption key and initialization vectors. These two parameters allows us to decrypt and encrypt the messages between attacker and victim.

We instruct the backdoor to start-up with the process name "[KWorker2:0]". This is because on our test systems, there are multiple `kworker*.*` processes running at any given time, and choosing the name specified will be able to easily mask the process. The first two letters have been highlighted to be able to recognize the process in order to kill it later.

2. Starting the attacker's shell.

On the attackers machine, enter command: `python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmnop`

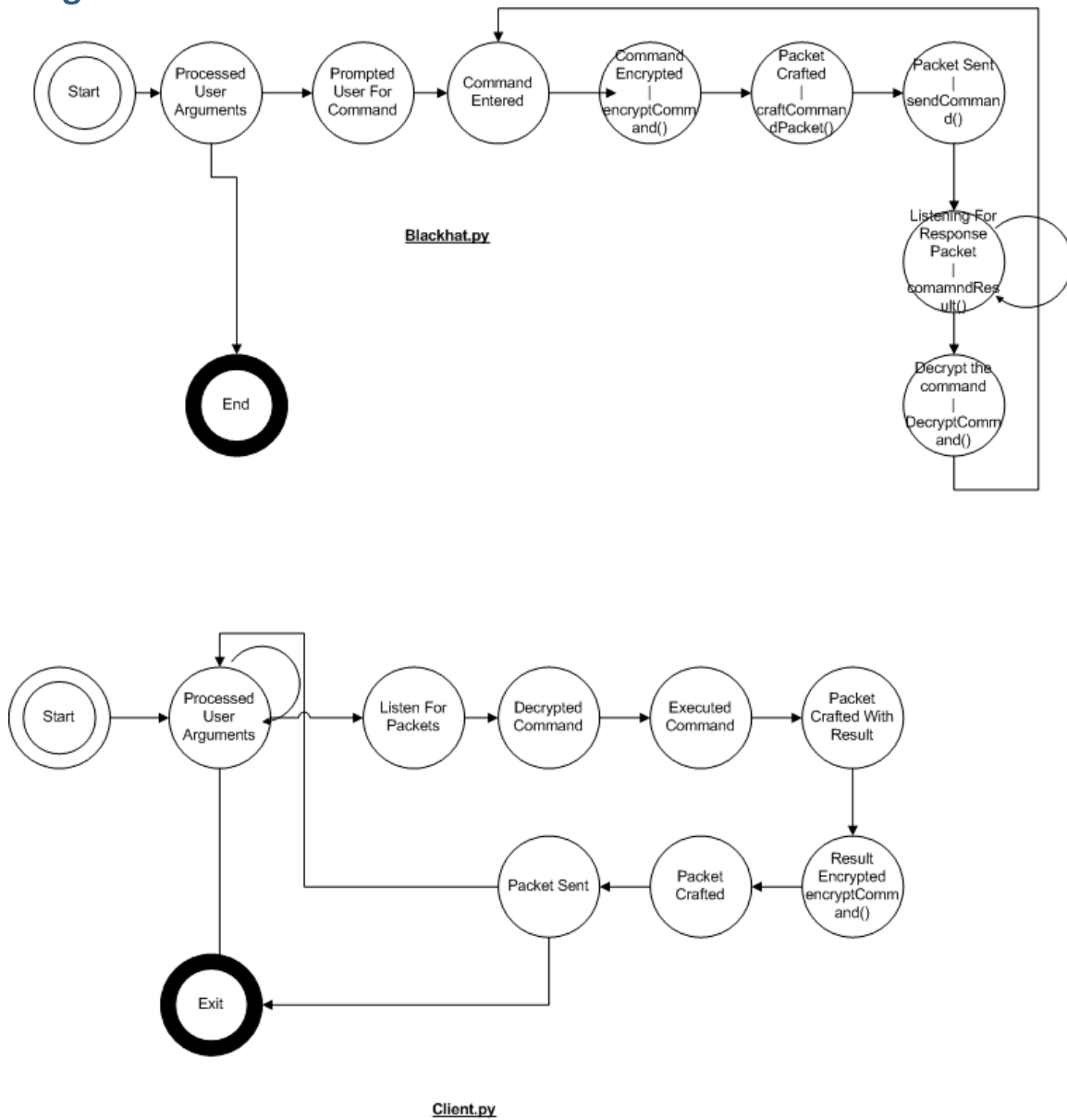
After the connection info has been entered, the attacker can simply begin entering in commands as if he or she were utilizing a shell on the victims machine.

## Results

After running tests and experiments, the results show that Backsniffer is able to:

- Send encrypted messages between the two systems.
- Execute commands on the client machine and send the output back.
- Evade a firewall that is dropping all packets.

## Diagrams



## Code Listings

The Backsniffer application is split up into two modules:

1. Blackhat.py – This is the module that sends commands to the target and waits for replies.
2. Client.py – The actual “Back-Door” that can be run on the client machine.

## Psuedocode

### Blackhat.py:

1. Take command line arguments
2. Prompt use for message
3. Encrypt message
4. Craft packet to with source port, destination port, and ttl specified in step 1
5. Send Packet
6. Listen for a response from the client
7. Decrypt the received packet
8. Display the result

### Client.py:

1. Take command line arguments
2. Set process title to what is specified in 1
3. Listen on the port specified in 1
4. For each packet received on that packet, check it against the signature specified in 1 (ttl and src port)
  - a. If a packet matches the signatures
    - i. Decrypt the command from packet
    - ii. Run command
    - iii. Get output
    - iv. Craft a packet destined for the source that the incoming packet came from.
    - v. Encrypt result
    - vi. Put result into packet
    - vii. Send packet
5. Listen for new packets

## Tests

#	Test / Resource(s)	Command	Expected	Result	Figs
1	Process name is masked. – Client.py	<code>python client.py 8071012345689abcdef abcdefghijklm nop [KWorker2:0]</code>	<p>Ps aux   grep “client.py” or “python.py” should not show any processes.</p> <p>Ps aux   grep “KWorker” should.</p>	PASS	1.1-1.3

2	Command's are received sent from the attacker to the client on the specified ports – Blackhat.py & Client.py	<b>python blackhat.py 192.168.0.3 500 80 71 0123456789ab cdef abcdefghijkl mnop</b>	Packet should be going from source 500 to port 80	PASS	2.1
3	Commands sent/received from the attacker are encrypted. Client.py	Same as 2	Data in packet payload should not be readable.	PASS	3.1,3.2,3.3
4	The right command is received and executed	Same as 2.	The outputs of the directory should be displayed.	PASS	4.1
5	The results are sent back and are encrypted	Same as 2	The outputs of the response are encrypted	PASS	4.5
6	Commands that produce no output send back the placeholder text	Client: <b>python client.py 80 71 012345689abcd ef abcdefghijklm nop [KWorker2:0]</b>  Server: <b>python blackhat.py 192.168.0.3 500 80 71 0123456789ab cdef abcdefghijkl mnop</b>	The commands executed should throw an error as they are decrypted/encrypted with the wrong key.	PASS	6.1, 6.2

## Figures & Tests

### Test 1

This test was to check that the backdoor program on the victims machine is able to successfully mask it's process name. In this case, to [KWorker2:0] as set in the command line arguments.



```
[root@datacomm Backsniffer3]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy DROP)
target     prot opt source                destination
[root@datacomm Backsniffer3]# python client.py 80 71 0123456789abcdef abcdefghijklmnop
[KWorker2:0]
dstPort is 80
ttlKey is 71
Decryption key is 0123456789abcdef
IV is abcdefghijklmnop
Process Name is [KWorker2:0]
```

Fig 1.1 – Client output after backdoor has been started.

```
[root@datacomm Backsniffer3]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy DROP)
target     prot opt source                destination
[root@datacomm Backsniffer3]# ps aux | grep "Python"
root      14975  0.0  0.0 114328  2288 pts/2    S+   17:48   0:00 grep --color=auto Python
[root@datacomm Backsniffer3]# ps aux | grep "client.py"
root      14983  0.0  0.0 114328  2292 pts/2    S+   17:49   0:00 grep --color=auto client.py
[root@datacomm Backsniffer3]# ps aux | grep -w "KWorker2"
root      14471  0.0  0.3 256248 28392 pts/1    S+   17:42   0:00 [KWorker2:0]
root      14991  0.0  0.0 114328  2292 pts/2    S+   17:49   0:00 grep --color=auto -w KWorker2
[root@datacomm Backsniffer3]# ps aux | grep -w "KWorker"
```

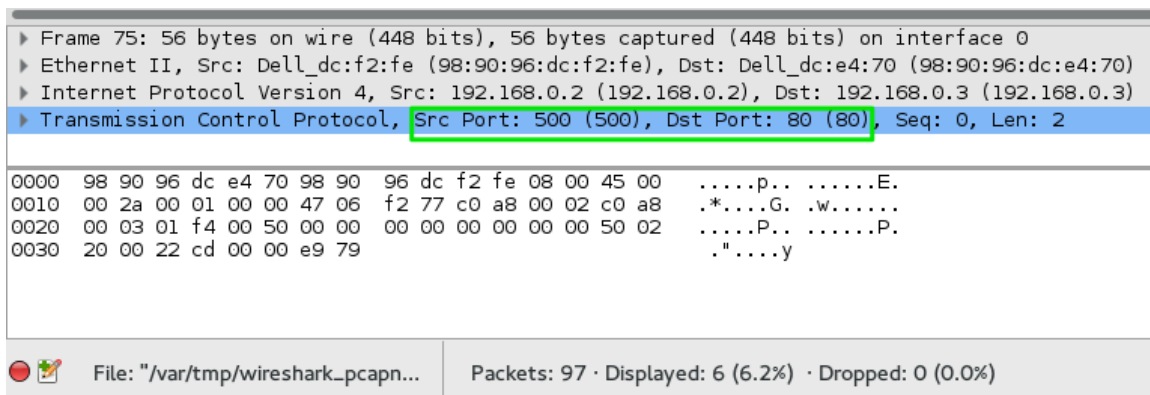
Fig 1.2- Client terminal showing that the process is not shown as client.py or as a python script. Client process is “KWorker2:0” as specified in the initial command.

```
[root@datacomm Backsniffer3]# ps aux | grep "orker"
root      5  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/0:0H]
root     17  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/1:0H]
root     24  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/2:0H]
root     31  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/3:0H]
root    333  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/1:1H]
root    334  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/0:1H]
root    345  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/3:1H]
root    443  0.0  0.0      0  0 ?    S<   14:01   0:00 [kworker/2:1H]
root    911  0.0  0.0 358748 7900 ?    Sl   14:01   0:00 gdm-session-worker [pa
m/gdm-launch-environment]
root   1199  0.0  0.1 373400 8408 ?    Sl   14:02   0:00 gdm-session-worker [pa
m/gdm-password]
root   13834 0.0  0.0      0  0 ?    S    17:11   0:00 [kworker/0:2]
root   13840 0.0  0.0      0  0 ?    S    17:12   0:00 [kworker/1:2]
root   13988 0.0  0.0      0  0 ?    S    17:21   0:00 [kworker/1:0]
root   13991 0.0  0.0      0  0 ?    S    17:22   0:00 [kworker/u8:1]
root   14059 0.0  0.0      0  0 ?    S    17:28   0:00 [kworker/0:1]
root   14081 0.0  0.0      0  0 ?    S    17:30   0:00 [kworker/2:1]
root   14135 0.0  0.0      0  0 ?    S    17:33   0:00 [kworker/3:1]
root   14188 0.0  0.0      0  0 ?    S    17:38   0:00 [kworker/u8:2]
root   14268 0.0  0.0      0  0 ?    S    17:39   0:00 [kworker/2:0]
root   14471 0.0  0.3 256248 28392 pts/1  S+   17:42   0:00 [Kworker2:0]
root   14592 0.0  0.0      0  0 ?    S    17:43   0:00 [kworker/u8:0]
root   14689 0.0  0.0      0  0 ?    S    17:44   0:00 [kworker/3:0]
root   15015 0.0  0.0 114332 2284 pts/2  S+   17:49   0:00 grep --color=auto orke
r
```

Fig 1.3 – Terminal displayed shows all other running KWorker threads. This puts into perspective what it would be like for a user or analyst viewing currently open processes.

## Test 2

This test was to check whether packets sent from the attacker to the victim were being sent to and from the correct ports. Specifically, from source port 500 to destination port 80 as per command line arguments.



The image shows a Wireshark packet capture interface. The top pane displays packet details for Frame 75, which is a Transmission Control Protocol (TCP) packet. The packet is 56 bytes on wire (448 bits) and 56 bytes captured (448 bits) on interface 0. The Ethernet II header shows the source as Dell\_dc:f2:fe (98:90:96:dc:f2:fe) and the destination as Dell\_dc:e4:70 (98:90:96:dc:e4:70). The Internet Protocol Version 4 header shows the source as 192.168.0.2 (192.168.0.2) and the destination as 192.168.0.3 (192.168.0.3). The Transmission Control Protocol header shows the source port as 500 (500) and the destination port as 80 (80), with a sequence number of 0 and a length of 2. The bottom pane shows the raw packet data in hexadecimal and ASCII format.

```

0000  98 90 96 dc e4 70 98 90 96 dc f2 fe 08 00 45 00  ....p.. ....E.
0010  00 2a 00 01 00 00 47 06 f2 77 c0 a8 00 02 c0 a8  .*....G. .w.....
0020  00 03 01 f4 00 50 00 00 00 00 00 00 00 00 50 02  ....P.. ....P.
0030  20 00 22 cd 00 00 e9 79  ....y

```

The bottom status bar indicates the file is "/var/tmp/wireshark\_pcapn...", with 97 packets displayed, 6 (6.2%) displayed, and 0 (0.0%) dropped.

Fig 2.1 – The packets are being sent from the right source port to the right destination port

### Test 3

This test was to check that the messages being sent between attacker and victim are encrypted and can't be read via an analyst viewing packet captures.

```
[root@datacomm Backsniffer3]# python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmnop
WARNING: No route found for IPv6 destination :: (no default route?)
START Victim IP is 192.168.0.3
START Sending from Blackhat port: 500
Send to destination port: 80
TTL Key is 71
Encryption key is 0123456789abcdef
IV is abcdefghijklmnop
ENTER COMMAND -> 192.168.0.3:ls
```

Fig 3.1 – Terminal Output showing the command that the attacker has sent.

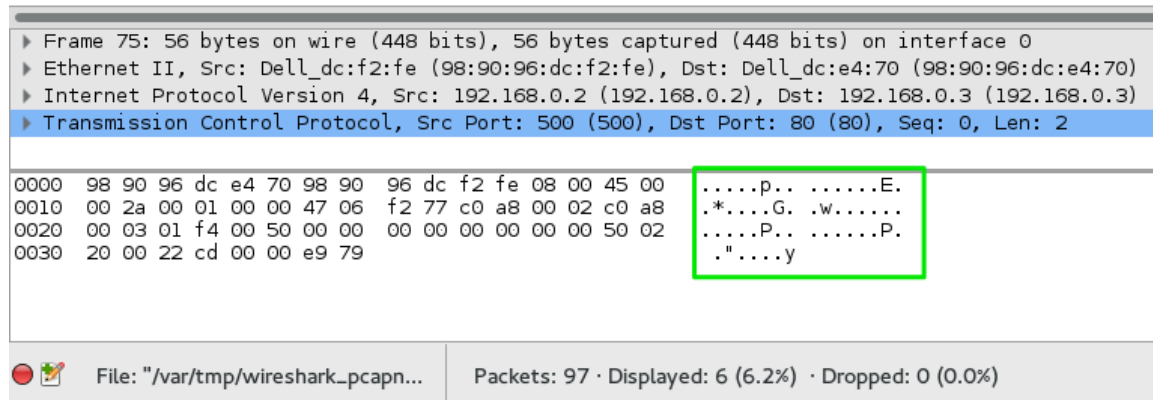


Fig 3.2 – The Wireshark capture at the attacker showing that the packet's data is not in plain-text.

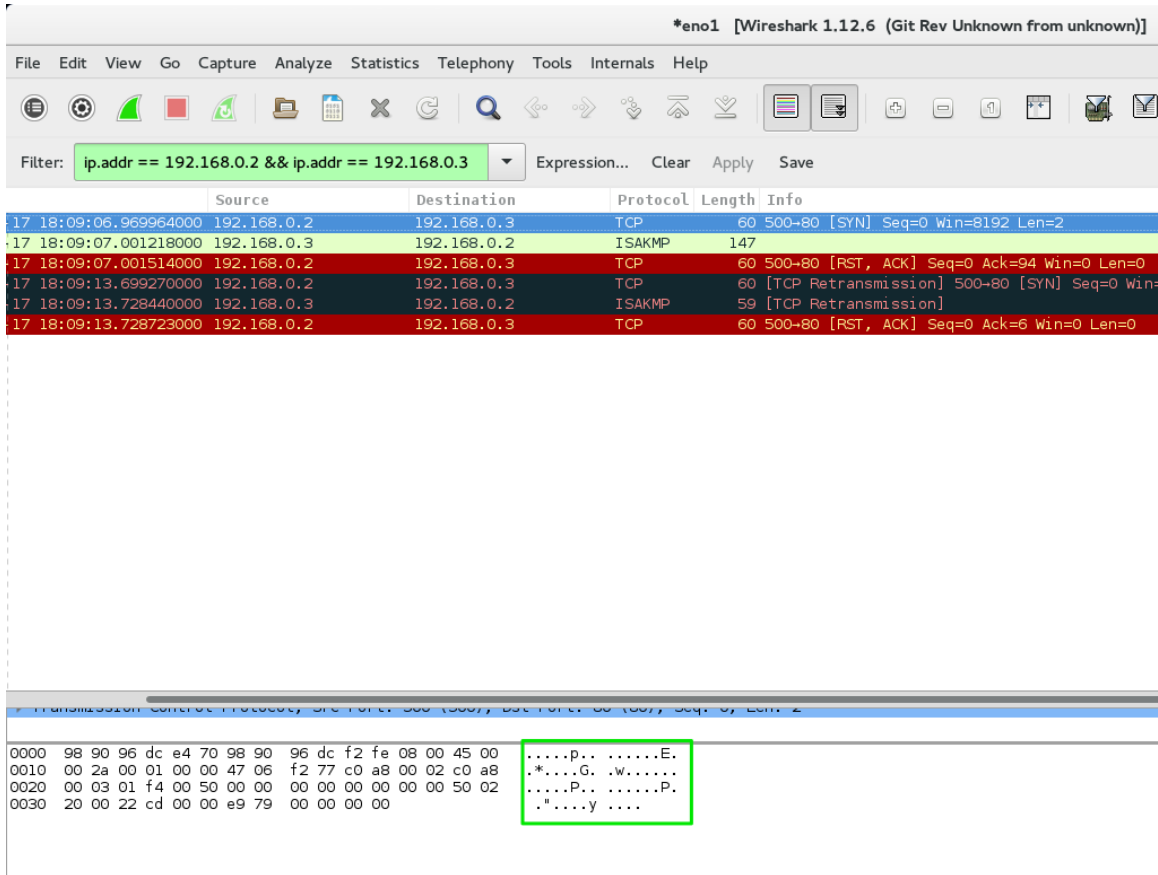


Fig 3.3 – The Wireshark capture at the client showing that the received packet is encrypted after being in transit.

Test 4 & 5:

This test checks to see whether the command is being executed on the victims system and whether results are being sent back. Figure 4.5 is part of Test 5 which shows that messages from the victim to the attacker are encrypted as well.

```
IV is abcdefghijklmnop
ENTER COMMAND -> 192.168.0.3:ls
blackhat.py
client.py
password.txt
rsync.sh
startup.sh
TestCases.txt
Tests
tests.sh
test.txt

ENTER COMMAND -> 192.168.0.3:whoami
root

ENTER COMMAND -> 192.168.0.3:exit
[root@datacomm Backsniffer3]#
```

Fig 4.1 – Results are sent back.

The screenshot displays two windows. The top window is Wireshark 1.12.6, capturing traffic on interface eno1. A filter is applied: `ip.addr == 192.168.0.3 && ip.addr == 192.168.0.3`. The packet list shows several packets between 192.168.0.2 and 192.168.0.3. The bottom window is a terminal titled `root@datacomm:~/Documents/Backsniffer3`. It shows the execution of `client.py` which sends back the results of the commands: `ls` (listing files like `blackhat.py`, `client.py`, etc.) and `whoami` (returning `root`).

No.	Time	Source	Destination
91	2015-10-17 19:43:27.344855000	192.168.0.2	192.168.0.3
92	2015-10-17 19:43:27.345178000	192.168.0.3	192.168.0.2
93	2015-10-17 19:43:27.366817000	192.168.0.3	192.168.0.2
94	2015-10-17 19:43:27.366854000	192.168.0.2	192.168.0.3
283	2015-10-17 19:44:29.141784000	199.212.24.20	192.168.0.3
284	2015-10-17 19:44:29.145467000	199.212.24.20	192.168.0.3

Fig 4.5 – The result coming back is encrypted

### Test 6

This test shows the results of the attacker or victim using incorrect encryption/decryption keys & Initialization Vector. If one party does not use the right key, then the commands interpreted and results sent back will be different.

It can be seen in the screenshots that the attacker uses initialization vector “abcdefghijklmnopqrstuvwxyz” whereas the victim uses “abcdefghijklmnopqrstuvwxyz”. This leads to a different result when decrypted, which explains the output seen in Fig 6.2

```
[root@datacomm Backsniffer3]# python client.py 80 71 0123456789abcdef abcdefghijklmnop [KWorker2:0]
dstPort is 80
ttlKey is 71
Decryption key is 0123456789abcdef
IV is abcdefghijklmnop
Process Name is [KWorker2:0]
sh: $'\271\355': command not found
sh: $'\242h\203\331\027\300': command not found
```

Fig 6.1 – Client: Backdoor start with different key.

```
[root@datacomm Backsniffer3]# python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmnopQ
WARNING: No route found for IPv6 destination :: (no default route?)
START Victim IP is 192.168.0.3
START Sending from Blackhat port: 500
Send to destination port: 80
TTL Key is 71
Encryption key is 0123456789abcdef
IV is abcdefghijklmnopQ
ENTER COMMAND -> 192.168.0.3:ls
0)
020 0000y2ut Produced
ENTER COMMAND -> 192.168.0.3:whoami
0)
020 0000y2ut Produced
ENTER COMMAND -> 192.168.0.3:
```

Fig 6.2 – Attacker: The result is wrong if the wrong encryption keys and decryption keys are specified