

Dos Attack To The DNS Server With Spoofed IP:

Student-id: 1505107

Source files:

1. **DNS_QueryBuilder.py** : This is a query builder. Given the domain name and dns server ip and port it will construct a dns query according to RFC-1035. This will return the query in hexadecimal that can be directly send using udp sockets and also if it is provided with the dns response, it can analyze the response and tell the ip address of the desired domain.
2. **dos.py** : This is the main attack tool. Inside it we are constructing packets, the payload is the dns query made using DNS_QueryBuilder.py. Inside an infinite while loop we will continuously send dns query thus launch a dns flood attack.

Attack Steps:

1. We make a dns query and fix our target, make a packet.
2. Then we enter an infinite while loop.
3. As the packets are sent in an infinite loop, a keyboard interrupt catching code has been set up. If we use 'Ctrl+c' the code will stop executing and before terminating it will print the number of packets sent so far.

Snapshots of the attack:

1. **dns query:** here we test the DNS_QueryBuilder.py. In test.py we simply take an udp socket and using DNS_QueryBuilder.py construct a query to get ip of the domain "codeforces.com". After sending it we wait for a udp response and analyze it using DNS_QueryBuilder.py. For convenience the tester code has been given in a separate "tester codes" folder

```
[09/06/19]seed@VM:~/Documents$ python3 test.py
00010100000100000000000000a636f6465666f7263657303636f6d0000010001
udp sent
No errors reported
This server isn't an authority for the given domain name
Recursion was desired for this request
Recursion is available on this DNS server
TTL : 20397 seconds
RDLENGTH : 4 bytes
77.234.215.195
[09/06/19]seed@VM:~/Documents$
```

Fig. dns query in action

25759	56.050410	192.168.0.100	8.8.8.8	DNS	74 Standard query 0x0001 A codeforces.com
25791	56.114572	8.8.8.8	192.168.0.100	DNS	90 Standard query response 0x0001 A codeforces.com A 77.234.215.195

Fig. query observed in wireshark

2. ip has been spoofed:

3559	7.043855	192.168.0.100	8.8.8.8	DNS	74 Standard query 0x0001 A codeforces.com
------	----------	---------------	---------	-----	---

Fig. query sent but no response from dns server

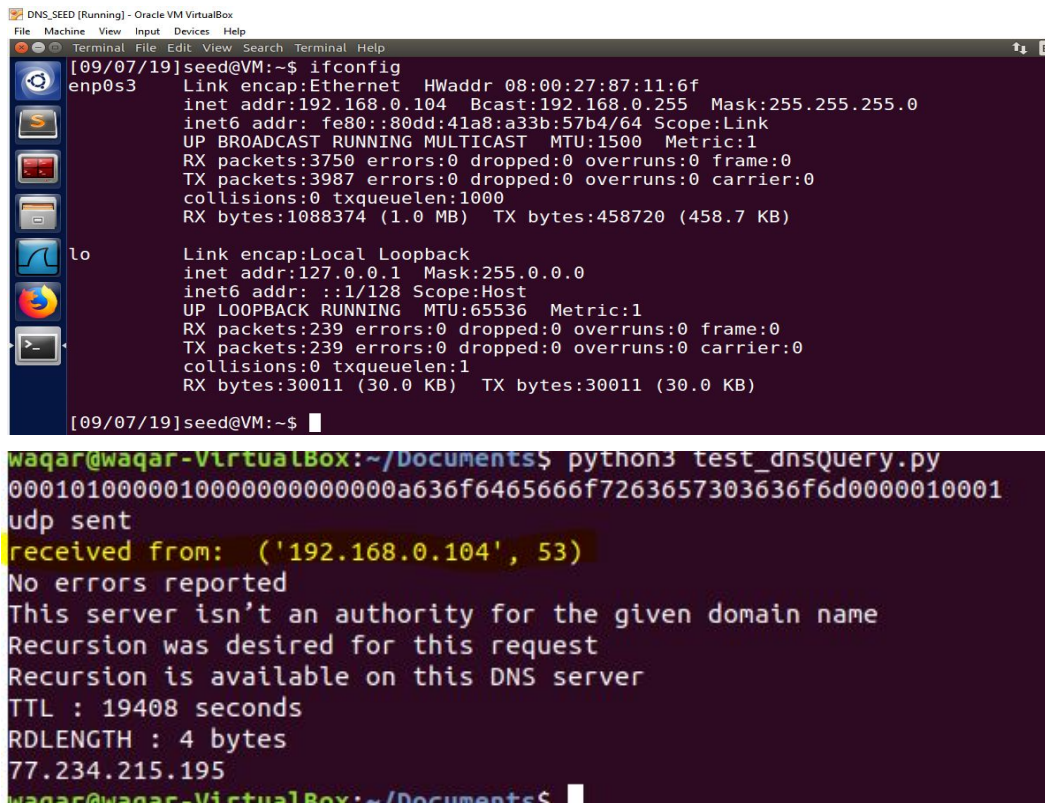
```
[09/07/19]seed@VM:~/Documents$ sudo python3 test.py
WARNING: No route found for IPv6 destination :: (no default route?). This affects only IPv6
.
Sent 1 packets.
^CTraceback (most recent call last):
  File "test.py", line 29, in <module>
    data, address = sock.recvfrom(4096)
KeyboardInterrupt
[09/07/19]seed@VM:~/Documents$ sudo python3 test.py
WARNING: No route found for IPv6 destination :: (no default route?). This affects only IPv6
.
Sent 1 packets.
```

Fig. waiting for response

We can see that when we use scapy to change the src ip we send the dns query but do not get any response. In the terminal it is still waiting for a response.

3. **Attack:** Now we simulate the attack on dns server. We use bind9 in ubuntu to make the pc to work as a dns server. As the dns server we use seed ubuntu as bind9 is pre-built in

it. After running we give the following command: 'ifconfig' to get the servers ip



The image contains two terminal screenshots. The top screenshot shows the output of the 'ifconfig' command in a terminal window titled 'DNS_SEED [Running] - Oracle VM VirtualBox'. It displays details for the 'enp0s3' interface, including its IP address (192.168.0.104), broadcast address (192.168.0.255), and other network statistics. The bottom screenshot shows the output of a Python script 'python3 test_dnsQuery.py' in a terminal window titled 'waqar@waqar-VirtualBox: ~/Documents\$'. The output shows a successful DNS query to 192.168.0.104 on port 53, returning the IP address 77.234.215.195.

```
[09/07/19]seed@VM:~$ ifconfig
enp0s3 Link encap:Ethernet HWaddr 08:00:27:87:11:6f
       inet addr:192.168.0.104 Bcast:192.168.0.255 Mask:255.255.255.0
       inet6 addr: fe80::80dd:41a8:a33b:57b4/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
       RX packets:3750 errors:0 dropped:0 overruns:0 frame:0
       TX packets:3987 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:1088374 (1.0 MB) TX bytes:458720 (458.7 KB)

lo      Link encap:Local Loopback
       inet addr:127.0.0.1 Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING MTU:65536 Metric:1
       RX packets:239 errors:0 dropped:0 overruns:0 frame:0
       TX packets:239 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1
       RX bytes:30011 (30.0 KB) TX bytes:30011 (30.0 KB)

[09/07/19]seed@VM:~$

waqar@waqar-VirtualBox:~/Documents$ python3 test_dnsQuery.py
000101000000100000000000000000a636f6465666f7263657303636f6d00000010001
udp sent
received from: ('192.168.0.104', 53)
No errors reported
This server isn't an authority for the given domain name
Recursion was desired for this request
Recursion is available on this DNS server
TTL : 19408 seconds
RDLENGTH : 4 bytes
77.234.215.195
waqar@waqar-VirtualBox:~/Documents$
```

Fig. response from bind9 dns server

In another vm we use ubuntu 18.04 where we run the tester code written for DNS_QueryBuilder.py and observe that indeed the response is coming from the dns server we have created. To do this we just need to set the destination ip and port as 192.168.0.104 and 53. The ip of the dns server may change when we change the network so we have to change the code accordingly.

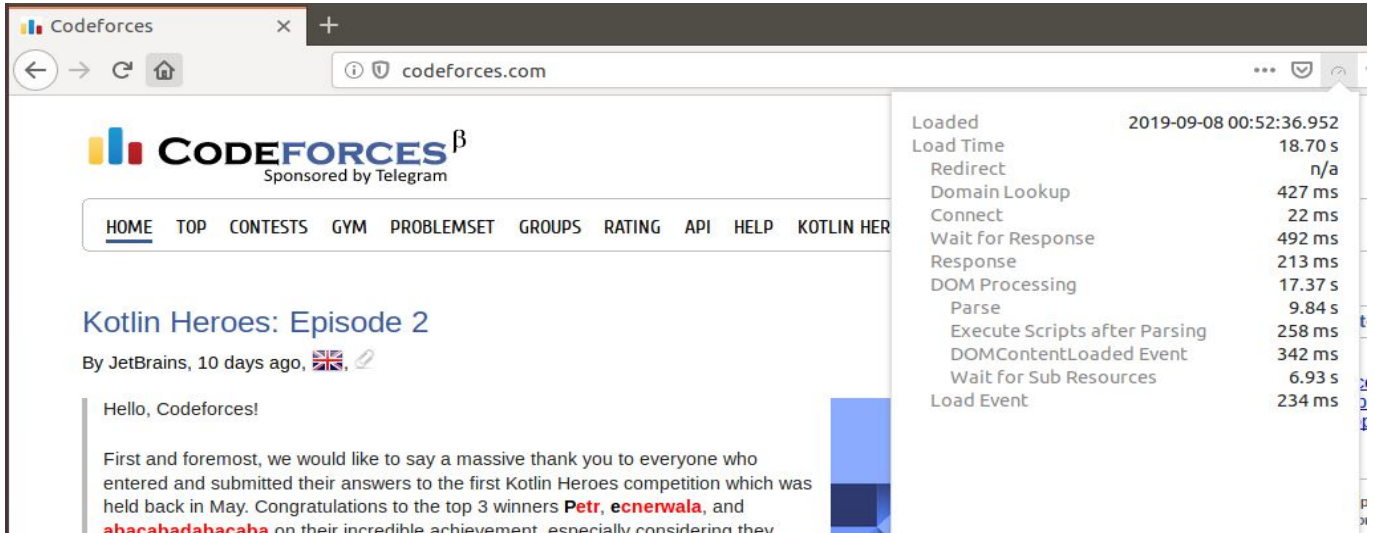
Then we take another vm that will work as the user-pc. We edit the file '/etc/resolv.conf'. There we write -> "namespace ip_of_dns" - here ip of dns is 192.168.0.104 in my case. Now whenever we use our browser it will use the dns server on 192.168.0.104. If we use anything other than it say 105, we shall observe that the browser is unable to find the desired domain that proves that editing the file has ensured we use our dns.

Now finally if we use our attacker-pc and flood the dns then the browser of the user-pc won't be able to find any domain

We run the code in three different terminals. Here we use a firefox extension "Load-Time" [<https://addons.mozilla.org/en-US/firefox/addon/load-time/>] to check if the attack is working

Does the attack works?:

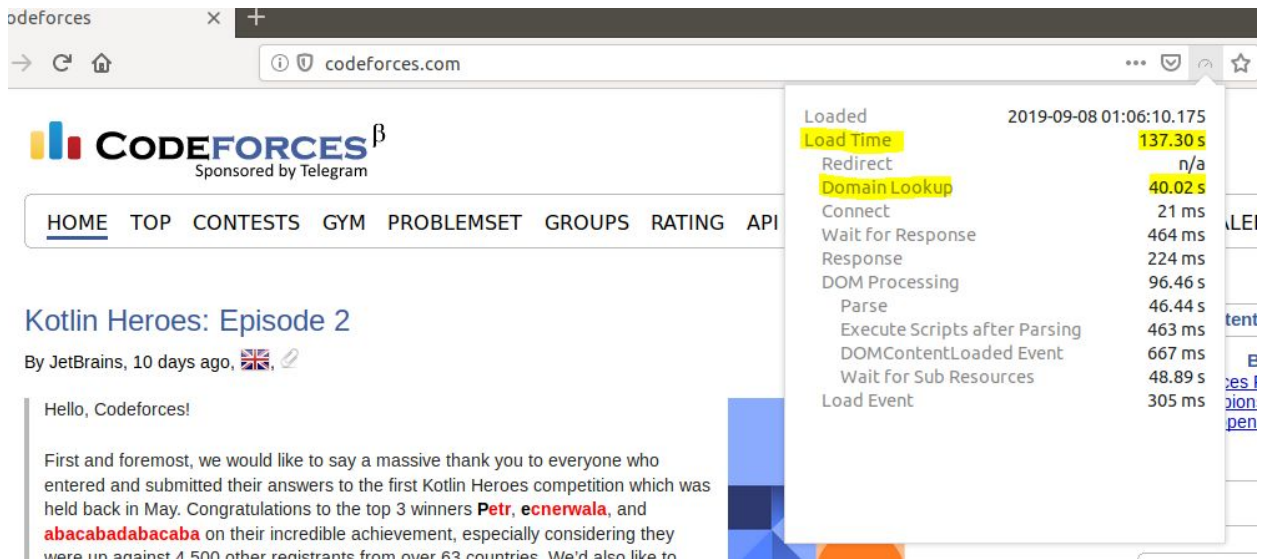
The attack works but without spoofing the src ip. If we check the screenshots below, they taken before running the attack and after running the attack. Though not always it becomes impossible to load the website, the attack definitely increases the traffic and thus the load-time of the websites increases.



The screenshot shows the Codeforces website before the attack. The browser window displays the Codeforces logo and navigation menu. The main content area shows the article "Kotlin Heroes: Episode 2" by JetBrains, dated 10 days ago. A performance overlay on the right side of the browser shows the following data:

Loaded	2019-09-08 00:52:36.952
Load Time	18.70 s
Redirect	n/a
Domain Lookup	427 ms
Connect	22 ms
Wait for Response	492 ms
Response	213 ms
DOM Processing	17.37 s
Parse	9.84 s
Execute Scripts after Parsing	258 ms
DOMContentLoaded Event	342 ms
Wait for Sub Resources	6.93 s
Load Event	234 ms

Fig. codeforces before the attack



The screenshot shows the Codeforces website after the attack. The browser window displays the Codeforces logo and navigation menu. The main content area shows the article "Kotlin Heroes: Episode 2" by JetBrains, dated 10 days ago. A performance overlay on the right side of the browser shows the following data:

Loaded	2019-09-08 01:06:10.175
Load Time	137.30 s
Redirect	n/a
Domain Lookup	40.02 s
Connect	21 ms
Wait for Response	464 ms
Response	224 ms
DOM Processing	96.46 s
Parse	46.44 s
Execute Scripts after Parsing	463 ms
DOMContentLoaded Event	667 ms
Wait for Sub Resources	48.89 s
Load Event	305 ms

Fig. codeforces after the attack



Fig. wikipedia before the attack

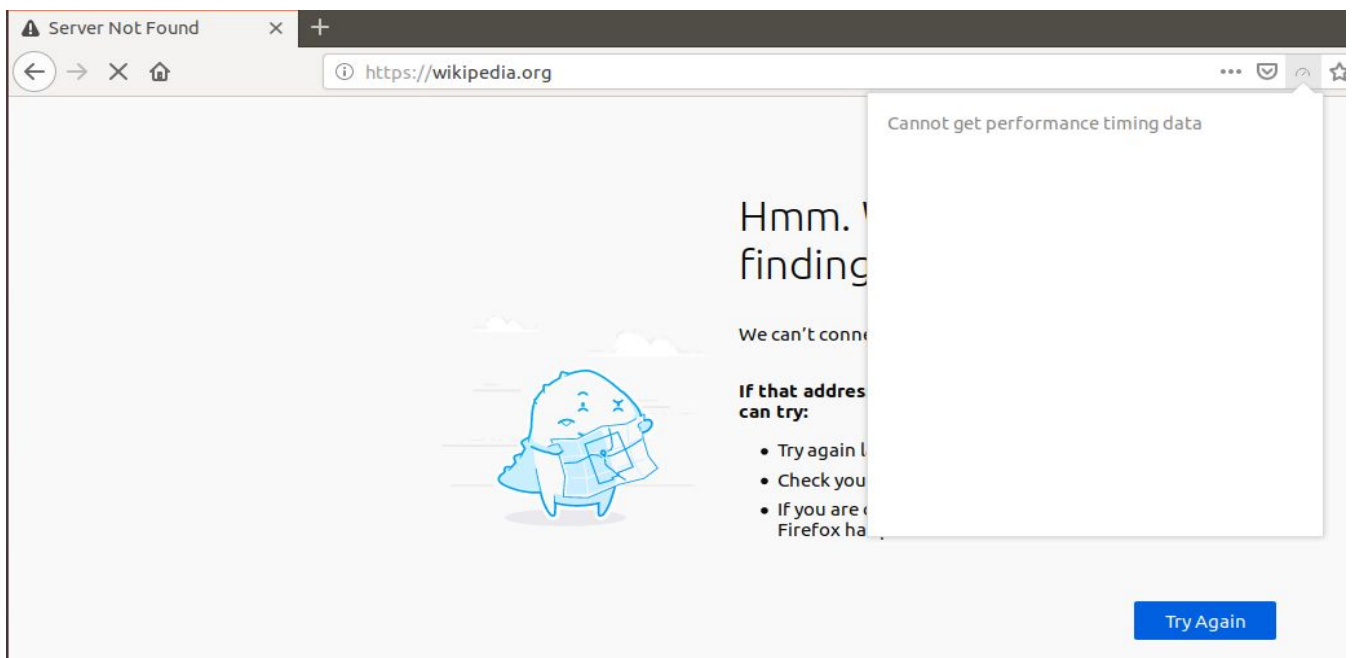


Fig wikipedia after the attack, it didn't load

Why spoofing is Not working:

The plan was to flood the dns server with the query written from scratch according to RFC-1035 using an infinite while loop. To spoof the src ip scapy packet crafting library was supposed to be used, as sending the query through a udp won't let us hide our ip. But the scapy library sends packet really slowly, e.g: within 30s scapy sends approx. 3500 packets where normal udp sends more than 100000 packets. Three terminals were used to run the code and within 30s they sent

121741, 124982 and 183077. So instead spoofing normal packets were sent so that the dns server can be flooded.