

Ex040

Joshua Main-Smith

2020-10-01

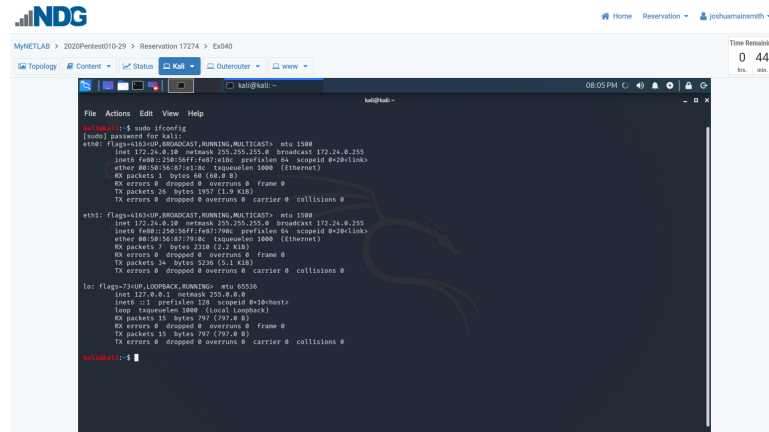
Contents

Attack Narrative	2
Packet Reconnaissance of Plunder	2
Packet Reconnaissance of F4rmC0rp	4
KEY006	5
Zoom Link	6

Attack Narrative

Packet Reconnaissance of Plunder

The ethernet interfaces are shown in the screenshot below.



```
MyNETLAB > 2020Pentest010-29 > Reservation 17274 > Ex040
Topology Content Status Kali Outrouter www
Time Remaining 0 44
brs est.

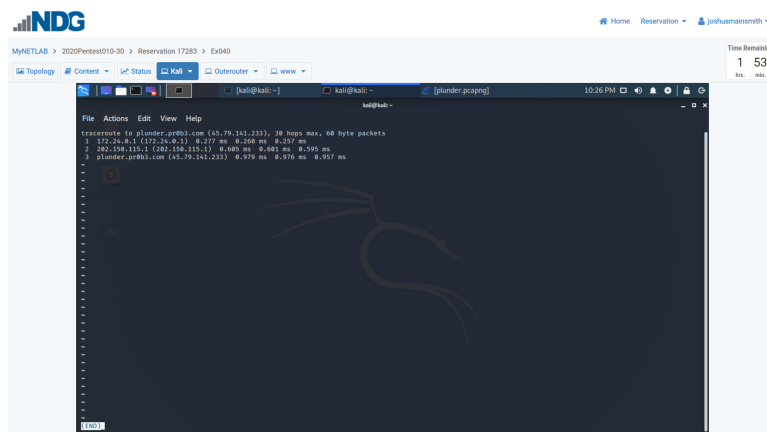
kali@kali:~$ sudo ifconfig
[sudo] password for kali:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.24.0.10 netmask 255.255.255.0 broadcast 172.24.0.255
    inet6 fe80::150:56ff:fe07:4163 prefixlen 64 scopeid 0x20<link>
    ether 08:00:56:07:41:63 txqueuelen 1000 (Ethernet)
    RX packets 1 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 26 bytes 1957 (1.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.24.0.10 netmask 255.255.255.0 broadcast 172.24.0.255
    inet6 fe80::150:56ff:fe07:4163 prefixlen 64 scopeid 0x20<link>
    ether 08:00:56:07:41:63 txqueuelen 1000 (Ethernet)
    RX packets 7 bytes 2318 (2.2 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 34 bytes 5236 (5.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

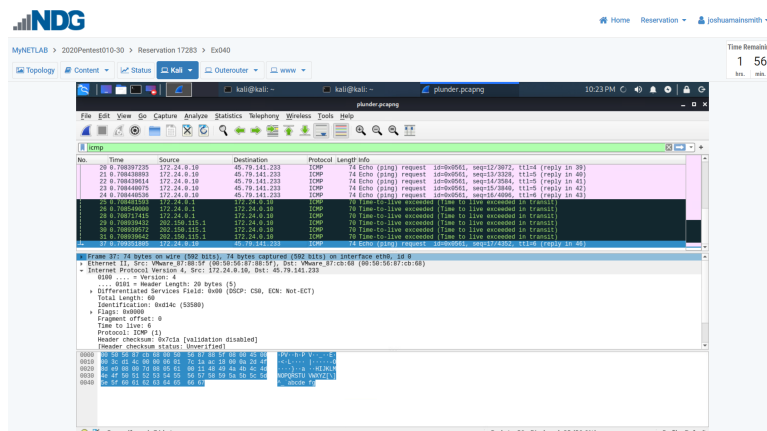
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (local loopback)
    RX packets 15 bytes 767 (0.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 15 bytes 767 (0.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

kali@kali:~$
```

My IP address can be seen under the eth0 interface as 172.24.0.10. To start up Wireshark, we entered the command **sudo wireshark**. Wireshark needs to be run as a privileged user, so the **sudo** command is necessary. The internet interface we chose was eth0. Once Wireshark began capturing packets on this interface, we used the command **traceroute -I plunder.pr0b3.com** to send some packets between our machine and plunder.pr0b3.com. The way traceroute works is by taking note of all the hops between the host machine and the target machine. This is done by taking advantage of the time to live (TTL) scheme used in packet transmission. Traceroute will send three packets for every TTL, then increment by one until the final destination is met. An example output is shown below.



Traceroute will set a TTL to 1 for three packets. If the target host hasn't been reached, the next iteration of packets is set to a TTL of 2, and so on. The screenshot above indicates that it took three hops for the packets to reach its destination, with the first column showing the hop number and the second column showing the IP address for this hop. The last columns are the times for the three packets to reach their destination. The results of the wireshark packet sniffing are in the screenshot below, with the source IP of the packets being our host machine (172.24.0.10) and the target machine being plunder.pr0b3.com (45.79.141.233).



The internet control message protocol is used by routers and other devices to communicate error or operational messages about the connection status of an incoming IP. If an incoming connection failed, the device may issue back an ICMP with a message indicating that the connection failed. In the instance of using traceroute, the responses we got back told us if the TTL exceeded or not (TTL decremented to zero upon arrival) and told us what the TTL was of

the packet upon arrival. In the above example, the TTL of packet number 37 (the last host response) shows it having a TTL of six. With traceroute sending three packets per TTL, the total number of pings would amount to 18. Often when using this command we may hit a router that won't allow any icmp echo messages to reach back to the host machine. An example of this is seen below, using the command **sudo traceroute -I microsoft.com**. At hop 15 and after, there are no packets reaching back to our host machine. This is probably due to filtering the "unlikely" UDP port scheme used in the default method.

```

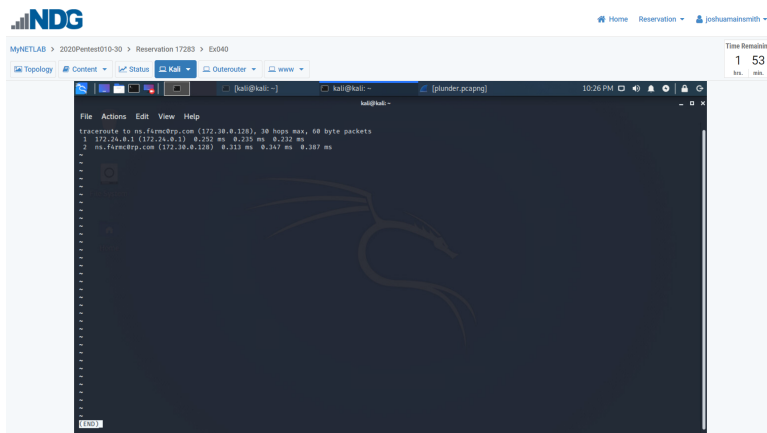
File Actions Edit View Help
kali@kali:~$ sudo traceroute -I microsoft.com
traceroute to microsoft.com (13.77.161.179), 30 hops max, 60 b
yte packets
 1 10.0.2.2 (10.0.2.2) 0.580 ms 0.557 ms 0.691 ms
 2 gateway.home (192.168.254.254) 1.762 ms 2.158 ms 2.261
ms
 3 107-204-182-1.lightspeed.mdsnwi.sbcglobal.net (107.204.182
.1) 3.144 ms 3.304 ms 3.623 ms
 4 76.229.12.76 (76.229.12.76) 3.265 ms 3.408 ms 4.409 ms
 5 * * *
 6 12.83.75.9 (12.83.75.9) 6.515 ms 3.989 ms 4.018 ms
 7 12.123.159.97 (12.123.159.97) 6.623 ms 6.900 ms 7.860 m
s
 8 12.245.40.18 (12.245.40.18) 7.797 ms 8.618 ms 8.595 ms
 9 ae31-0.1cr02.ch2.ntwk.msn.net (104.44.237.21) 9.783 ms 9
.928 ms 10.584 ms
10 be-102-0.1br01.ch2.ntwk.msn.net (104.44.11.249) 77.623 ms
54.916 ms 54.972 ms
11 be-4-0.1br01.dsm05.ntwk.msn.net (104.44.18.215) 125.917 m
s 125.913 ms 126.052 ms
12 be-5-0.1br01.cys04.ntwk.msn.net (104.44.19.86) 55.792 ms
56.222 ms 57.853 ms
13 be-8-0.1br01.mwh01.ntwk.msn.net (104.44.18.223) 56.611 ms
56.724 ms 56.685 ms
14 ae101-0.1cr03.mwh01.ntwk.msn.net (104.44.21.144) 56.051 m
s 56.280 ms 56.276 ms
15 * * *
16 * * *
17 * * *
18 * * *

```

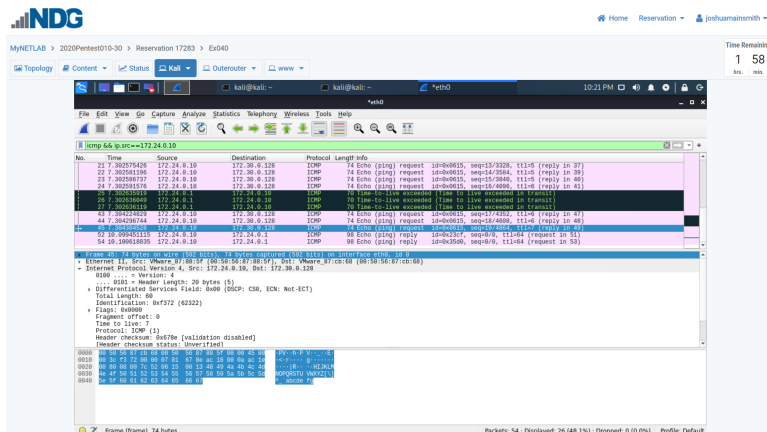
A modern alternative is to use **sudo traceroute -T microsoft.com**, where the **-T** flag specifies the tcp technique, which bypasses the firewalls. It works by using the "half-open" scheme, where applications aren't able to see our probing. A TCP RST is sent instead of a TCP ACK, so that the session is dropped before the machine notices. More of this can be seen in **traceroute --help 2>&1 | less**, where **2>&1** combines the output of stderr to the stdout (redirecting both to the same location).

Packet Reconnaissance of F4rmC0rp

Now that we laid the groundwork for using traceroute and Wireshark, we inspected **ns.f4rmc0rp.com**. Firing up Wireshark and listening in while using the command **sudo traceroute -I ns.f4rmc0rp.com**, we found that there were two hops between our host machine and the target machine (image below). The final IP address of our target domain is 172.30.0.128.

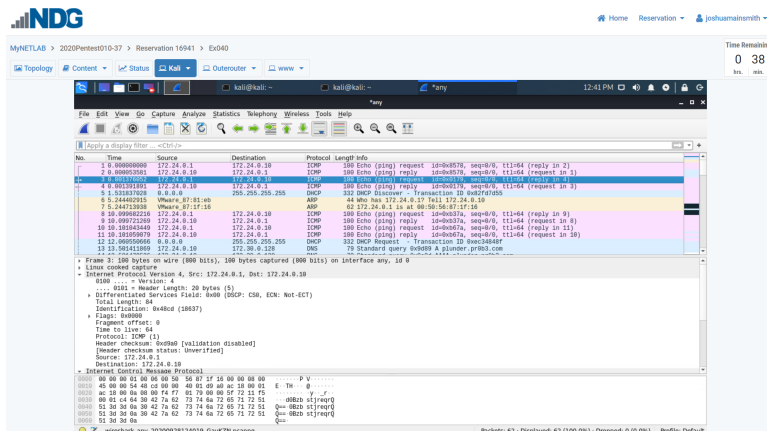
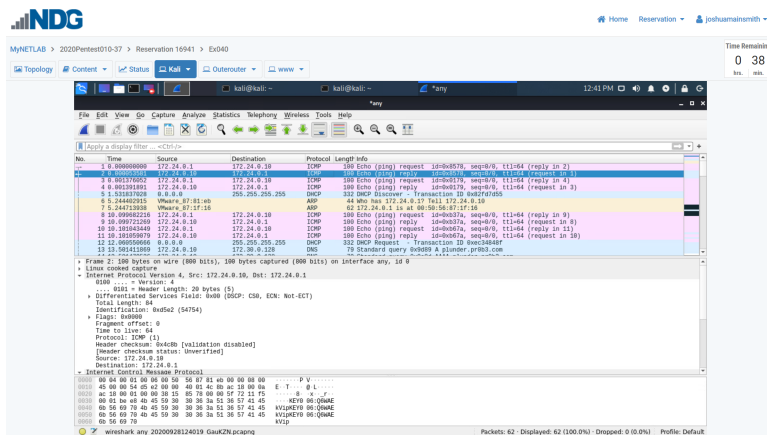


Wireshark picked up that the last packet to reach the host had a TTL of seven. This was the only TTL of seven, so the total number of pings is $6 \times 3 + 1 = 19$, between the source (172.24.0.10) and destination (172.30.0.128). Once reaching its destination, there's no need to continue sending packets.



KEY006

Something interesting to not is a set of packets being sent from 172.24.0.10 to 172.24.0.1 and vice versa.



Something interesting to note from these packets is the data field under the ICMP header contains a string of characters matching what appears to be a key. The value of this key, between two corresponding packets, is **KEY006:Q6WAEkVip0BzbstjreqrQQ==**.

Zoom Link

September 29, 2020