

Covert Communications

Reliable Covert Communications over IP with AES 128 Encryption

Dhishan Amaranath

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

The data (a character) is hidden in the **fragmentation offset** field in the IP header. The Field is 12 Bit long larger than the required 8 Bit for sending an ASCII character. This layer would be a good choice for the covert communication because the field would be of no sense to the receiver unless it is expecting a larger packet with the identification field.

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. Also to separate the covert communication over the other traffic from the same sender, the protocol field in the IP packet can be used to with an undefined protocol number. Here the Protocol number 150 which is not defined according to the [iana](#) is used for distinguishing.

Server Code:

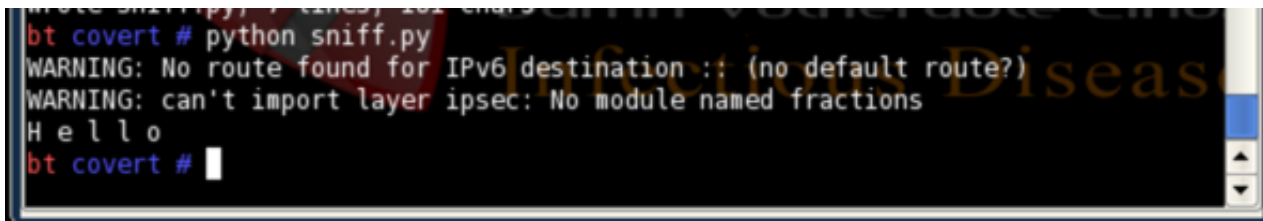
```
from scapy.all import *  
  
for c in "hello":  
    val = hex(ord(c))
```

```
pkt = IP(dst="10.10.111.104",proto=150,frag=val)
send(pkt,inter=0.5)
```

Client Code for Listening:

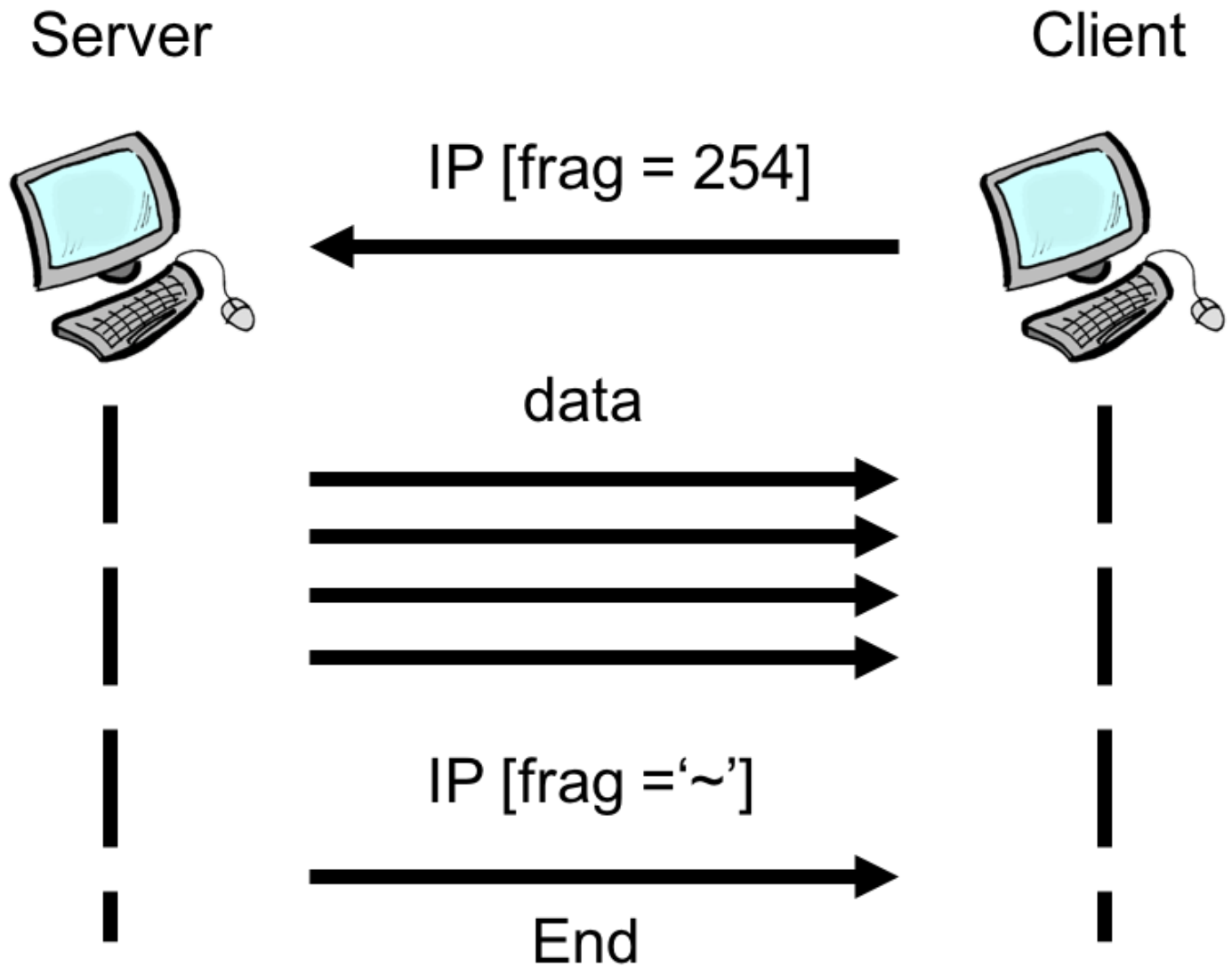
```
from scapy.all import *
lfil = lambda(r): IP in r and r[IP].proto == 150
p = sniff(count =5,filter = "src host 10.10.111.103",lfilter = lfil)
[chr(c[IP].frag) for c in p]
```

Screenshot of the result:



Handshake for initiating and Termination

Block Diagram:

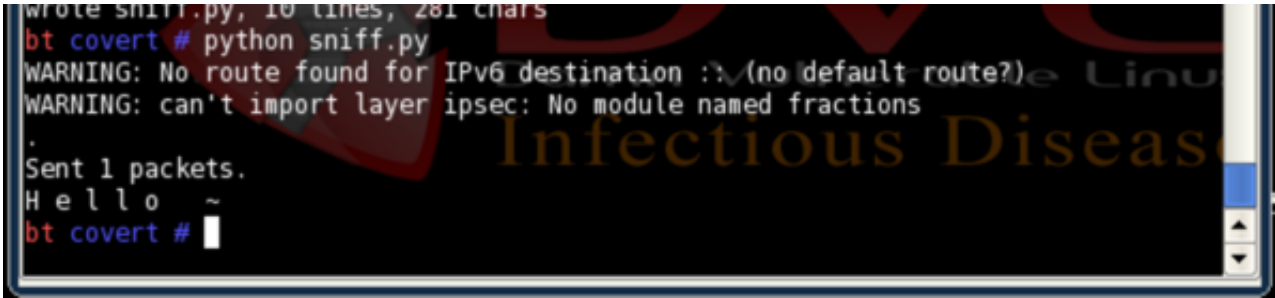


The Client Sends a decimal value of 254 in the fragment offset field informing the server that it is listening for the message. The server sends an '~' character at the end of the message in the fragmentation offset field indicating the end of the communication.

Screenshot of the Result Sent **254**

```
root@bt: ~/Dhishan/covert
Sent 1 packets.
.
Sent 1 packets.
root@bt:~/Dhishan/covert# gedit send.py
root@bt:~/Dhishan/covert# python send.py
WARNING: No route found for IPv6 destination :: (no default route?)
254
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
```

Screenshot of the Result Recieved ~

A screenshot of a terminal window with a black background and white text. The text shows the execution of a Python script named 'sniff.py'. It displays several warning messages about IPv6 and a missing module, followed by the successful sending of a packet containing the word 'Hello'. The prompt 'bt covert #' is visible at the bottom.

```
wrote sniff.py, 10 lines, 281 chars  
bt covert # python sniff.py  
WARNING: No route found for IPv6 destination : (no default route?)  
WARNING: can't import layer ipsec: No module named fractions  
.  
Sent 1 packets.  
H e l l o ~  
bt covert #
```

Reliability

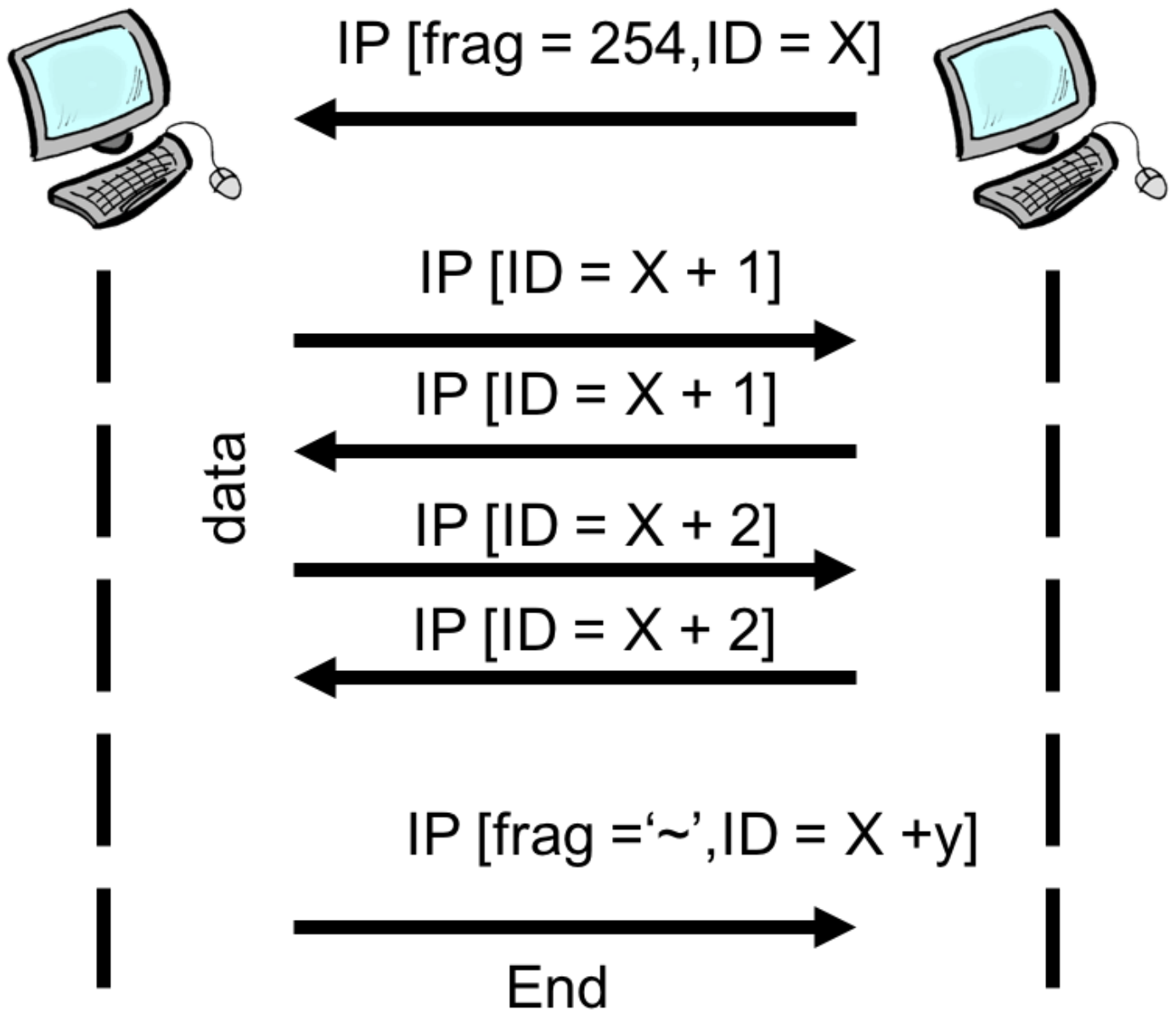
Since the communication is over IP where there is no reliability, A system where data reception is acknowledged is required. The identification field in the IP layer could be used for this purpose.

The Client sends an initial identification number in its initial Handshake and then for each packet the server sends the identification field is incremented by one. The server waits for the acknowledgement of the packet with the same identification value as the previous packet before sending the new packet. The clients waits for 10 seconds for the new packet, if packet is lost, an duplicate acknowledgement for the previous data received is sent indicating the non reception of any further packets.

Working Block Diagram:

Server

Client

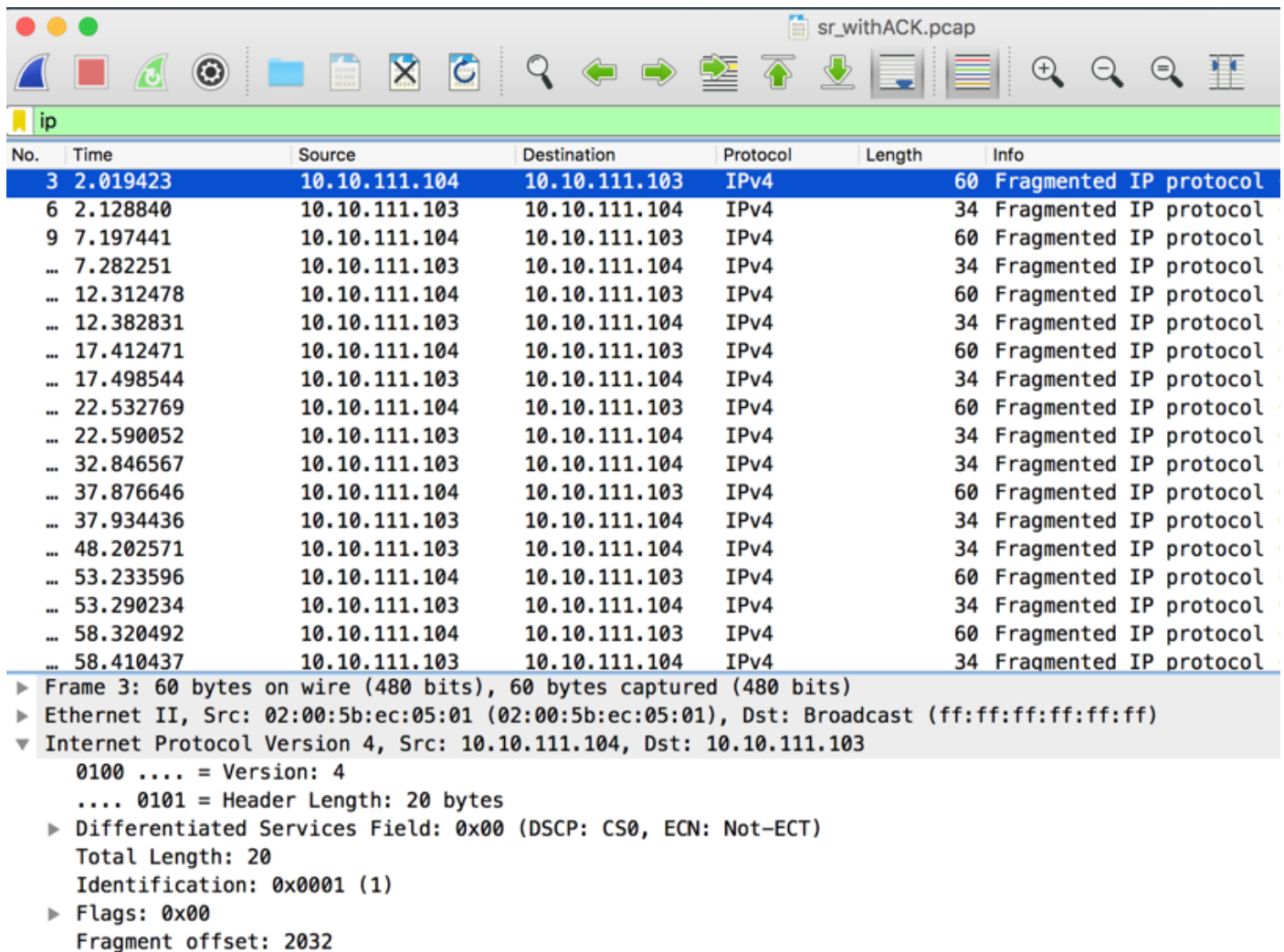


Screenshot of the Exercise:

```
Shell - Konsole
.
Sent 1 packets.
h
.
Sent 1 packets.
i
.
Sent 1 packets.
s
.
Sent 1 packets.
h
.
Sent 1 packets.
a
.
Sent 1 packets.
n
.
Sent 1 packets.
~
.
Sent 1 packets.
This is a secret message dhishan~
bt covert #
```



Wireshark Capture:



No.	Time	Source	Destination	Protocol	Length	Info
3	2.019423	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
6	2.128840	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
9	7.197441	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	7.282251	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	12.312478	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	12.382831	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	17.412471	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	17.498544	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	22.532769	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	22.590052	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	32.846567	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	37.876646	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	37.934436	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	48.202571	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	53.233596	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	53.290234	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol
...	58.320492	10.10.111.104	10.10.111.103	IPv4	60	Fragmented IP protocol
...	58.410437	10.10.111.103	10.10.111.104	IPv4	34	Fragmented IP protocol

▶ Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
 ▶ Ethernet II, Src: 02:00:5b:ec:05:01 (02:00:5b:ec:05:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▼ Internet Protocol Version 4, Src: 10.10.111.104, Dst: 10.10.111.103
 0100 = Version: 4
 0101 = Header Length: 20 bytes
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 20
 Identification: 0x0001 (1)
 ▶ Flags: 0x00
 Fragment offset: 2032

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.

Following is the Server Script:

```

from scapy.all import *

from Crypto.Cipher import AES

lfil = lambda(r): IP in r and r[IP].proto == 150

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

print a[0][IP].frag

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

message = encryption_suit.encrypt("This is a secret message dhishan")
  
```

```

message += "~";
print message
ident = a[0][IP].id
if(ident != 20):
    print "Error!"
for c in message:
    rcv = 1
    val=ord(c)
    pkt = IP(dst="10.10.111.104",id=ident+1,proto=150,frag=val)
    while(rcv):
        send(pkt)
        p = sniff(count=1,filter="src host 10.10.111.104",lfilter = lfil,timeout = 10)
        if(len(p) != 0 and p[0][IP].id == ident+1):
            ident +=1
            rcv = 0

```

Following is the Client Script:

```

from scapy.all import *
from Crypto.Cipher import AES
import time
ident = 20
send(IP(dst="10.10.111.103",id=ident,proto=150,frag=0xFE))
lfil = lambda(r): IP in r and r[IP].proto == 150
flag = 1
str=""
char1=0
while(flag):
    p = sniff(count=1,filter = "src host 10.10.111.103",lfilter = lfil)
    if(p[0][IP].id == ident + 1):
        ident +=1
        str +=chr(p[0][IP].frag)
        time.sleep(5)
    send(IP(dst="10.10.111.103",id=ident,proto=150,frag=p[0][IP].frag))

```



```

if p[0][IP].frag == 126:

    flag = 0

str = str[:-1]

decrypt_suite = AES.new('netsec favorite',AES.MODE_CBC,'vector8 vector10')
message = decrypt_suite.decrypt(str)

print message

```

Screenshot of the Experiment

