CS 426 Project 2 Answers

Name: Jack Bauer

Purdue ID: bauer88

NOTE: I attempted to use the Wireshark GUI or tshark to complete all tasks, but I felt separate scripts were needed to keep running lists of ISNs and time to live for questions 7 and 8 respectively. I included these scripts as separate python files. All tshark scripts were run on Windows 10 in Powershell.

1. Answer:
107.209.122.239
107.209.140.237
169.52.145.47
171.120.141.223
173.122.147.191
173.122.215.203
173.87.213.190
173.88.187.207
175.20.219.170
175.20.219.205
183.157.123.171
233.144.182.73
233.176.212.206
233.184.242.174
233.242.180.94
235.16.181.171
235.16.181.234
235.184.209.171
237.210.148.238
237.240.108.203
237.240.108.74
237.240.108.94
41.122.223.232
43.186.190.205
43.190.250.201
43.190.254.93
45.178.180.62
45.90.187.168
45.90.187.200
45.90.187.201
55.210.154.170
55.210.154.206
57.82.179.109
59.180.185.173

    59.210.150.94
    59.244.240.47

   a. Script used:
      ```
      .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y http.response -T fields -e ip.src | sort -u
      ```

   b. For this problem, I filtered all packets to include only http response packets. An http response indicates that the web server was successfully visited, regardless of the status code. By including the "-e ip.src" field and the "sort -u" command, I was able to print out all of the unique IP addresses of web servers that sent a response, i.e., the web servers that were successfully visited.

2. Answer: 43.126.249.174
   a. Script used:
      ```
      .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'http.request.uri contains \"../../../\"' -T fields -e ip.src | sort -u
      ```

   b. For this problem, I used the contains filter to filter all http request packets that contained the string "../../../". This indicates that the user sending the request is intending to traverse the directory of the server in order to locate specific file(s) they are not intended to see. I then used the "-e ip.src" field and the "sort -u" command to print out all unique source IP addresses that contained the given string resulting in the IP address 43.126.249.174.
3. Answer: Attacker IP: 233.24.213.78, host IP: 237.240.108.94
   a. Script used:
      ```
      .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'ftp.response.code==530' -T fields -e ip.dst -e ip.src | sort -u
      ```

   b. For this problem, I first used the filter "ftp" in the Wireshark GUI to isolate all ftp packets. I noticed that in the response packets, there was always a response code, similar to http. I did some research and found that the code 530 indicates a failed authentication, meaning that an incorrect password was used (source: https://shockbyte.com/). Therefore, I used the above script to filter all packets that included a response code of 530. I then printed out the unique destination IP addresses that this code was being sent to, given by the "-e ip.dst" field and the "sort -u" command. I also printed out the source IP address that was sending this code, given by the "-e ip.src" field, just to be safe because I wasn't sure if needed that to be in our answer. Since the response code 530 was being sent to the printed-out IP addresses, this indicates that the user at that IP address is the attacker attempting to guess passwords.
4. Answer: Username: calrules, Password: thisissosecure
   a. Filter used: "telnet"

   b. For this problem, I used the Wireshark GUI to filter all telnet packets that were sent by simply using the filter "telnet". After filtering all telnet packets, I used the "follow TCP stream" tool to see all of the data sent through the telnet packets together. This resulted in seeing the username: calrules and password: thisissosecure in plaintext. Before using the "telnet" filter, I did the same method described above with FTP, HTTP, and POP3 (although there were no POP3 packets in this trace), but the TCP streams of these kinds of packets did not show any usernames or passwords.

5. Answer: 43.190.254.93
    a. Scripts used:
    ```
    .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'http.server contains
    \"Apache/\"' -T fields -e http.server | sort -u

    .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'http.server contains
    \"Apache/1.3.28\"' -T fields -e ip.src | sort -u
    ```
    b. For this question, I used two separate tshark scripts to find my solution. First, I used the first script listed above to find all the versions of Apache that were listed in the http packets, as given by the field "-e http.server", and sorted them. I found that Apache/1.3.28 was the oldest version out of all the packets. I then used the second script listed above to find all unique source IP addresses of the server(s) running Apache/1.3.28, given by the field "-e ip.src" and the "sort -u" command. This resulted in printing out the IP 43.190.254.93.
6. Answer: 43.190.254.93, 43.190.254.94
    a. Script used:
    ```
    .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'dns.flags==0x0100' -T fields -e
    dns -e ip.src -e udp.port | sort -u
    ```
    b. For this question, by inspecting a DNS request query and a DNS response query in the Wireshark GUI, I found that all DNS request queries have the flag 0x0100. I used that in my script to isolate all DNS request queries and print their respective unique source IP address/UDP port combinations given by the "-e ip.src -e udp.port" fields and the "sort -u" command. I then located the IP addresses in that list that only appeared once (meaning they only used one UDP port for all packets). This resulted in the IP addresses 43.190.254.93, and 43.190.254.93.
7. Answer: 233.16.159.221, 237.240.108.203
    a. Script used:
    question_7.py
    b. For this problem, I first used the filter "tcp.flags.syn == 1 and tcp.flags.ack == 0" in the Wireshark GUI to filter all packets that contained only SYN flags. SYN flags are necessary for this problem because sending a SYN flag is the first step for a TCP 3-way handshake. A SYN/ACK packet is then sent back, but to obtain the ISN, we need to look at the packets containing only a SYN flag. I originally attempted to do this problem using tshark. However, I found that it is not scalable as there are a very large number of packets and I would have to do a lot of math by hand. Therefore, I wrote a short python script that contained the same filter shown above and utilized the pyshark library to iterate over the filtered packets. I keep a running list of all (source IP, destination IP)

tuples and their respective ISNs using a python dictionary with the tuple as the key and a list of ISNs as the value. By keeping a running list of ISNs, I am able to iterate over those values and find the endpoints with the broadest range by computing the maximum ISN minus the minimum ISN. I then print the tuple that satisfies this condition resulting in the endpoints 233.16.159.221 and 237.240.108.203.

8. Answer:
    a. Script used: Host: 233.18.121.211, Destination: 237.240.112.42
    question_8.py
    b. For this problem, I first used the filter "udp and not dns and not mdns and not browser" in the Wireshark GUI. This filter will isolate all of the udp packets but will not include any UDP packets that are involved in the application layer. The protocols dns, browser, and mdns all utilize udp for transport, but those are not the packets we are interested in. As in question 7, I originally tried to do this problem with tshark. However, it would not be scalable as I would have to do a lot of inspection into a large number of packets using Wireshark. Therefore, I wrote a short python script that contained the same filter as shown above and utilized the pyshark library to iterate over the filtered packets. In this script, I used a very similar approach as I did with question 7. I kept a running list of all (source IP, destination IP) tuples and their respective time to live using a python dictionary with the tuple as the key and a list of times to live as the value. By keeping a running list of times to live, I am able to iterate over those values and find the two endpoints that have an increasing time to live. I do this by storing all the tuples' times to live in the dictionary and finding the key, value pair that has the greatest length. The tuple with the longest list of times to live would be the indicate that the source is the host running traceroute since the time to live continues to increase. Finding the tuple with the longest list of times to live resulted in the source being 233.18.121.211 and the destination being 237.240.112.42.
9. Answer: 237.240.108.203
    a. Script used:
    ```
    .\tshark -r C:\Users\jnbau\Downloads\project2.pcap -Y 'http.request.uri contains \"<script>\" and http.request.uri contains \"</script>\"' -T fields -e ip.dst | sort -u
    ```
    b. For this question, I looked for all http requests that included a <script>…</script> field somewhere in the request. Since the attacker includes a script in the URL, whatever source included <script>…</script>, the contains filter allowed me to find any request that had those strings in it. The script then prints out the unique destination IP addresses for all of the matching requests given by the "-e ip.dst" field and the "sort -u" command resulting in the IP address 237.240.108.203. For this project specifically, it is sufficient to only include "contains <script>", but I figured that this may not always be the case, so I included "contains </script>" as well for scalability.