Project 3: Network Attacks & Defenses

1. Nmap Port Scanning

Idea: Want to probe which ports are open on our target machine (victim server). Ran the nmap command below on the target machine `scanme.nmap.org`. The scan took 11 minutes over throttled stanford network. The output is shown in Figure 1.

```
sudo nmap -sS -A -T4 -p0-65535 scanme.nmap.org
```

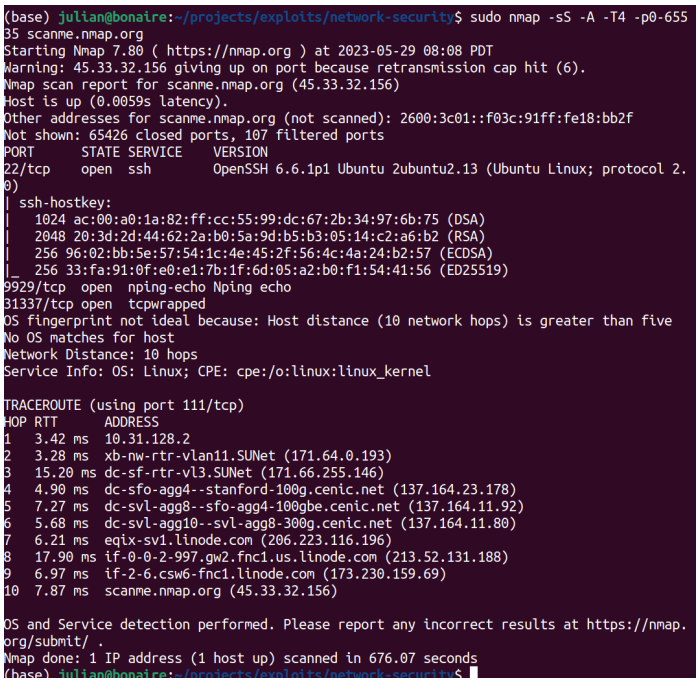

Figure 1: nmap console output

Working notes:

- Flags used: sS = TCP SYN scan; -A = OS detection (-O), version detetction (-sV), script scanning (-sC), traceroute (–traceroute); T4 = quick scan; -p0-65535 = expand scan from top 1000 to all ports
- Relevant websites: `https://nmap.org/book/man-version-detection.html` and `https://nmap.org/book/man-port-scanning-techniques.html`

2. Wireshark Packet Sniffing

Idea: Wireshark is a tool to monitor and record local network traffic (packet sniffing). We start recording packets, then run the nmap command from part 1 and stop recording

after nmap completes its scan. Export the wireshark output as a .pcap file and open it in wireshark to analyze.

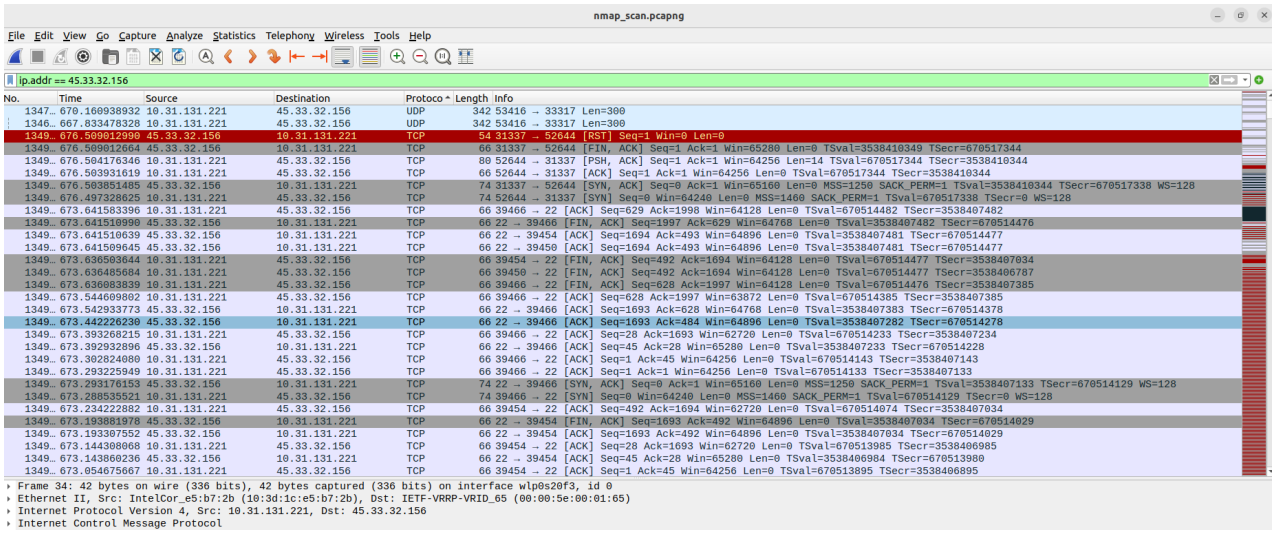

Figure 2: wireshark gui output

Working notes:

- A port is ‘closed‘ if no application or service from target machine is listening on that port. We can identify closed ports on wireshark when the target machine responds immediately with an RST,ACK packet, or NIL if not packet is sent.
- A port is ‘filtered‘ if your probe did not reach the destination port because it was dropped by the firewall. We can identify filtered ports on wireshark when the target machine responds with no response packet. This is sometimes followed by an ICMP communication to confirm that the port is unreachable.
- To find which types of http requests were made by nmap, we simply filter on HTTP for protocol field. There are HTTP GET, HTTP OPTIONS, HTTP POST, and HTTP PROPFIND requests made during the scan.
- To check TCP parameters changed by nmap, scroll through wireshark output and inspect TCP packets. We find that only Destination Port, [Stream Index], Sequence Number (raw), Checksum are changed.

3. Programmatic Packet Processing

Idea: Programmatically analyze a PCAP (Packet Capture) file to detect Port Scanning and ARP Spoofing. Port Scanning is when an attacker uses `nmap` (or similar) to find all open ports on a known host. ARP Spoofing is when an attacker sends ARP packets to a target machine to redirect traffic to a different machine.
Code to parse ip and tcp layer packets.

```
func parseIpTcp(addresses *map[string][2]int,
                ipLayer gopacket.Layer,
                tcpLayer gopacket.Layer) {

    // Get IP packet...
    ipPacket, _ := ipLayer.(*layers.IPv4)

    // ... and TCP packet...
    tcpPacket, _:= tcpLayer.(*layers.TCP)

    // ... then update sent-syn count
    if tcpPacket.SYN && !tcpPacket.ACK {
        srcIpString := net.IP(ipPacket.SrcIP).String()
        counts := (*addresses)[srcIpString]
        counts[0]++
        (*addresses)[srcIpString] = counts
    }  else if tcpPacket.SYN && tcpPacket.ACK {
        //  ...and update received-synack count
        dstIpString := net.IP(ipPacket.DstIP).String()

        counts := (*addresses)[dstIpString]
        counts[1]++
        (*addresses)[dstIpString] = counts
    }
}
```

Code to parse arp layer packets.

```
func parseArp(arpRequests *map[string]map[string]int,
              arpMac *map[string]int,
              arpLayer gopacket.Layer) {

    arp, _ := arpLayer.(*layers.ARP)

    // Parse arp to get additional info
    if arp.Operation == 1 { // ARP request
        // IP address that ARP is looking for
        arpReqIp := net.IP(arp.DstProtAddress).String()
        arpReqSrcMac := net.HardwareAddr(arp.SourceHwAddress).String()

        // add new entry to arpRequests map if first arpRequest for this IP address
        _, exists := (*arpRequests)[arpReqIp]
        if !exists {
            (*arpRequests)[arpReqIp] = make(map[string]int)
            (*arpRequests)[arpReqIp][arpReqSrcMac] = 1
        } else { // if val, ok := arpReqEntry[arpReqSrcMac]; ok
```

```
            // increment num requests for IP from this MAC address
            currVal := (*arpRequests)[arpReqIp][arpReqSrcMac]
            currVal++
            (*arpRequests)[arpReqIp][arpReqSrcMac] = currVal
        }

    } else if arp.Operation == 2 { // ARP reply
        ArpSrcMac := net.HardwareAddr(arp.DstHwAddress).String()

        IPRequested := net.IP(arp.SourceProtAddress).String()

        // if arpRequest exists in requests
        value, exists := (*arpRequests)[IPRequested]
        if exists && (*arpRequests)[IPRequested][ArpSrcMac] > 0 {
            // decrement num open reqs for tuple corresponding to
            // (IP addr it's looking for & MAC that sent it)
            currNumOpenReqs := value[ArpSrcMac]
            currNumOpenReqs--
            (*arpRequests)[IPRequested][ArpSrcMac] = currNumOpenReqs
        } else { // unsollicited reply
            // increment num offenses by given MAC address
            srcMacAddress := net.HardwareAddr(arp.SourceHwAddress).String()
            currNumOffenses := (*arpMac)[srcMacAddress]
            currNumOffenses++
            (*arpMac)[srcMacAddress] = currNumOffenses
        }
    }
}
```

Working notes:

- Port Scanning: Record if an ip address makes 3 times as many SYN packets as the number of SYN+ACK packets they received and also sent more than 5 SYN packets in total. Populate `addresses` map.
- ARP Spoofing: Unsolicited ARP replies are those which contain a source IP and destination MAC address combination that does not correspond to a previous request (in other words, each request should correspond to at most one reply, and any extra replies are unsolicited).

4. Monster-In-The-Middle

Idea: xx

```
code goes here
```

Working notes:

- xx
- xx