Lucas McMahon
7/30/24

Detecting ICMP and DNS Anomalies using Wireshark

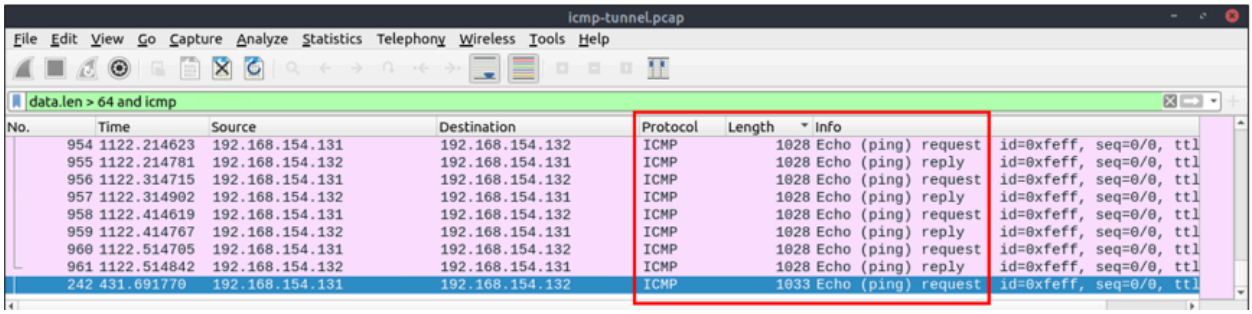
Traffic tunneling is the process of encrypting and controlling traffic traveling from a public to a private network (or vice versa). Traffic tunneling provides traffic security so it is very commonly used in enterprise networks on large and small scales. However, it also provides data anonymity, which attackers can use to fit malicious traffic between legitimate traffic, often exploiting trusted protocols like ICMP and DNS.

The goal of this exercise is to provide the reader with an understanding of how wireshark can be used to sniff data and find malicious/abnormal traffic, as well as enhance my own learning. I will be using TryHackMe's learning module, "Wireshark: Traffic Analysis' ', to guide the learning through notes and questions.

ICMP Analysis

ICMP is used to send diagnostic messages for network issues, and is a part of routine network traffic. Since it is a trusted protocol, it can be used for Denial-of-Service attacks, and Command-and-Control data exfiltration. An adversary will attach http, tcp or ssh data to an ICMP packet. A large volume of ICMP traffic will indicate malicious ICMP tunneling. Below are filters used to navigate ICMP traffic on Wireshark.

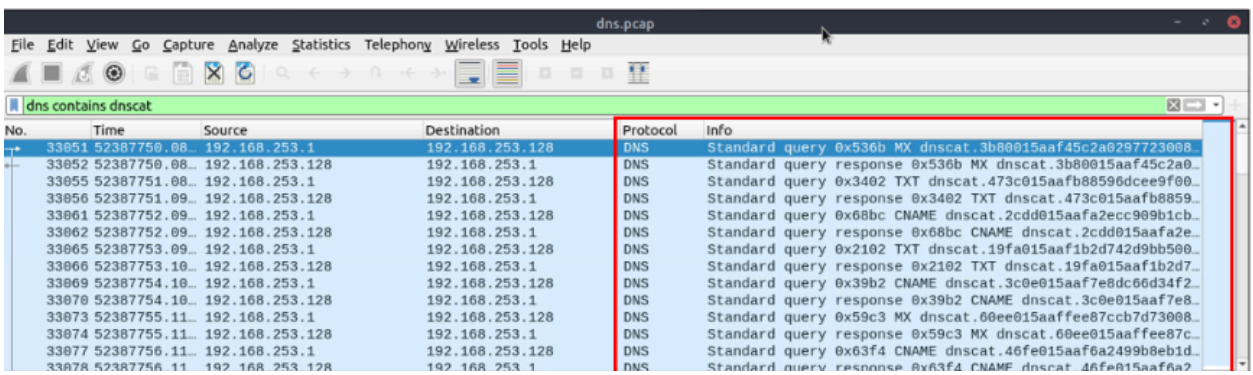
Notes	Wireshark filters
Global search	• `icmp`
"ICMP" options for grabbing the low-hanging fruits: • Packet length. • ICMP destination addresses. • Encapsulated protocol signs in ICMP payload.	• `data.len > 64 and icmp`



DNS Analysis

Domain Name System is used to convert IP hostnames to IP addresses. As with ICMP, it is an essential part of internet traffic, it is often overlooked. It can commonly be used to mask Command-and-Control attacks and Denial-of-Service attacks. Below are filters used on Wireshark to navigate DNS traffic.

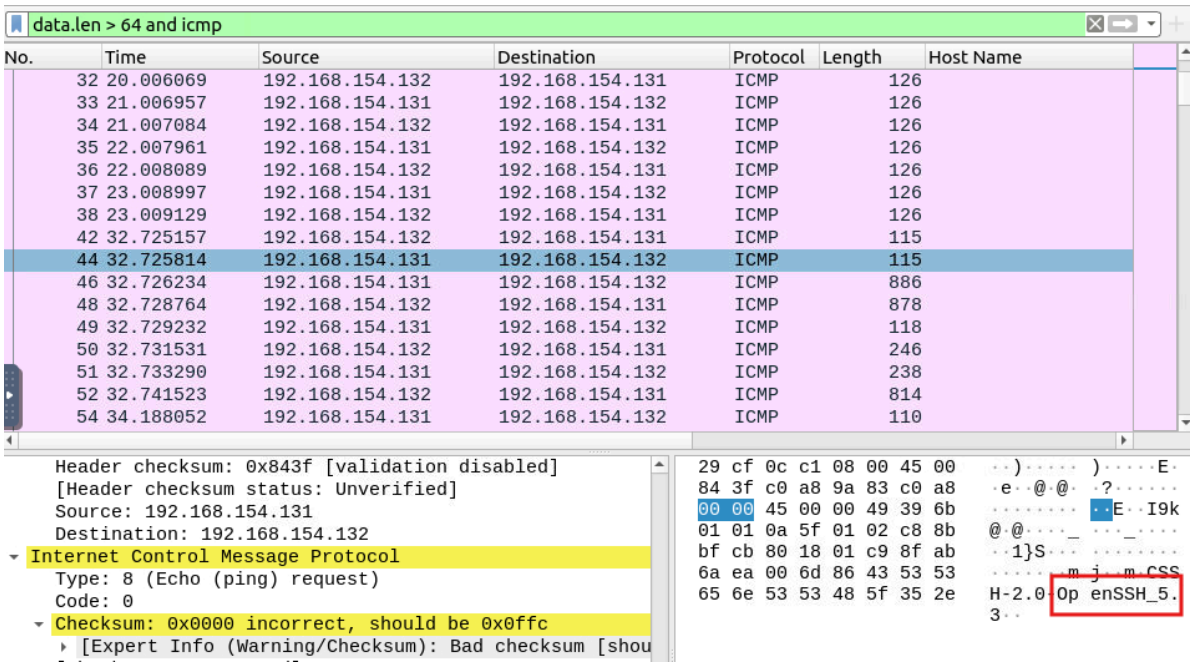
Notes	Wireshark Filter
Global search	• `dns`
"DNS" options for grabbing the low-hanging fruits: • Query length. • Anomalous and non-regular names in DNS addresses. • Long DNS addresses with encoded subdomain addresses. • Known patterns like dnscat and dns2tcp. • Statistical analysis like the anomalous volume of DNS requests for a particular target. **!mdns:** Disable local link device queries.	• `dns contains "dnscat"` • `dns.qry.name.len > 15 and !mdns`



Investigation

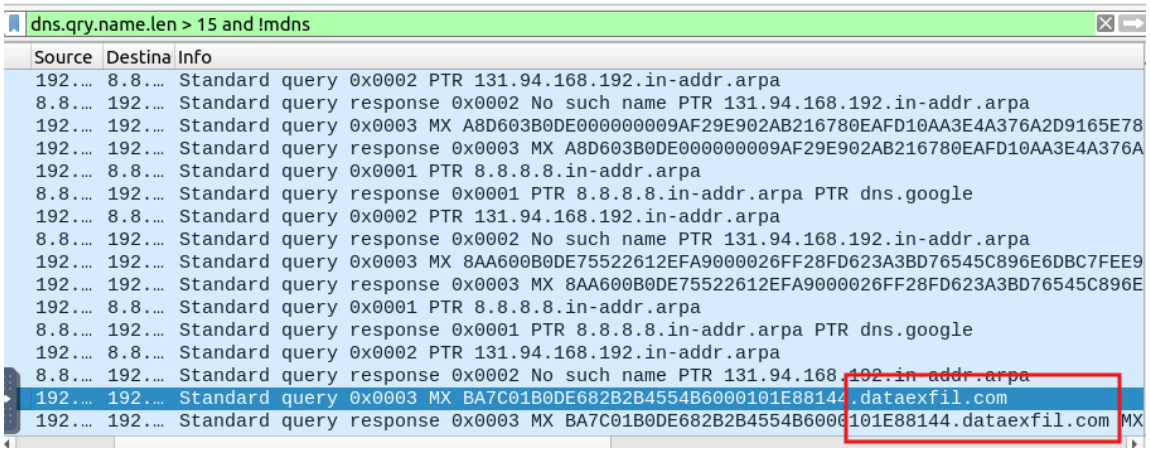
My first step in my investigation is to filter through ICMP traffic, and find out what protocol is being exploited and embedded into malicious traffic. My initial thought is how can we start out by filtering through normal ICMP traffic? As stated earlier, ICMP is a very common protocol and in this scenario, we are given a file that contains several thousand ICMP packets. We could manually go through each of these and investigate the data, but it would be a needle in a haystack.

ICMP packets are usually very small in size, no more than a couple of bytes. Using the filter given earlier, "*data.len > 64 and ICMP*" to display any ICMP traffic that might be malicious from large packet sizes, we can search through the contents of the packets. Wireshark gives you the ability to display the raw data of a packet, which can be a very powerful tool used in an investigation. After opening up one of the first few packets, ssh protocols are found hidden within the data.



Switching over to investigate the malicious DNS traffic,I am given a DNS traffic file with several thousand DNS packets. As the same with ICMP, DNS packets, specifically DNS queries, come in small packet sizes. I can use the filter given earlier "*dns.qry.name.len >15*" to sort through the abnormally large data packets.

The question asked is to find what website the attackers are sending malicious DNS traffic to. In this scenario, the attackers did not take much effort to conceal their hacking other than DNS masking, and we can find many instances of large DNS query packets being sent to "dataexfil[.]com", which is exactly what I was looking for, and will conclude this exercise.



dns.qry.name.len > 15 and !mdns

Source	Destina	Info
192.…	8.8.…	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.…	192.…	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.…	192.…	Standard query 0x0003 MX A8D603B0DE000000009AF29E902AB216780EAFD10AA3E4A376A2D9165E78
192.…	192.…	Standard query response 0x0003 MX A8D603B0DE000000009AF29E902AB216780EAFD10AA3E4A376A
192.…	8.8.…	Standard query 0x0001 PTR 8.8.8.8.in-addr.arpa
8.8.…	192.…	Standard query response 0x0001 PTR 8.8.8.8.in-addr.arpa PTR dns.google
192.…	8.8.…	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.…	192.…	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.…	192.…	Standard query 0x0003 MX 8AA600B0DE75522612EFA9000026FF28FD623A3BD76545C896E6DBC7FEE9
192.…	192.…	Standard query response 0x0003 MX 8AA600B0DE75522612EFA9000026FF28FD623A3BD76545C896E
192.…	8.8.…	Standard query 0x0001 PTR 8.8.8.8.in-addr.arpa
8.8.…	192.…	Standard query response 0x0001 PTR 8.8.8.8.in-addr.arpa PTR dns.google
192.…	8.8.…	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.…	192.…	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.…	192.…	Standard query 0x0003 MX BA7C01B0DE682B2B4554B6000101E88144.dataexfil.com
192.…	192.…	Standard query response 0x0003 MX BA7C01B0DE682B2B4554B6000101E88144.dataexfil.com MX