Report on DNS Spoofer and Spoof Detector applications

By:
Rajadorai DS

I. Description of implementation for dnspoison.go:

- Obtain the input specifications from the user using ‘flag’ package

Input Components in dnspoison.go:

- -i Interface name (Ex: en0 , lo0 etc.)
- -f hostnames (Ex: poisonhosts)
- [expression] BPF Filter (Ex: “udp and port 53”)

Steps involved in dnspoison.go:

a) Read packets from interface:

- Capture packets in promiscuous mode from the user-specified interface (ie. value of arg -i). If the interface is not provided, then a device name is picked up using packet.FindAllDevs() and an interface to capture the packets is chosen.

b) Filter for DNS query packets and send spoofed response:

- If input file for **“hostnames” is provided** by the user via the -f arg, the file is read and each malicious IP-Hostname pair on each line is processed and stored.

  The malicious IP - Hostname pair is provided in the following format:

10.6.6.6	www..netflix.com
10.6.6.6	www.nba.com

- As each DNS Query packet is processed, the **hostnames in the file** are **compared** with the DNS query packet’s **question name** (the extracted string from the packet’s “layers.DNSQuestion” field). If there is a **match**, the information from this packet (such as Transaction ID, source IP etc) is extracted and a **fake response is created** and sent to the interface by handcrafting a **UDP packet** with protocol type **DNS**.

- If the “hostnames” file is **not** provided, no matching/comparison is done and a **spoofed** response is sent for **all** corresponding DNS query packets.

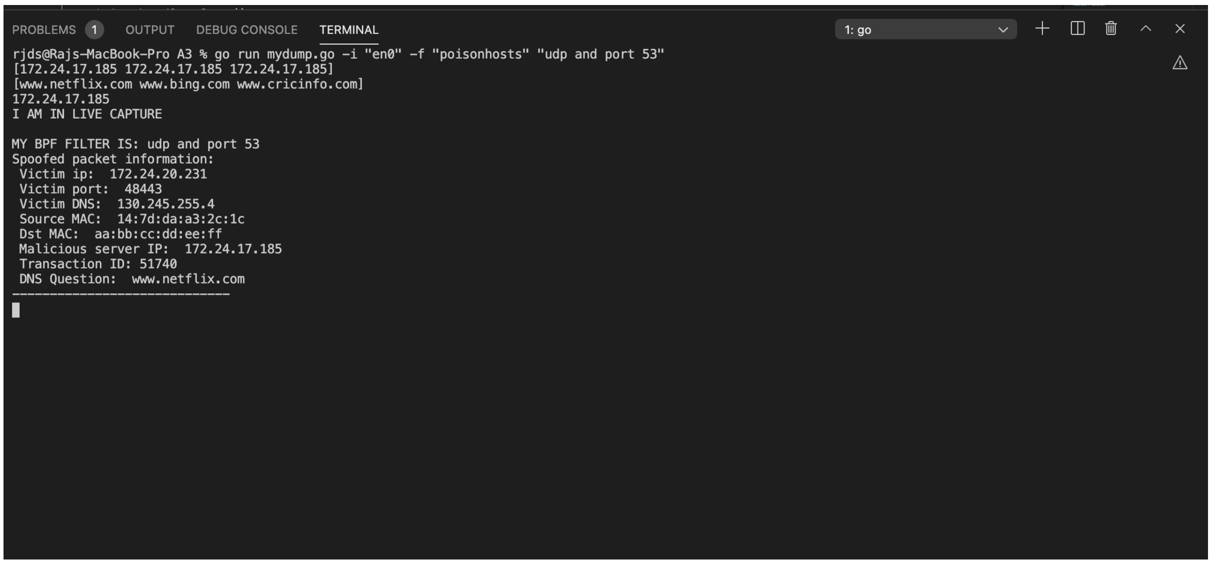
Sample Input Patterns:

(Note: Input flags (-i , -f) can be in any order and the BPF Filter (if present) should be at the end of the input line as the final argument - as it does not have an identifier flag. All arguments are **optional**)

- go run dnspoison.go -i “en0” -f “poisonhosts” “udp and port 53”
- go run dnspoison.go -f “poisonhosts” “udp and port 53”
- go run dnspoison.go -i “en0” -f “poisonhosts” “udp and port 53”
- go run dnspoison.go -f “poisonhosts” “udp and port 53”
- go run dnspoison.go “udp and port 53”
- go run dnspoison.go

If BPF filter is not specified, “udp and port 53” filter will be specified automatically by the program.

Sample output of dnspoison.go (Prints information of all spoofed packets):



```
rjds@Rajs-MacBook-Pro A3 % go run mydump.go -i "en0" -f "poisonhosts" "udp and port 53"
[172.24.17.185 172.24.17.185 172.24.17.185]
[www.netflix.com www.bing.com www.cricinfo.com]
172.24.17.185
I AM IN LIVE CAPTURE

MY BPF FILTER IS: udp and port 53
Spoofed packet information:
 Victim ip:  172.24.20.231
 Victim port:  48443
 Victim DNS:  130.245.255.4
 Source MAC:  14:7d:da:a3:2c:1c
 Dst MAC:  aa:bb:cc:dd:ee:ff
 Malicious server IP:  172.24.17.185
 Transaction ID: 51740
 DNS Question:  www.netflix.com
-----------------------------
```

II. Description of implementation for dnsdetect.go:

- Obtain the input specifications from the user using ‘flag’ package

  Input Components in dnsdetect.go:

  - -i Interface name (Ex: en0 , lo0 etc.)
  - -r Tracefile (Ex:spoofAttack.pcap)
  - [expression] BPF Filter (Ex: “udp and port 53”)

Steps involved in dnsdetect.go:

- **Determine mode of capture/detection:**

  a) Read packets from trace file

  - If trace filename is provided, the packets are captured from it (even if an interface name is provided by the user for live capture).

  b) Capture packets from interface

  - If the trace file is not given, the packets are captured via **live capture** through the interface given in the user input. If the interface is also not provided, then a device name is picked up using packet.FindAllDevs() and an interface to capture the packets is chosen.

- **Filter for DNS query and response packets and check for spoof:**

  - Two frequency counters are maintained for each unique “transaction ID + hostname” (domain name) pair.

  - One frequency counter (in the form of a map) maintains the frequency of a “transaction ID + hostname” pair is **DNS query** packets.

    (Stored as => r**equestFrequency map[string]int**)

    The other frequency counter maintains the frequency of a “transaction ID + hostname” pair is **DNS response** packets.

    (Stored as => r**esponseFrequency map[string]int**)

  - As each packet is captured, the program checks if:

    responseFrequency[key] > requestFrequency[key], where “key” is the "transaction ID + hostname" pair string. (For ex: key = "12345|netflix.com", where "12345" is the transaction ID and "netflix.com" is the domain name)

If the above condition is true, the **timestamp** of the first DNS query packet for a specific "key" value is compared with the **timestamp** of the latest DNS response packet that has arrived with the same "**key"** value. If the difference between the two timestamps is **less than 5 seconds,** the spoof condition is satisfied and a **spoof alert is raised**.

Summary of strategy for dnsdetect.go:

To summarize the above steps, frequency of the response and request packets for a particular "transaction ID+hostname" string (AKA "key") is maintained. If the **frequency** of the **DNS response** packets for a specific "key" **> frequency** of the **DNS query** packets for the same "key" **AND** the time difference between the first request and the latest response is **less than 5 seconds,** a **spoof alert** is raised.

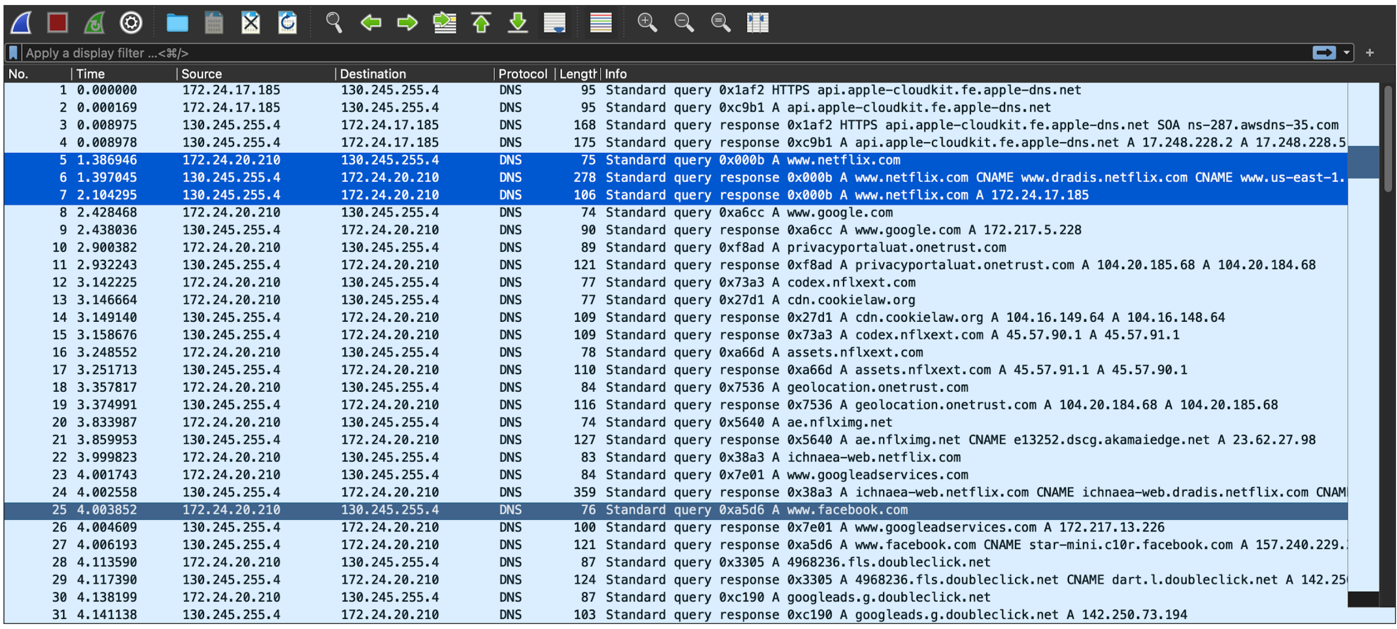
Sample Input Patterns:

(Note: Input flags (-i , -r) can be in any order and the BPF Filter (if present) should be at the end of the input line as the final argument - as it does not have an identifier flag. All arguments are **optional**)

- go run dnsdetect.go -i "en0" -r "spoofAttack.pcap" "udp and port 53"
- go run dnsdetect.go -r "spoofAttack.pcap" "udp and port 53"
- go run dnsdetect.go -i "en0" -r "spoofAttack.pcap" "udp and port 53"
- go run dnsdetect.go -r "spoofAttack.pcap" "udp and port 53"
- go run dnsdetect.go "udp and port 53"
- go run dnsdetect.go

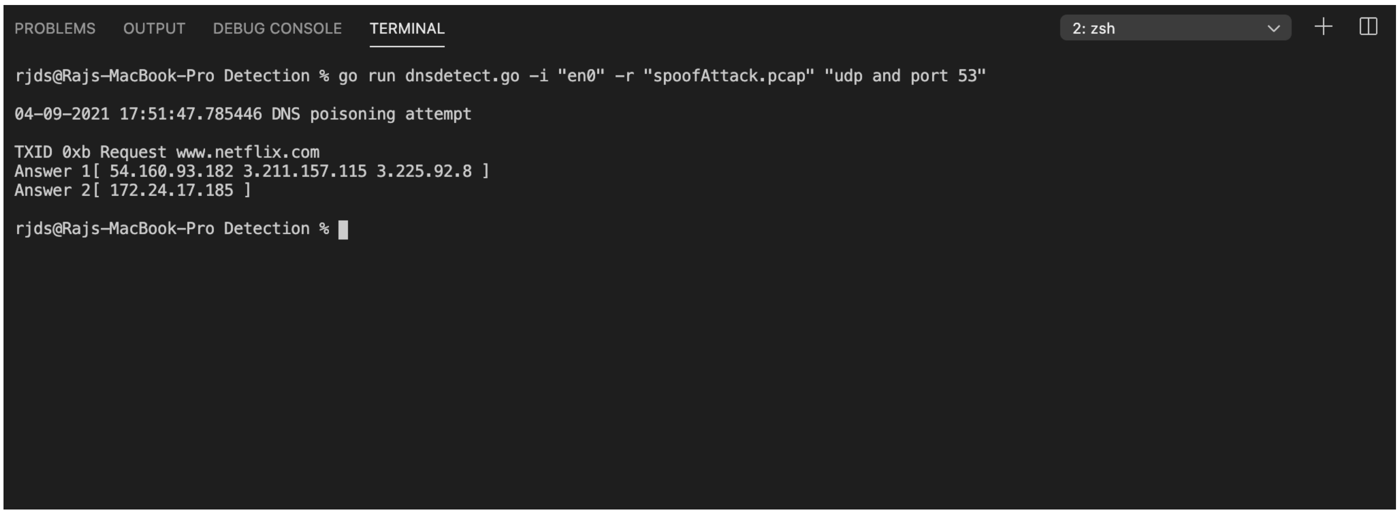
If BPF filter is **not specified**, "udp and port 53" filter will be specified **automatically** by the program.

Screengrab from "spoofAttack.pcap" (Packet Trace file containing my spoof attack):



No.	Time	Source	Destination	Protocol	Lengtł	Info
1	0.000000	172.24.17.185	130.245.255.4	DNS	95	Standard query 0x1af2 HTTPS api.apple-cloudkit.fe.apple-dns.net
2	0.000169	172.24.17.185	130.245.255.4	DNS	95	Standard query 0xc9b1 A api.apple-cloudkit.fe.apple-dns.net
3	0.008975	130.245.255.4	172.24.17.185	DNS	168	Standard query response 0x1af2 HTTPS api.apple-cloudkit.fe.apple-dns.net SOA ns-287.awsdns-35.com
4	0.008978	130.245.255.4	172.24.17.185	DNS	175	Standard query response 0xc9b1 A api.apple-cloudkit.fe.apple-dns.net A 17.248.228.2 A 17.248.228.5
5	1.386946	172.24.20.210	130.245.255.4	DNS	75	Standard query 0x000b A www.netflix.com
6	1.397045	130.245.255.4	172.24.20.210	DNS	278	Standard query response 0x000b A www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-east-1.
7	2.104295	130.245.255.4	172.24.20.210	DNS	106	Standard query response 0x000b A www.netflix.com A 172.24.17.185
8	2.428468	172.24.20.210	130.245.255.4	DNS	74	Standard query 0xa6cc A www.google.com
9	2.438036	130.245.255.4	172.24.20.210	DNS	90	Standard query response 0xa6cc A www.google.com A 172.217.5.228
10	2.900382	172.24.20.210	130.245.255.4	DNS	89	Standard query 0xf8ad A privacyportaluat.onetrust.com
11	2.932243	130.245.255.4	172.24.20.210	DNS	121	Standard query response 0xf8ad A privacyportaluat.onetrust.com A 104.20.185.68 A 104.20.184.68
12	3.142225	172.24.20.210	130.245.255.4	DNS	77	Standard query 0x73a3 A codex.nflxext.com
13	3.146664	172.24.20.210	130.245.255.4	DNS	77	Standard query 0x27d1 A cdn.cookielaw.org
14	3.149140	130.245.255.4	172.24.20.210	DNS	109	Standard query response 0x27d1 A cdn.cookielaw.org A 104.16.149.64 A 104.16.148.64
15	3.158676	130.245.255.4	172.24.20.210	DNS	109	Standard query response 0x73a3 A codex.nflxext.com A 45.57.90.1 A 45.57.91.1
16	3.248552	172.24.20.210	130.245.255.4	DNS	78	Standard query 0xa66d A assets.nflxext.com
17	3.251713	130.245.255.4	172.24.20.210	DNS	110	Standard query response 0xa66d A assets.nflxext.com A 45.57.91.1 A 45.57.90.1
18	3.357817	172.24.20.210	130.245.255.4	DNS	84	Standard query 0x7536 A geolocation.onetrust.com
19	3.374991	130.245.255.4	172.24.20.210	DNS	116	Standard query response 0x7536 A geolocation.onetrust.com A 104.20.184.68 A 104.20.185.68
20	3.833987	172.24.20.210	130.245.255.4	DNS	74	Standard query 0x5640 A ae.nflximg.net
21	3.859953	130.245.255.4	172.24.20.210	DNS	127	Standard query response 0x5640 A ae.nflximg.net CNAME e13252.dscg.akamaiedge.net A 23.62.27.98
22	3.999823	172.24.20.210	130.245.255.4	DNS	83	Standard query 0x38a3 A ichnaea-web.netflix.com
23	4.001743	172.24.20.210	130.245.255.4	DNS	84	Standard query 0x7e01 A www.googleadservices.com
24	4.002558	130.245.255.4	172.24.20.210	DNS	359	Standard query response 0x38a3 A ichnaea-web.netflix.com CNAME ichnaea-web.dradis.netflix.com CNAM
25	4.003852	172.24.20.210	130.245.255.4	DNS	76	Standard query 0xa5d6 A www.facebook.com
26	4.004609	130.245.255.4	172.24.20.210	DNS	100	Standard query response 0x7e01 A www.googleadservices.com A 172.217.13.226
27	4.006193	130.245.255.4	172.24.20.210	DNS	121	Standard query response 0xa5d6 A www.facebook.com CNAME star-mini.c10r.facebook.com A 157.240.229.
28	4.113590	172.24.20.210	130.245.255.4	DNS	87	Standard query 0x3305 A 4968236.fls.doubleclick.net
29	4.117390	130.245.255.4	172.24.20.210	DNS	124	Standard query response 0x3305 A 4968236.fls.doubleclick.net CNAME dart.l.doubleclick.net A 142.25
30	4.138199	172.24.20.210	130.245.255.4	DNS	87	Standard query 0xc190 A googleads.g.doubleclick.net
31	4.141138	130.245.255.4	172.24.20.210	DNS	103	Standard query response 0xc190 A googleads.g.doubleclick.net A 142.250.73.194

Output of dnsdetect.go when the above packet trace file is given as input:



```
rjds@Rajs-MacBook-Pro Detection % go run dnsdetect.go -i "en0" -r "spoofAttack.pcap" "udp and port 53"

04-09-2021 17:51:47.785446 DNS poisoning attempt

TXID 0xb Request www.netflix.com
Answer 1[ 54.160.93.182 3.211.157.115 3.225.92.8 ]
Answer 2[ 172.24.17.185 ]

rjds@Rajs-MacBook-Pro Detection %
```

Sidenote for winning the race between actual response and spoofed response:

The SBU DNS server responses are almost instant and the SBU network blocks connections to most 3rd party DNS servers such as Yandex etc. My mobile hotspot connection did not give internet access to the VM (virtualbox on Mac OS X) when attached via bridged mode (reference: Piazza question @161) and hence, I was unable to change the DNS server to a slower/moderately timed 3rd party DNS server in order to win the race