

Covert Communications

Reliable Covert Communications over DNS with AES 128 Encryption

Dhishan Amaranath

N16909360

Objective:

- Communicate by sending data over any IP protocol without using the data fields in any packet
- Provide CIA using any encryption algorithms
- Handshake like protocol for initiating and terminating the communications
- Provide Reliability for the communication

Basic Covert Communication

DNS is one of the protocol usually the Firewall have an forward rule. So sending and receiving a packet using the DNS would be a perfect cover for covert Communication. Also using DNS query and DNS response would be a best way for Reliable two way Communication.

The byte of data is divided appropriately to fit into the **flags** of the DNS packet.

Any one who is observing using the wireshark would just see a set of DNS queries and DNS responses and would be a perfect covert message.

No.	Time	Source	Destination	Protocol	Length	Info
2	2...	10.10.111.104	10.10.111.103	DNS	79	Unknown operation (7) 0xab7b A www.oqzyencmeva.ru
5	7...	10.10.111.103	10.10.111.104	DNS	114	Unknown operation (7) response 0xab7b A www.oqzyencmeva.ru[Packet siz...

17	18	19	20	21	22	23	24
Opcode				AA	TC	RD	RA

The 8 bits of the data is fit into the following flag values:

The message is hidden inside the flags field:

▼ Flags: 0xbc80 Unknown operation response, No error

- 1... .. = Response: Message is a response
- .011 1... .. = Opcode: Unknown (7)
-1.. = Authoritative: Server is an authority for domain
-0. = Truncated: Message is not truncated
-0 = Recursion desired: Don't do query recursively
- 1... .. = Recursion available: Server can do recursive queries
-0.. = Z: reserved (0)
-0. = Answer authenticated: Answer/authority portion was not authenticated by the server
-0 = Non-authenticated data: Unacceptable
- 0000 = Reply code: No error (0)

Even though they get suspicious of the many DNS queries, since the byte values are hidden in the flag field, It would be hard for the analyzer to decode. With Encryption is near to impossible

Note: The Screenshot displays unknown operation. This is because the scapy on the bt5 and dvl are older and wouldnt allow use of all the flag fields. The following is a DNS packet crafted on a newer version of scapy in personnel computer

No.	Time	Source	Destination	Protocol	Length	Info
6...	6...	172.16.185.208	8.8.8.8	DNS	79	Standard query 0x44a6 A dns2dtejenuxggnu.ch
6...	6...	8.8.8.8	172.16.185.208	DNS	130	Standard query response 0x44a6 No such name A dns2dtejenuxggnu.ch S0A ...

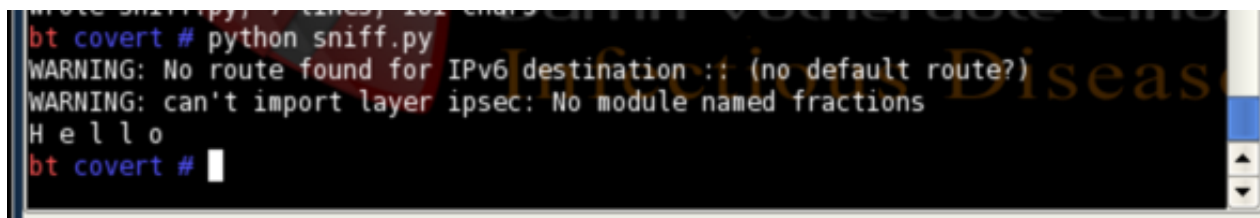
Validity of the packet and choosing the correct sender

In a network with multiple hosts, the traffic would be enormous and filter argument in the scapy sniff function for accepting only IP's from the expected receiver is a good choice.

```
a=sniff(filter="src host 10.10.111.103")
```

Sending the Message as a DNS response for the query address with same DNS id would separate the covert traffic from the other traffic in the network

Screenshot of the result:

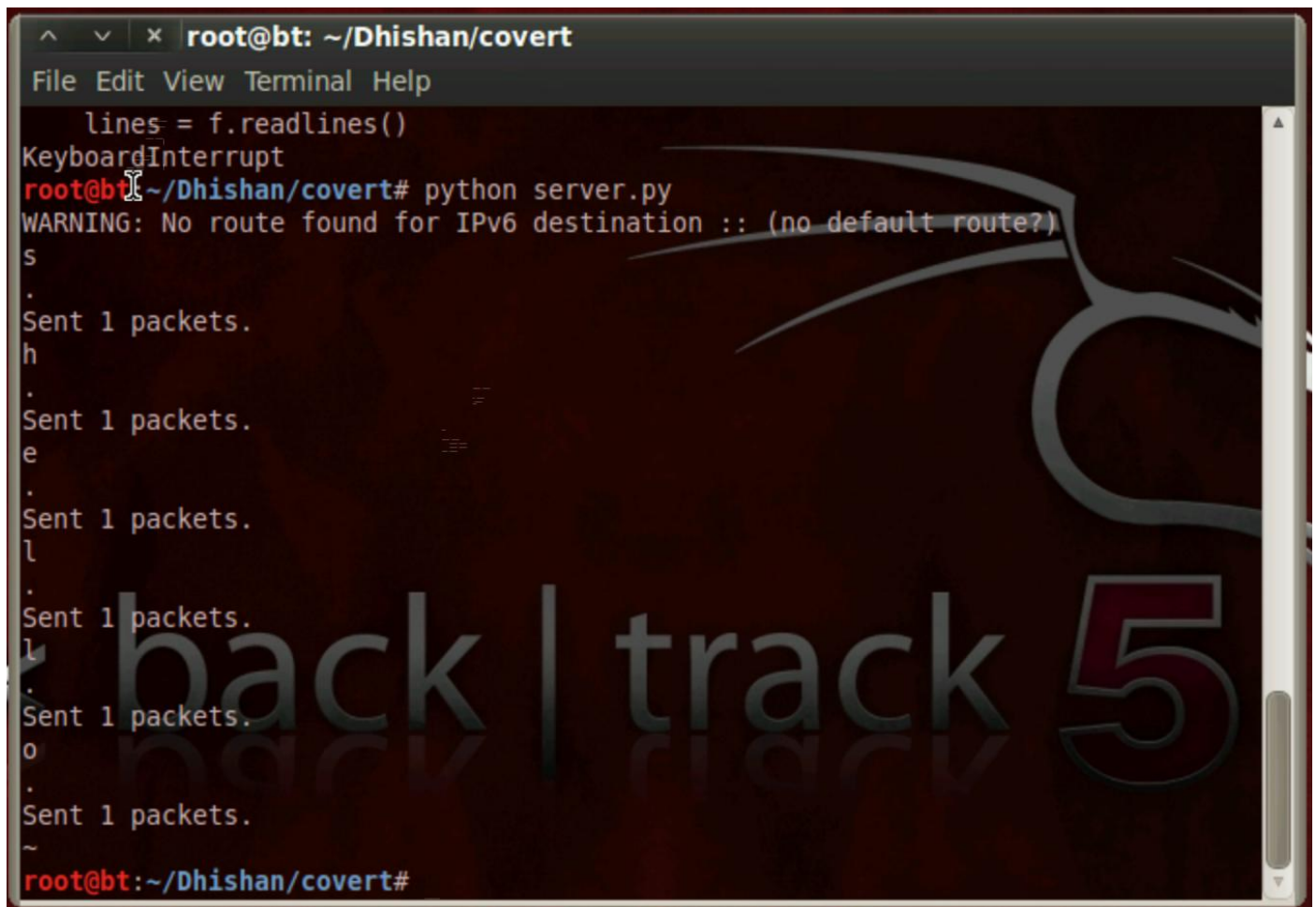


Handshake for initiating and Termination

Handshake

The Client Sends a character **s** in the initial DNS query with an **Ephemeral** source port number on the UDP, informing the server that it is Listening for the first character. The server sends DNS responses character hidden in the flag fields with the same DNS id and directed to the same destination port number where it received from.

Screenshot of the Result Sent **s** & **~**



```
root@bt: ~/Dhishan/covert
File Edit View Terminal Help
lines = f.readlines()
KeyboardInterrupt
root@bt:~/Dhishan/covert# python server.py
WARNING: No route found for IPv6 destination :: (no default route?)
s
.
Sent 1 packets.
h
.
Sent 1 packets.
e
.
Sent 1 packets.
l
.
Sent 1 packets.
l
.
Sent 1 packets.
o
.
Sent 1 packets.
~
root@bt:~/Dhishan/covert#
```

Termination

The server sends an **~** character at the end of the message indicating the end of the communication.

Screenshot of the Result Recieved **s** & **~**

```
Shell - Konsole
WARNING: can't import layer ipsec: No module named fractions
/usr/local/lib/python2.5/site-packages/Crypto/Util/number.py:57: PowmInsecureWarning: Not using mpz_powm_sec. You should rebuild using libgmp >= 5 to avoid timing attack vulnerability.
  _warn("Not using mpz_powm_sec. You should rebuild using libgmp >= 5 to avoid timing attack vulnerability.", PowmInsecureWarning)

.
Sent 1 packets.
www.xmvfxherxeto.ru
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
hello~
```



Reliability

The communication is over UDP where there is no reliability. Here I have used IP ident field and the DNS id field for providing the reliability. Here how it works:

- The client sends the server with an DNS query with the message character **s** embedded inside the flags field.
- The server creates a new packet with a new IPv4 id value and crafts the DNS response with the same DNS id as the query, with the first character embedded in its flag field.

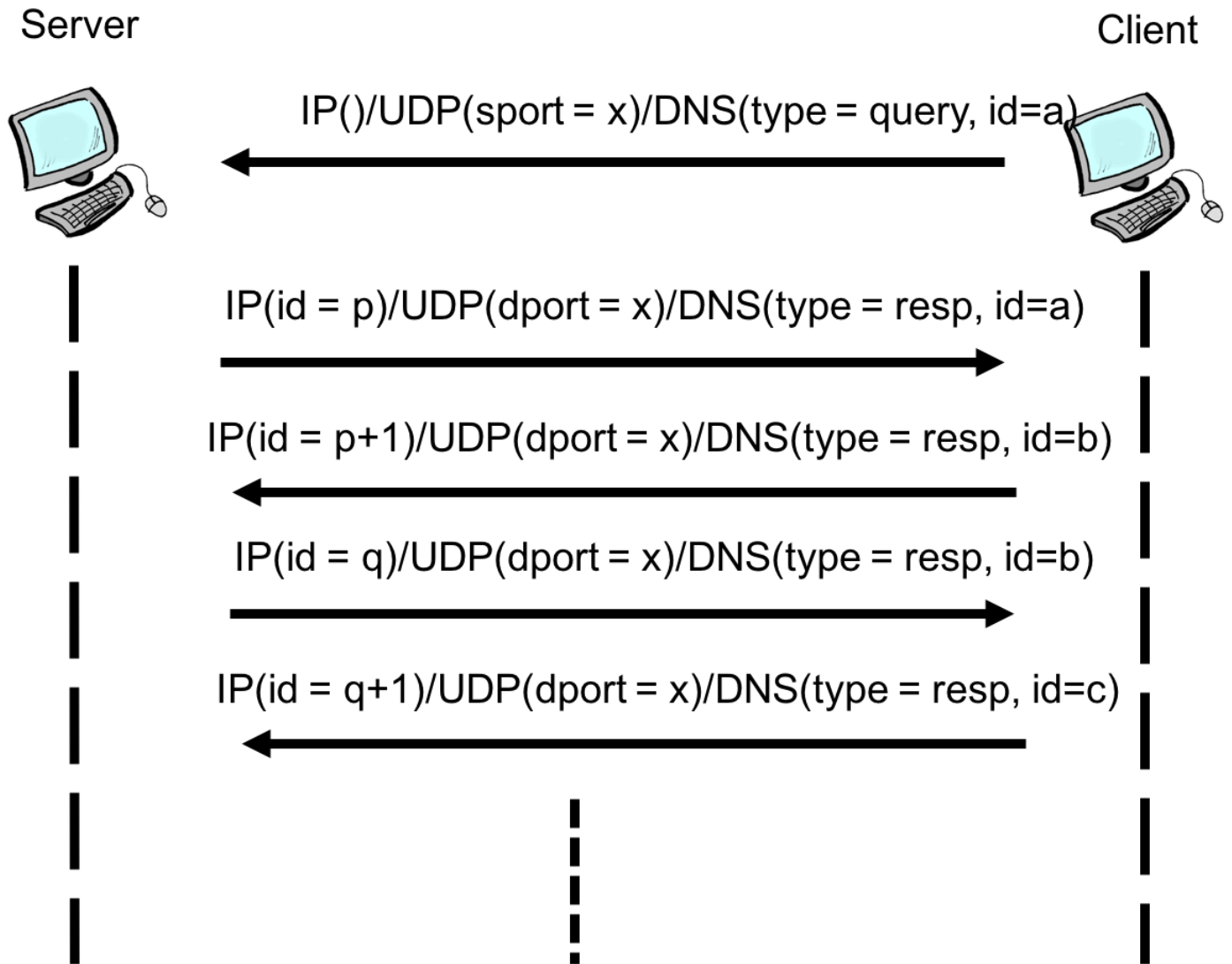
The Client now recognizes the DNS response and extracts the first character.

- The Client acknowledges the receipt of the character by crafting a new DNS query with the lower layer IPv4 id field set to the IPv4 id + 1 value it received from the previous packet.

With the **id + 1** the server now can know for sure the delivery of the packet.

- The process returns until the server sends an **~** character in its flag fields

Block Diagram:



Screenshot of the Exercise:



Wireshark Capture:

No.	Time	Source	Destination	Protocol	Length	Info
2	2...	10.10.111.104	10.10.111.103	DNS	79	Unknown operation (7) 0xab7b A www.oqzyyencmeva.ru
5	7...	10.10.111.103	10.10.111.104	DNS	114	Unknown operation (7) response 0xab7b A www.oqzyyencmeva.ru[Packet siz...
...	1...	10.10.111.104	10.10.111.103	DNS	80	Unknown operation (7) 0x43c5 A www.txmyyphqszed.com
...	1...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (9) response 0x43c5 A www.txmyyphqszed.com[Packet si...
...	2...	10.10.111.104	10.10.111.103	DNS	80	Unknown operation (9) 0xc881 A www.yfbgatsbsbec.com
...	2...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (8) response 0xc881 A www.yfbgatsbsbec.com[Packet si...
...	3...	10.10.111.104	10.10.111.103	DNS	79	Unknown operation (8) 0xa429 A www.fmnsksxsoej.fr
...	3...	10.10.111.103	10.10.111.104	DNS	114	Standard query response 0xa429 A www.fmnsksxsoej.fr[Packet size limit...
...	4...	10.10.111.104	10.10.111.103	DNS	80	Standard query 0xc629 A www.cyshezzgyzuw.com
...	4...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (8) response 0xc629 A www.cyshezzgyzuw.com[Packet si...
...	5...	10.10.111.104	10.10.111.103	DNS	79	Unknown operation (8) 0x9f34 A www.fablgwkmpcr.li
...	5...	10.10.111.103	10.10.111.104	DNS	114	Server status request response 0x9f34 A www.fablgwkmpcr.li[Packet siz...
...	6...	10.10.111.104	10.10.111.103	DNS	80	Server status request 0x3ca7 A www.ciqsqhvxmuyd.com
...	6...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (9) response 0x3ca7 A www.ciqsqhvxmuyd.com[Packet si...
...	7...	10.10.111.104	10.10.111.103	DNS	80	Unknown operation (9) 0x348a A www.qqjsukutnxkg.net
...	7...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (14) response 0x348a A www.qqjsukutnxkg.net[Packet s...
...	8...	10.10.111.104	10.10.111.103	DNS	80	Unknown operation (14) 0x2a8b A www.pyrxaidxunex.com
...	8...	10.10.111.103	10.10.111.104	DNS	116	Unknown operation (13) response 0x2a8b A www.pyrxaidxunex.com[Packet s...

Encryption

Confidentiality, Integrity and Authenticity, are bare minimum for any secure communication. AES-128 symmetric key encryption is used here. With a fairly simple python PyCrypto library the encryption is straight forward. A 16 byte - 128 bit key, and an 128 bit IV are used for AES CBC mode.

The message is encrypted and appended with the ~ character and sent character by character.

Following is the Server Script:

```
from scapy.all import *
import random
import time
from Crypto.Cipher import AES

bin4 = lambda(n): ''.join(str(1 & int(n) >> i ) for i in range(4)[::-1]) #
python < 2.6

# DNS response packet
def constructpkt(char,ip_ident,dns_ident,dport,query):
    RA_c = ord(char) % 2
    RD_c = (ord(char) >> 1) % 2
    TC_c = (ord(char) >> 2) % 2
    AA_c = (ord(char) >> 3) % 2
    op_c = (ord(char) >> 4) % 16
    pkt = IP(src="10.10.111.103", dst="10.10.111.104", id=ip_ident)/UDP(spo
rt=53, dport=dport)
    pkt /= DNS(id=dns_ident, qr=1, opcode=op_c,ra=RA_c,rd=RD_c,tc=TC_c,aa=A
A_c,qd=query,an=DNSRR(rrname=getattr(query,"qname"),rdata=str(RandIP())))
    return pkt

#extracts the character from the packet
def deconstruct(a):
    RA_c = a[0][DNS].ra
    RD_c = a[0][DNS].rd
    TC_c = a[0][DNS].tc
    AA_c = a[0][DNS].aa
    op_c = a[0][DNS].opcode
    bin_s = bin4(op_c) + str(AA_c) + str(TC_c) + str(RD_c) + str(RA_c)
    char = chr(int(bin_s,2))
    return char

## Initiate Message
```

```

lfil = lambda(r): UDP in r and DNS in r and (r[DNS].id == r[UDP].sport + 7)
and (r[DNS].opcode == 7)

a=sniff(count=1,filter="src host 10.10.111.104",lfilter = lfil)

print deconstruct(a)

ip_id = random.randint(1000,60000)

udp_dport = a[0][UDP].sport
prev_query = a[0][DNS].qd
dns_id_prev = a[0][DNS].id

# Message Encryption

encryption_suit = AES.new('netsec favorite',AES.MODE_CBC,'vector8 vector1
0')

message = encryption_suit.encrypt("This is a secret message dhishan")
message += "~";

#sending the message

for c in message:

    ip_id = random.randint(1000,60000)

    pkt = constructpkt(c,ip_id,dns_id_prev,udp_dport,prev_query)

    time.sleep(5)

    flag = 1

    while(flag):

        send(pkt)

        lfila = lambda(r): UDP in r and DNS in r and (r[DNS].id == r[UDP].s
port + 7) and (r[IP].id == ip_id + 1)

        rcv=sniff(count=1,filter="src host 10.10.111.104",lfilter = lfila,t
imeout=10)

        if(len(rcv)!=0):

            flag=0

            udp_dport = rcv[0][UDP].sport

            prev_query = rcv[0][DNS].qd

            dns_id_prev = rcv[0][DNS].id

            print deconstruct(rcv)

```


Following is the Client Script:

```
from scapy.all import *
import string, random
import time
from Crypto.Cipher import AES

top = [ ".com", ".net" , ".com", ".edu" , ".ch", ".de", ".li", ".jp", ".ru",
".tv",".nl",".fr" ]

# converts int into 4 bit bin value
bin4 = lambda(n): ''.join(str(1 & int(n) >> i ) for i in range(4)[::-1]) #
python < 2.6
#{0:b}".format(op_c).zfill(4) #Python > 2.6

def constructpkt(char,ip_ident,dns_ident,sport,query):
    RA_c = ord(char) % 2
    RD_c = (ord(char) >> 1) % 2
    TC_c = (ord(char) >> 2) % 2
    AA_c = (ord(char) >> 3) % 2
    op_c = (ord(char) >> 4) % 16

    pkt = IP(src="10.10.111.104",dst="10.10.111.103",id=ip_ident)/UDP(sport
=sport,dport=53)

    pkt /= DNS(id=dns_ident,qr=0,opcode=op_c,ra=RA_c,rd=RD_c,tc=TC_c,aa=AA_
c,qd=DNSQR(qname=query))

    return pkt

def constructquery(char,ip_id):
    sport = random.randint(10000,60000)
    dns_id = sport + 7
    query = "www."
    query += ''.join(random.choice(string.ascii_lowercase) for x in range(1
2))
    query += top[random.randrange(len(top))]
```

```
return dns_id,constructpkt(char,ip_id,dns_id,sport,query)
```

```
def deconstruct(a):
```

```
    RA_c = a[0][DNS].ra
```

```
    RD_c = a[0][DNS].rd
```

```
    TC_c = a[0][DNS].tc
```

```
    AA_c = a[0][DNS].aa
```

```
    op_c = a[0][DNS].opcode
```

```
    bin_s = bin4(op_c) + str(AA_c) + str(TC_c)+ str(RD_c) + str(RA_c)
```

```
    char = chr(int(bin_s,2))
```

```
    return char
```

```
dns_id , pkt = constructquery('s',random.randint(1,60000))
```

```
send(pkt)
```

```
print getattr(pkt[DNS].qd,"qname")
```

```
c = ""
```

```
flag = 1
```

```
while(flag):
```

```
    lfil = lambda(r): UDP in r and DNS in r and (r[DNS].id == dns_id)
```

```
    rcv=sniff(count=1,filter="src host 10.10.111.103",lfilter = lfil)
```

```
    if(len(rcv)!=0):
```

```
        rchar = deconstruct(rcv)
```

```
        if(rchar == '~'):
```

```
            flag = 0
```

```
        c += rchar
```

```
        dns_id, ack = constructquery(rchar,rcv[0][IP].id +1)
```

```
        time.sleep(5)
```

```
        send(ack)
```

```
c = c[:-1]
```

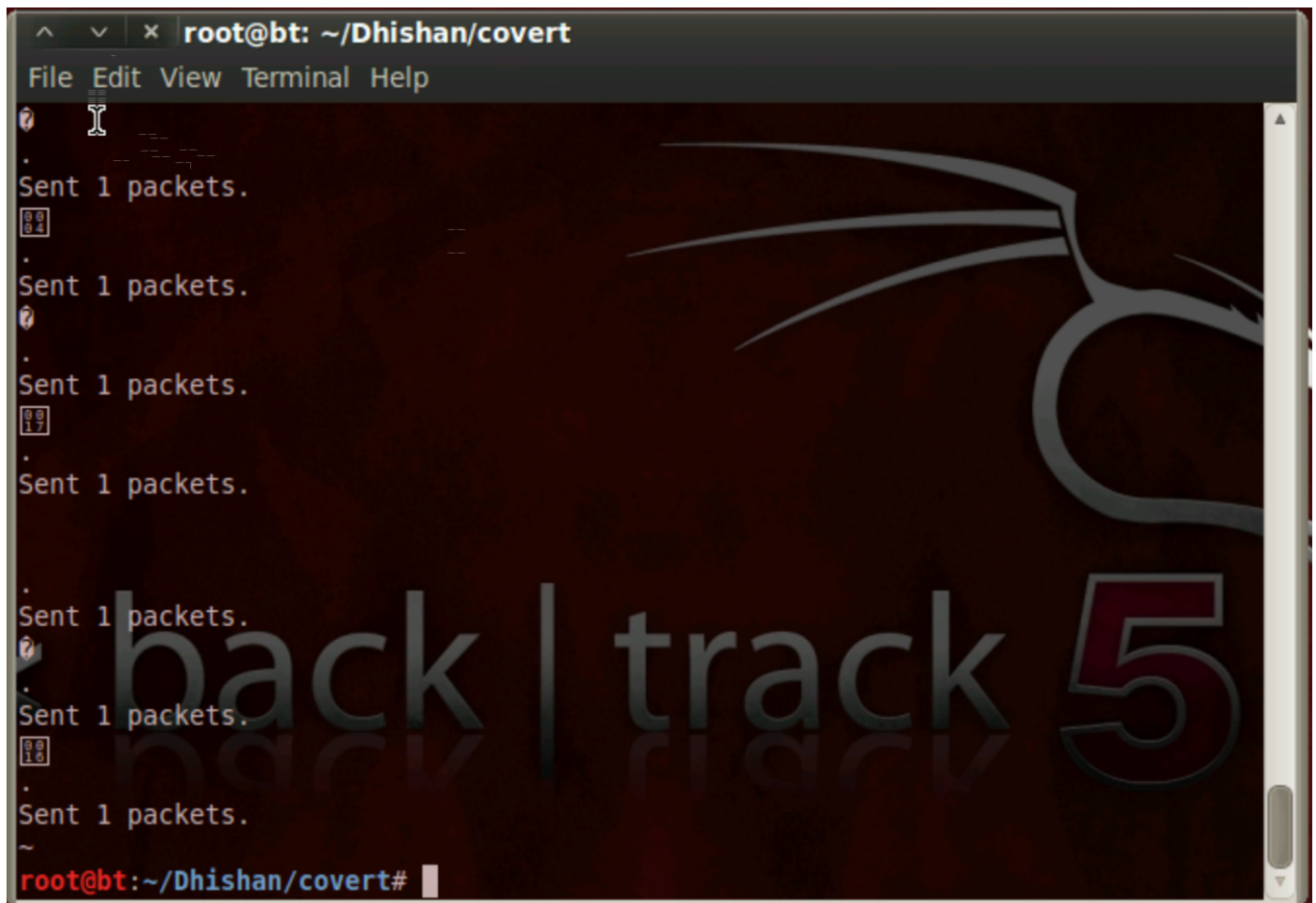
```
print c
```

```
decrypt_suite = AES.new('netsec favorite',AES.MODE_CBC,'vector8 vector10')
```

```
message = decrypt_suite.decrypt(c)
```

```
print message
```

Screenshot of the Server



Screenshot of the Client

