

Detecting ICMP and DNS Anomalies using Wireshark

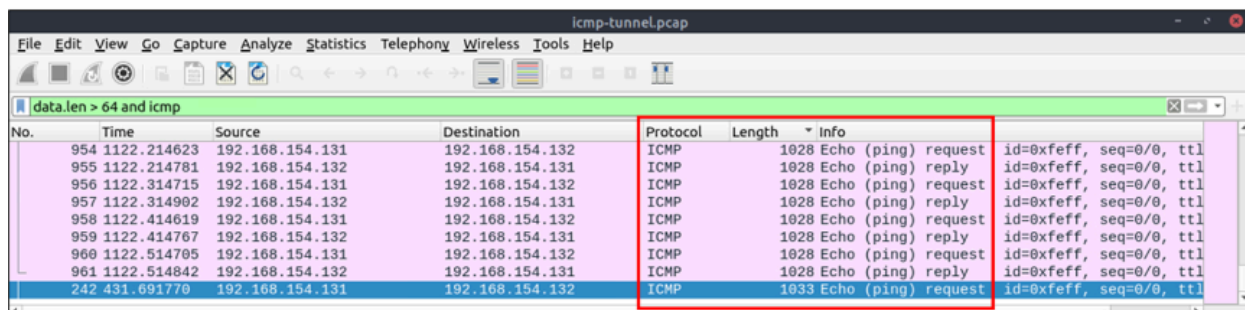
Traffic tunneling is the process of encrypting and controlling traffic traveling from a public to a private network (or vice versa). Traffic tunneling provides traffic security so it is very commonly used in enterprise networks on large and small scales. However, it also provides data anonymity, which 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	<ul style="list-style-type: none"><code>icmp</code>
"ICMP" options for grabbing the low-hanging fruits: <ul style="list-style-type: none">Packet length.ICMP destination addresses.Encapsulated protocol signs in ICMP payload.	<ul style="list-style-type: none"><code>data.len > 64 and icmp</code>

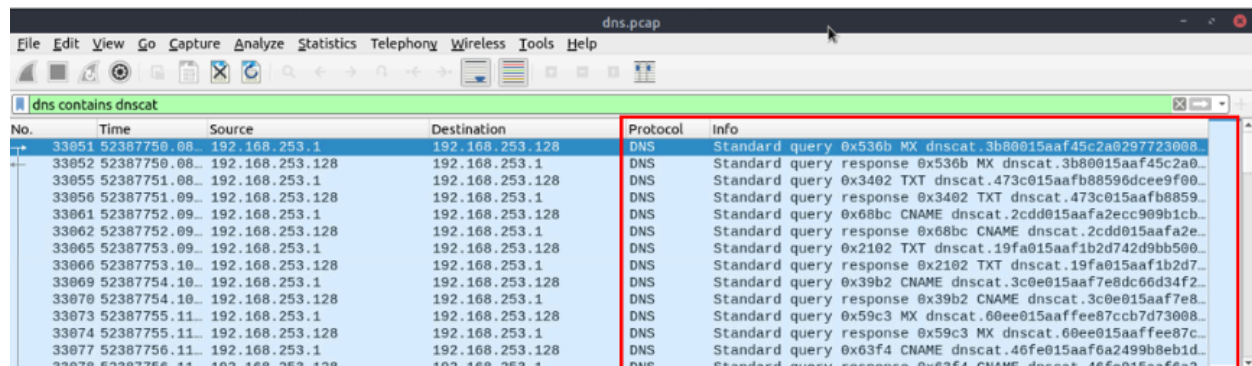


No.	Time	Source	Destination	Protocol	Length	Info
954	1122.214623	192.168.154.131	192.168.154.132	ICMP	1028	Echo (ping) request id=0xfeff, seq=0/0, ttl=64
955	1122.214781	192.168.154.132	192.168.154.131	ICMP	1028	Echo (ping) reply id=0xfeff, seq=0/0, ttl=64
956	1122.314715	192.168.154.131	192.168.154.132	ICMP	1028	Echo (ping) request id=0xfeff, seq=0/0, ttl=64
957	1122.314902	192.168.154.132	192.168.154.131	ICMP	1028	Echo (ping) reply id=0xfeff, seq=0/0, ttl=64
958	1122.414619	192.168.154.131	192.168.154.132	ICMP	1028	Echo (ping) request id=0xfeff, seq=0/0, ttl=64
959	1122.414767	192.168.154.132	192.168.154.131	ICMP	1028	Echo (ping) reply id=0xfeff, seq=0/0, ttl=64
960	1122.514705	192.168.154.131	192.168.154.132	ICMP	1028	Echo (ping) request id=0xfeff, seq=0/0, ttl=64
961	1122.514842	192.168.154.132	192.168.154.131	ICMP	1028	Echo (ping) reply id=0xfeff, seq=0/0, ttl=64
242	431.691770	192.168.154.131	192.168.154.132	ICMP	1033	Echo (ping) request id=0xfeff, seq=0/0, ttl=64

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	<ul style="list-style-type: none">• dns
"DNS" options for grabbing the low-hanging fruits: <ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• dns contains "dnscat"• dns.qry.name.len > 15 and !mdns



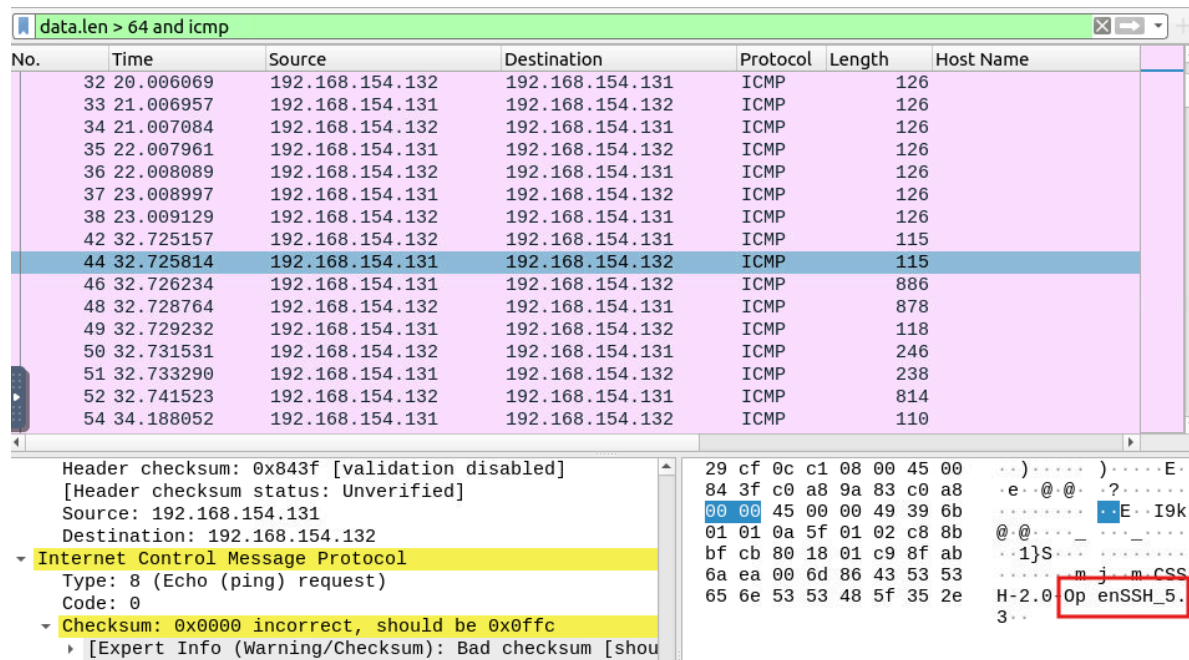
The image shows a Wireshark packet capture window titled 'dns.pcap'. The filter bar at the top displays 'dns contains dnscat'. The packet list table below shows 18 captured packets, all of which are DNS queries or responses related to 'dnscat'. The selected packet (No. 33051) is a standard query for 'dnscat.3b80015aaf45c2a0297723008'.

No.	Time	Source	Destination	Protocol	Info
33051	52387750.00...	192.168.253.1	192.168.253.128	DNS	Standard query 0x536b MX dnscat.3b80015aaf45c2a0297723008
33052	52387750.08...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x536b MX dnscat.3b80015aaf45c2a0...
33055	52387751.08...	192.168.253.1	192.168.253.128	DNS	Standard query 0x3402 TXT dnscat.473c015aafb88596dcee9f00...
33056	52387751.09...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x3402 TXT dnscat.473c015aafb8859...
33061	52387752.09...	192.168.253.1	192.168.253.128	DNS	Standard query 0x60bc CNAME dnscat.2cdd015aafa2ecc909b1cb...
33062	52387752.09...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x60bc CNAME dnscat.2cdd015aafa2e...
33065	52387753.09...	192.168.253.1	192.168.253.128	DNS	Standard query 0x2102 TXT dnscat.19fa015aaf1b2d742d9bb500...
33066	52387753.10...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x2102 TXT dnscat.19fa015aaf1b2d7...
33069	52387754.10...	192.168.253.1	192.168.253.128	DNS	Standard query 0x39b2 CNAME dnscat.3c0e015aaf7e8dc66d34f2...
33070	52387754.10...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x39b2 CNAME dnscat.3c0e015aaf7e8...
33073	52387755.11...	192.168.253.1	192.168.253.128	DNS	Standard query 0x59c3 MX dnscat.60ee015aaffee87ccb7d73008...
33074	52387755.11...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x59c3 MX dnscat.60ee015aaffee87c...
33077	52387756.11...	192.168.253.1	192.168.253.128	DNS	Standard query 0x63f4 CNAME dnscat.46fe015aaf6a2499b8eb1d...
33078	52387756.11...	192.168.253.128	192.168.253.1	DNS	Standard query response 0x63f4 CNAME dnscat.46fe015aaf6a2...

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.



Wireshark packet capture showing ICMP traffic filtered by "data.len > 64 and icmp". The packet list shows several ICMP Echo requests. Packet 44 is selected, showing an ICMP Echo request with a large data payload. The packet details pane shows the ICMP header and the raw data payload, which contains the string "OpenSSH_5.3".

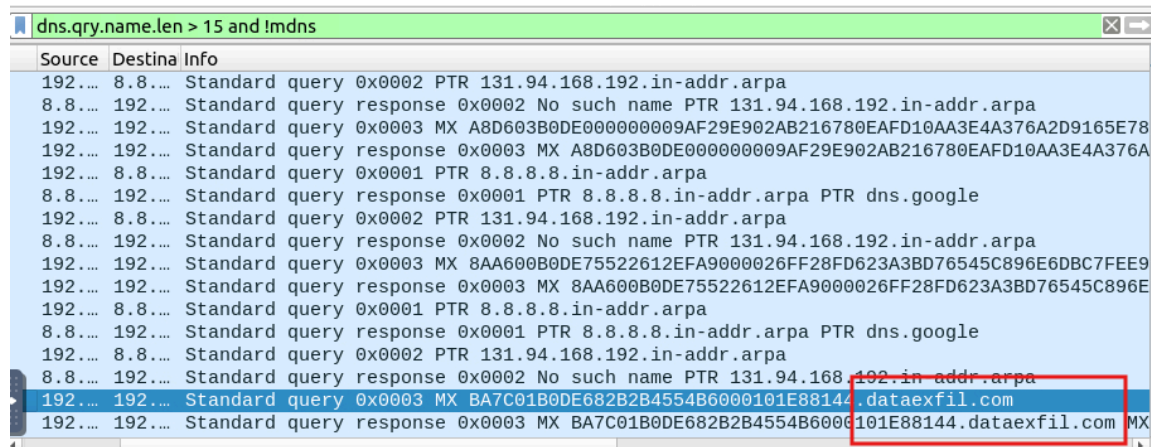
No.	Time	Source	Destination	Protocol	Length	Host Name
32	20.006069	192.168.154.132	192.168.154.131	ICMP	126	
33	21.006957	192.168.154.131	192.168.154.132	ICMP	126	
34	21.007084	192.168.154.132	192.168.154.131	ICMP	126	
35	22.007961	192.168.154.131	192.168.154.132	ICMP	126	
36	22.008089	192.168.154.132	192.168.154.131	ICMP	126	
37	23.008997	192.168.154.131	192.168.154.132	ICMP	126	
38	23.009129	192.168.154.132	192.168.154.131	ICMP	126	
42	32.725157	192.168.154.132	192.168.154.131	ICMP	115	
44	32.725814	192.168.154.131	192.168.154.132	ICMP	115	
46	32.726234	192.168.154.131	192.168.154.132	ICMP	886	
48	32.728764	192.168.154.132	192.168.154.131	ICMP	878	
49	32.729232	192.168.154.131	192.168.154.132	ICMP	118	
50	32.731531	192.168.154.132	192.168.154.131	ICMP	246	
51	32.733290	192.168.154.131	192.168.154.132	ICMP	238	
52	32.741523	192.168.154.132	192.168.154.131	ICMP	814	
54	34.188052	192.168.154.131	192.168.154.132	ICMP	110	

Header checksum: 0x843f [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.154.131
Destination: 192.168.154.132
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x0000 incorrect, should be 0x0ffc
[Expert Info (Warning/Checksum): Bad checksum [shou

29 cf 0c c1 08 00 45 00 ..).....)....E.
84 3f c0 a8 9a 83 c0 a8 .e..@..?.....
00 00 45 00 00 49 39 6bE..I9k
01 01 0a 5f 01 02 c8 8b @.._.....
bf cb 80 18 01 c9 8f ab ..1}S.....
6a ea 00 6d 86 43 53 53m j m CSS
65 6e 53 53 48 5f 35 2e H-2.0.0p enSSH_5.
3..

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.



Source	Destination	Info
192.168.1.100	8.8.8.8	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.8.8	192.168.1.100	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.168.1.100	192.168.1.100	Standard query 0x0003 MX A8D603B0DE000000009AF29E902AB216780EAFD10AA3E4A376A2D9165E78
192.168.1.100	192.168.1.100	Standard query response 0x0003 MX A8D603B0DE000000009AF29E902AB216780EAFD10AA3E4A376A
192.168.1.100	8.8.8.8	Standard query 0x0001 PTR 8.8.8.8.in-addr.arpa
8.8.8.8	192.168.1.100	Standard query response 0x0001 PTR 8.8.8.8.in-addr.arpa PTR dns.google
192.168.1.100	8.8.8.8	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.8.8	192.168.1.100	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.168.1.100	192.168.1.100	Standard query 0x0003 MX 8AA600B0DE75522612EFA9000026FF28FD623A3BD76545C896E6DBC7FEE9
192.168.1.100	192.168.1.100	Standard query response 0x0003 MX 8AA600B0DE75522612EFA9000026FF28FD623A3BD76545C896E
192.168.1.100	8.8.8.8	Standard query 0x0001 PTR 8.8.8.8.in-addr.arpa
8.8.8.8	192.168.1.100	Standard query response 0x0001 PTR 8.8.8.8.in-addr.arpa PTR dns.google
192.168.1.100	8.8.8.8	Standard query 0x0002 PTR 131.94.168.192.in-addr.arpa
8.8.8.8	192.168.1.100	Standard query response 0x0002 No such name PTR 131.94.168.192.in-addr.arpa
192.168.1.100	192.168.1.100	Standard query 0x0003 MX BA7C01B0DE682B2B4554B6000101E88144.dataexfil.com
192.168.1.100	192.168.1.100	Standard query response 0x0003 MX BA7C01B0DE682B2B4554B6000101E88144.dataexfil.com MX