

ASSIGNMENT 3

Harsh Shah

10/17/2016

1 Task 1: HOSTS file attack

Since, I used VMware for this lab, different IP addresses were assigned to the Vm's, every time I established a new connection. For this attack, the IP addresses are:

DNS server: 192.168.74.134

www.example.com: 192.168.74.101

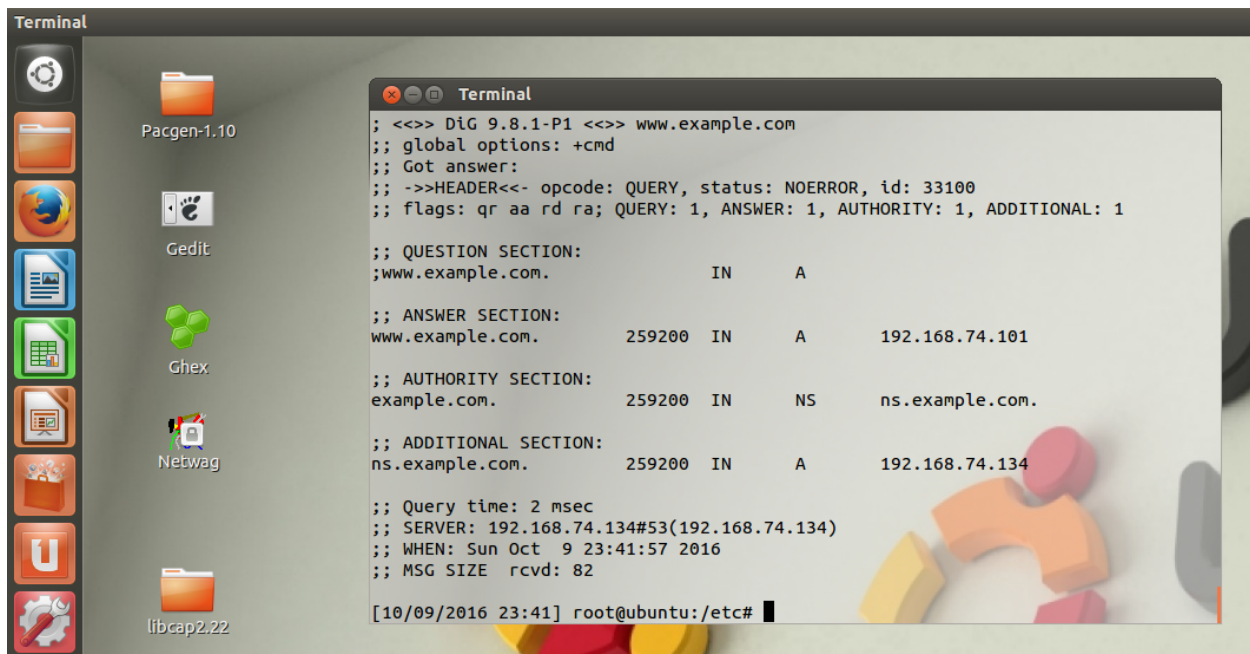


Figure 1. Original IP of www.example.com

1.1 Steps :

After the initial configuration is done, the only thing to do, is to modify the /etc/hosts file to provide a spoofed mapping of www.example.com. This is done using the following steps.

1. Login as root user using su command

2. Go to /etc directory

3. Use a file editing command to add your spoofed mapping to the /etc/hosts file: *nano hosts*

In my case, I added the mapping "74.125.196.147 www.example.com" to the hosts file, which is one of the IP addresses of www.google.com. This mapping is shown in Figure 2.

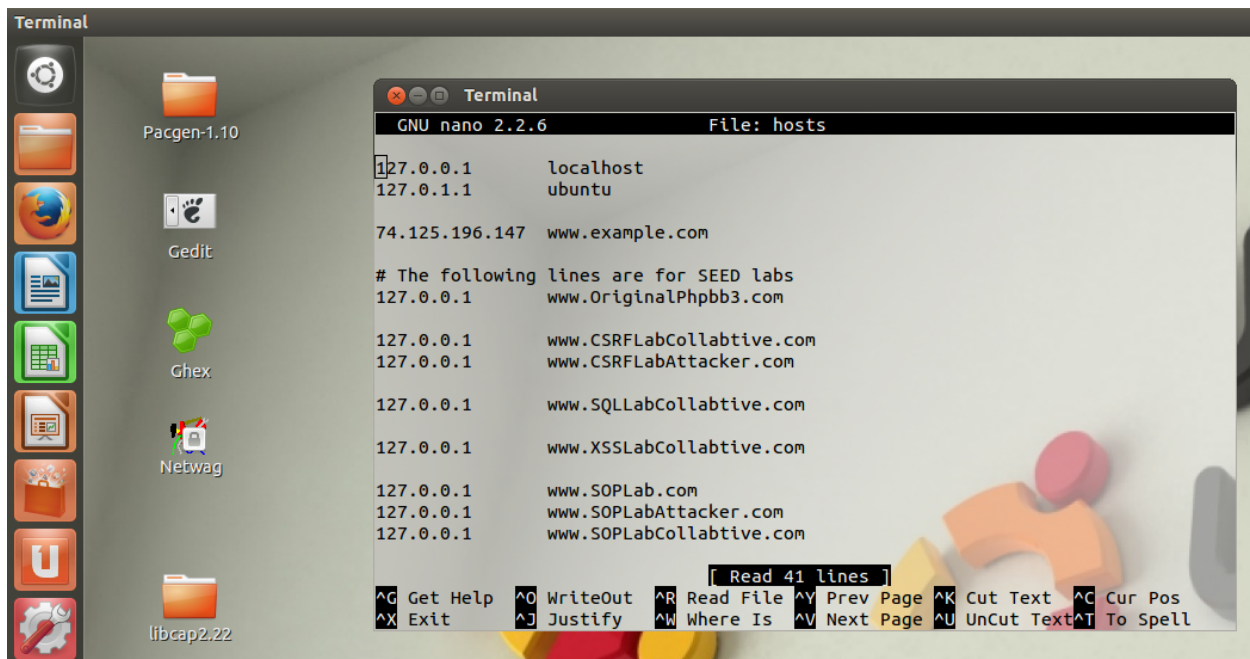


Figure 2. Modified hosts file

The work of attacker is done here. Now, we wait for the victim to use www.example.com. This is shown in Figure 3, where the user tries to ping www.example.com, and replies come from 74.125.96.147. Thus, the attack is successful.

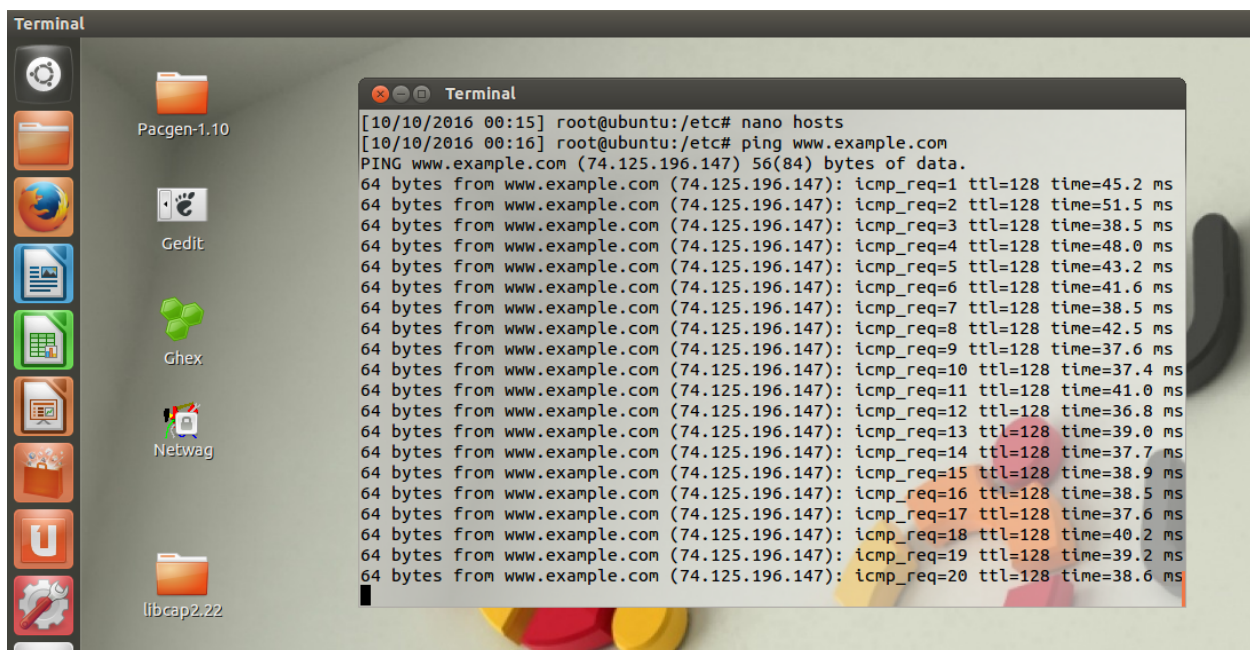


Figure 3. Attack is successful

Observations: nslookup and dig ignore the hosts file. Attack works for ping and browser.

1.2 Real-world problem and viability :

A real-world attacker may spoof the IP of a banking site such as *www.wellsfargo.com* in the hosts file, to make it point to a site under his/her control, which looks like *www.wellsfargo.com*. Now, it is easy to obtain the victim's username and password. This type of phishing is more efficient and difficult to detect, as unlike traditional e-mail phishing, the user does not even have to click on a malicious link. Also, URL checking cannot be used to detect the phishing site, as a legitimate URL is mapped to an IP of a malicious site. Since, a user does not usually check the hosts file, he/she is unaware that an attack has even taken place.

This attack assumes that an attacker gains control of the victim's machine, which is not very easy today, due to the use of strong passwords. Hence, this attack is not too viable. But, if an attacker can fool the victim into downloading and installing a malware, he/she can run a program to change the hosts file.

1.3 Conclusion :

This attack is difficult to implement today, given the security of machines, malicious link filters and ad-blockers. But once successful, it is very difficult to detect unless the victim checks the hosts file.

2 Task 2: Host-Level Response Spoofing

2.1 Description :

This attack requires the following three machines to be connected to the same network,

1. DNS server (192.168.74.145)
2. Attacker (192.168.74.140)
3. Victim (192.168.74.144)

When the victim requests for *www.example.com*, it gets a reply from the DNS server i.e *192.168.74.101*. Our goal is to send a spoofed DNS response containing some IP of your choice (IP of *www.wikipedia.com*, in my case) to the victim before the reply from DNS server reaches him/her. This can be done using Netwag tool.

2.2 Steps :

1. Launch Netwag.
2. Select the tool 'sniff and send DNS answers (105)'.
3. Put appropriate values in the fields.

hostname: Put the website you are trying to spoof (*www.example.com*)

hostname IP: IP address of your choice (*www.wikipedia.com*)

authns: Name of the victim site's nameserver (*ns.example.com*)

authns IP: IP address of the nameserver (*192.168.74.145*)

You may use *filter* to only monitor the packets of your victim (*src host 192.168.74.144*)

The following command can be run in Netwag to provide the same functionality,

```
105 -hostname "www.example.com" -hostnameip 208.80.154.224 -authns "ns.example.com" -authnsip 192.168.74.144 -filter "src host 192.168.74.144"
```

4. Run

The Netwag tool generates DNS answers based on your parameters and the queryID that it captures by

sniffing DNS queries. As soon as the victim asks for *www.example.com*, it receives our spoofed response and goes to the website of our choice (*www.wikipedia.com*). The attack can be demonstrated using Figures 4,5 and 6.

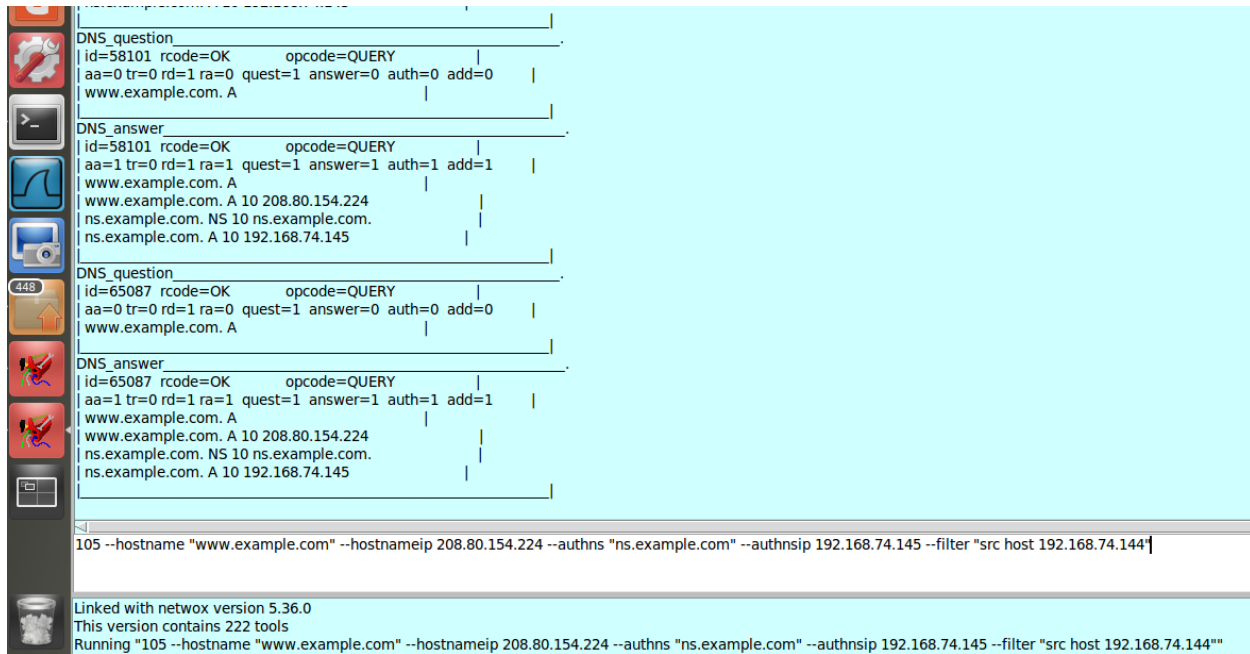


Figure 4. Netcat Tool on Attacker's machine

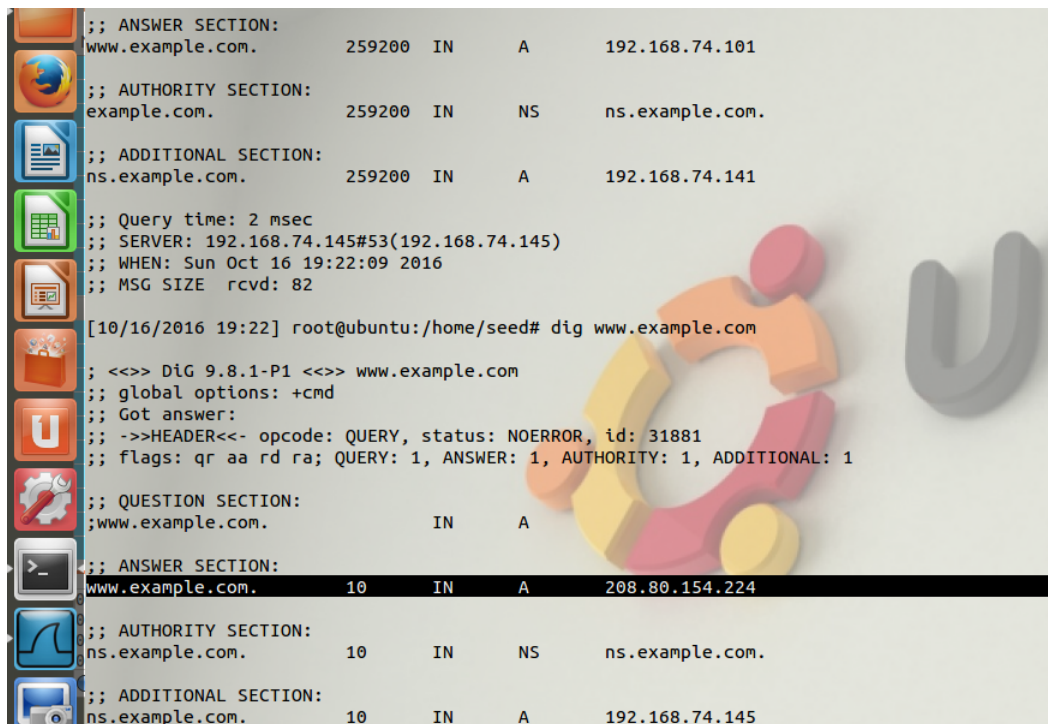


Figure 5. Output of the dig command on Victim's machine

The figure above shows that the IP has been changed, as *dig www.example.com* returned the spoofed IP.

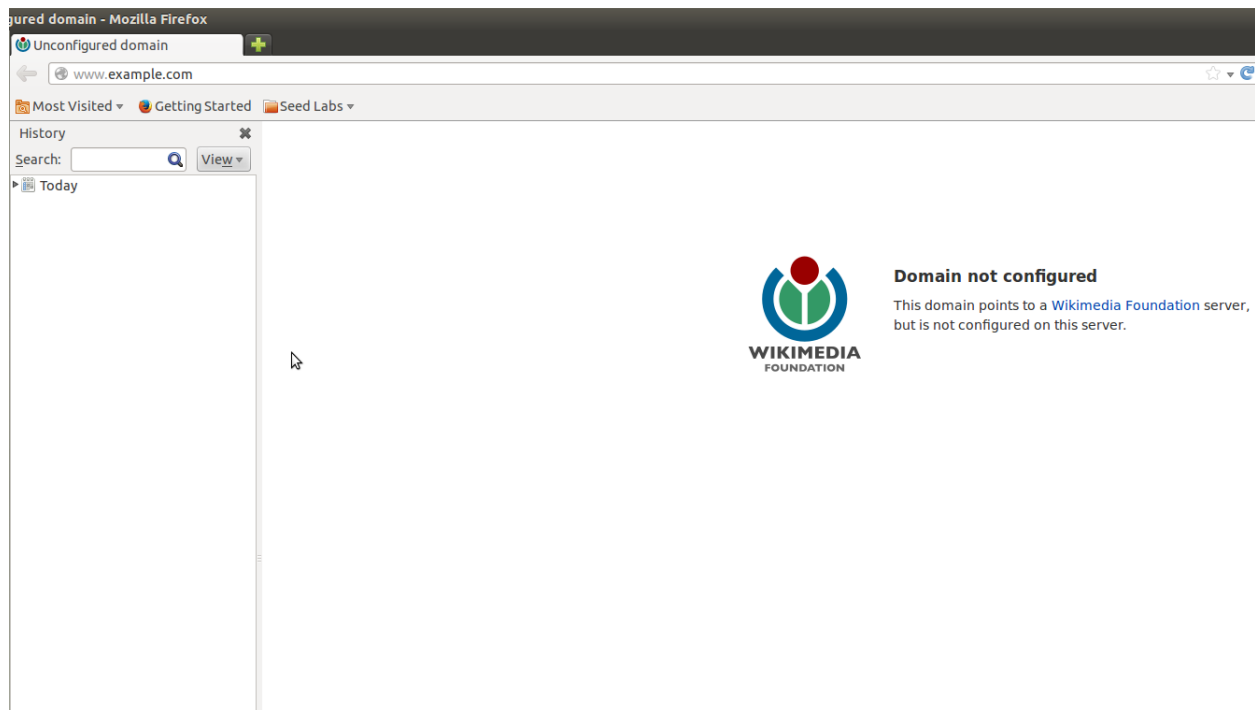


Figure 6. Attack is successful

The above figure shows that the victim is indeed directed to *www.wikipedia.com*. Since Wikipedia checks the domain name, it will not display the website, but the attack is actually successful. This was just for demonstration purposes.

2.3 Real-world problem and viability :

A real-world attacker could use this attack to direct the victim to his/her site for phishing as mentioned in the first attack, or provide a non-existent IP address to cause Denial of service. The key to making sure that this attack works, is to correctly identify the transaction ID of DNS query. Since, the attacker is on the same network as the victim (Insiders), it is easy to find transaction ID using tools like Netwag or Wireshark. This puts a limitation on the scope of the attack, as attacker has to be present in the same network, which is not always possible. Also, the attacker has to sniff every request that the victim makes and send a spoofed response every time. This is not viable for the attacker. A better way would be where the attacker has to spoof the response only once. This is done using cache poisoning.

3 Task 3: Server-Level Response Spoofing

3.1 Description :

Unlike the previous attack, this attack sends spoofed responses directly to the victim's DNS server. Once successful, the spoofed mapping lives in the DNS server's cache for a specified time (TTL). This is known as cache poisoning. I will demonstrate an attack that would point *www.yahoo.com* (98.139.183.24) to *www.gmail.com* (216.58.194.37).

Machines required to be connected to the same network:

1. DNS server (192.168.74.141)

2. Attacker (192.168.74.140)

3.2 Steps :

This attack involves the same steps as mentioned in the previous attack. The only difference is, the values in hostname IP, authns, authns IP and filter fields. The name (authns) and IP address of name server (authns IP) of the website that you are trying to spoof, can be found by querying the website and monitoring reply, using Wireshark.

The following command shows the values used in my case,

105 -hostname "www.yahoo.com" -hostnameip 216.58.194.37 -authns "ns.yahoo.com" -authnsip 92.242.140.2 -ttl 600 -filter "src host 192.168.74.141" -spoofip "rawlink"

Once the response is accepted, this command would cause the poisoned mapping to live in server's cache for 10 minutes (TTL=600 seconds).

Figures 7,8 and 9 demonstrate the attack.

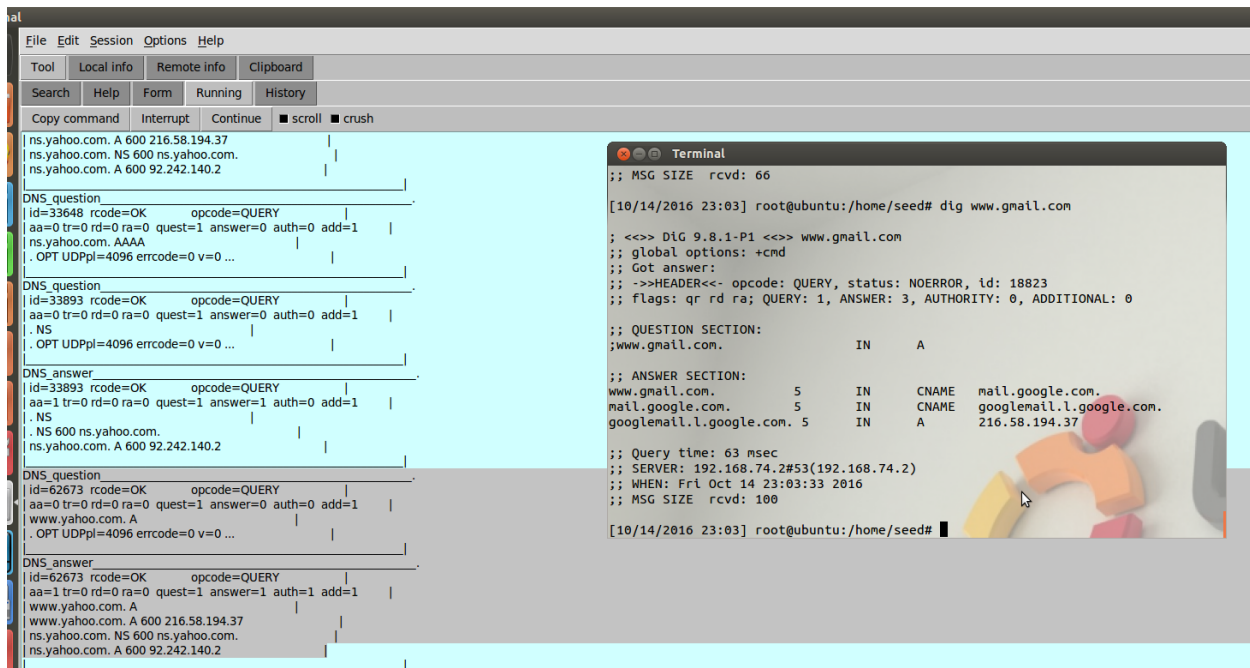
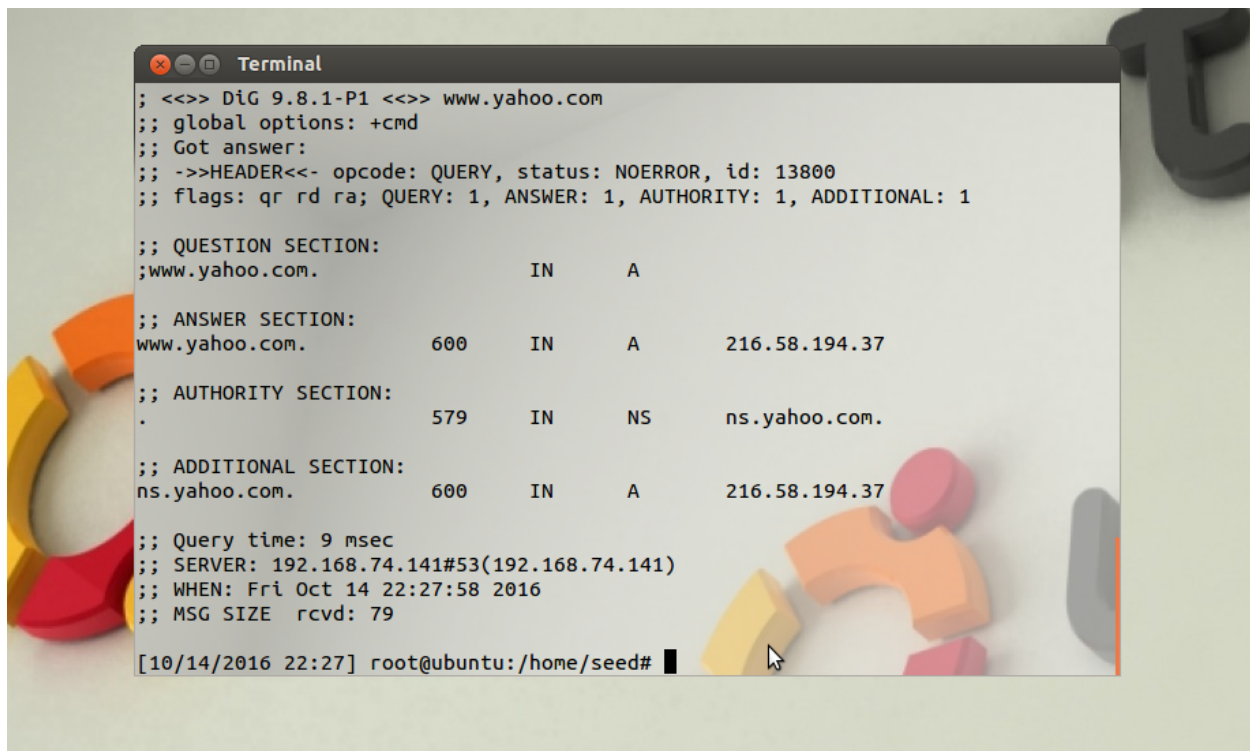


Figure 7. Netwag tool query and response

As seen from the figure above, the IP address 216.58.194.37 belongs to *www.gmail.com*, and we sent a DNS spoofed response, mapping this IP to target site *www.yahoo.com*. The next figure shows that mapping is successful.



```
Terminal
; <<>> DiG 9.8.1-P1 <<>> www.yahoo.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13800
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.yahoo.com.                IN      A

;; ANSWER SECTION:
www.yahoo.com.                600     IN      A      216.58.194.37

;; AUTHORITY SECTION:
.                             579     IN      NS      ns.yahoo.com.

;; ADDITIONAL SECTION:
ns.yahoo.com.                600     IN      A      216.58.194.37

;; Query time: 9 msec
;; SERVER: 192.168.74.141#53(192.168.74.141)
;; WHEN: Fri Oct 14 22:27:58 2016
;; MSG SIZE rcvd: 79

[10/14/2016 22:27] root@ubuntu:/home/seed#
```

Figure 8. Output of dig command on the Victim's machine

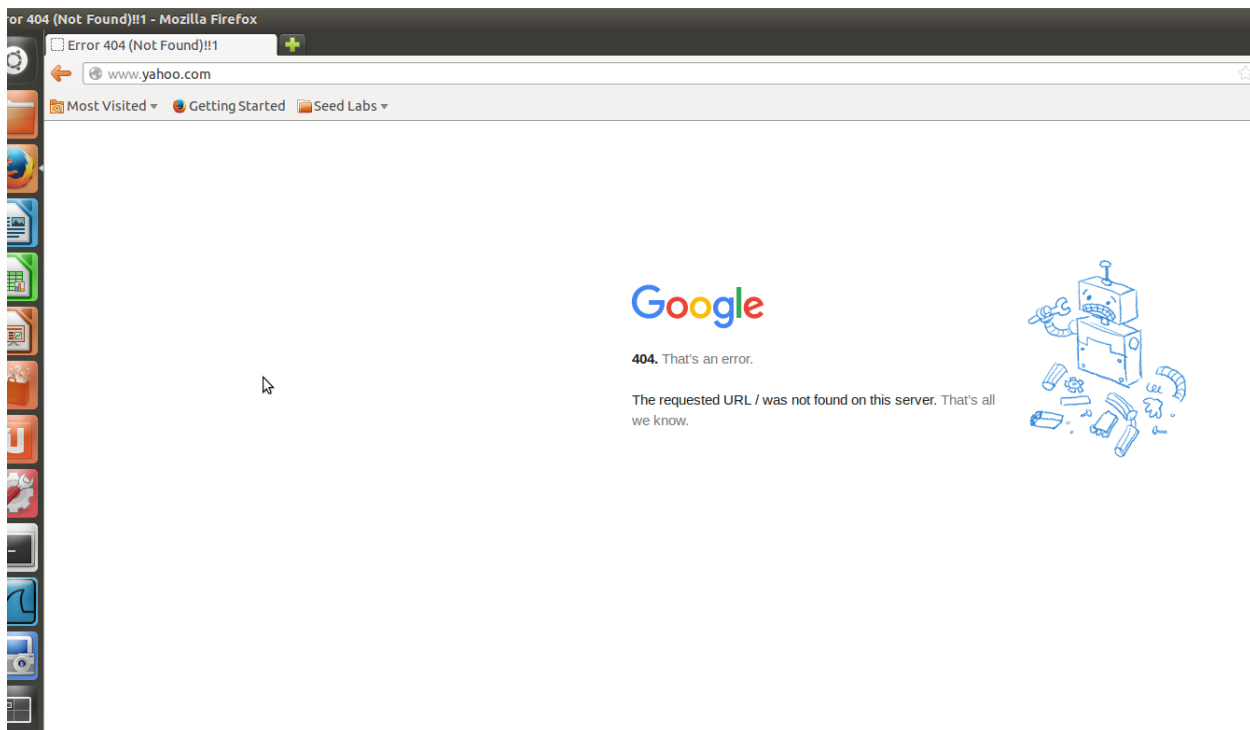


Figure 9. Attack is successful

The figure above shows that *www.yahoo.com* indeed got directed to one of Google's servers (*www.gmail.com*).

3.3 Real-world problem and viability :

This attack is a serious problem as the victim can do nothing about it. Also, all the users that are served by the poisoned DNS server are affected by this attack, which could lead to pharming attacks. Once the DNS cache is poisoned, the attacker's work is done and he can relax for a specified time (TTL). After TTL amount of time, the attacker carries out the attack again.

But, like the previous attack, this attack also requires the adversary to be present on the same network as DNS server (Insiders), which is again difficult.

4 Task 4: Kaminsky Attack

4.1 Description :

This attack assumes that the attacker is not on the same network as the victim or DNS server. Note that since I am using VMware, both the attacker and DNS server are actually on the same network, but we assume that they are not. I created a 'second' Bind server called forward resolver, which takes DNS queries for the domain *dnsphishinglab.com*, forwarded by the first DNS server. This stops the "leaking" of packets onto the internet. Also, I turned off the WiFi on my Laptop to make sure that there is no internet connectivity. The following figure shows that when a victim requests for *www.dnsphishinglab.com*, the request first goes to the default DNS server, which in turn forwards the request to second DNS server.

9	2016-10-16	18:57:14.781	192.168.74.143	192.168.74.2	DNS	81 Standard query AAAA changelogs.ubuntu.com
13	2016-10-16	18:57:17.671	192.168.74.144	192.168.74.145	DNS	82 Standard query A www.dnsphishinglab.com
15	2016-10-16	18:57:17.671	192.168.74.145	192.58.128.30	DNS	70 Standard query NS <Root>
17	2016-10-16	18:57:17.671	192.168.74.145	192.168.74.143	DNS	93 Standard query A www.dnsphishinglab.com
18	2016-10-16	18:57:17.671	192.168.74.143	192.168.74.145	DNS	142 Standard query response A 192.168.74.110
19	2016-10-16	18:57:17.671	192.168.74.145	192.168.74.144	DNS	131 Standard query response A 192.168.74.110
20	2016-10-16	18:57:18.471	192.168.74.145	199.7.83.42	DNS	70 Standard query NS <Root>
21	2016-10-16	18:57:19.271	192.168.74.145	192.33.4.12	DNS	70 Standard query NS <Root>
22	2016-10-16	18:57:19.751	192.168.74.143	192.168.74.2	DNS	81 Standard query AAAA changelogs.ubuntu.com
25	2016-10-16	18:57:20.071	192.168.74.145	192.203.230.10	DNS	70 Standard query NS <Root>
26	2016-10-16	18:57:20.881	192.168.74.145	192.5.5.241	DNS	70 Standard query NS <Root>
27	2016-10-16	18:57:21.681	192.168.74.145	128.63.2.53	DNS	70 Standard query NS <Root>
28	2016-10-16	18:57:22.481	192.168.74.145	128.8.10.90	DNS	59 Standard query NS <Root>
33	2016-10-16	18:57:23.281	192.168.74.145	192.228.79.201	DNS	59 Standard query NS <Root>
34	2016-10-16	18:57:24.081	192.168.74.145	192.112.36.4	DNS	59 Standard query NS <Root>
35	2016-10-16	18:57:24.881	192.168.74.143	192.168.74.2	DNS	93 Standard query AAAA changelogs.ubuntu.com.localdomain
36	2016-10-16	18:57:24.881	192.168.74.145	198.41.0.4	DNS	59 Standard query NS <Root>

Figure 9. Wireshark packet capture on DNS server

The figure above clearly shows that the attacker (*192.168.74.144*) requests the DNS server (*192.168.74.145*) for *www.dnsphishinglab.com*, which in turn forwards the request to forward resolver (*192.168.74.143*). Now, the DNS server sends the query to NS <Root> as well, by default. This could cause a possible leak to the internet. But, since I have turned off the internet connectivity on my machine, there is no chance of leakage. The forward resolver looks up the request in its zone file and sends an appropriate answer to the DNS server, which then forwards the response to the attacker. Thus, we have established that our forwarding works and that packet leakage is not an issue.

4.2 Steps :

1. The *pacgen.c* file loads the *eth_header*, *ip_header* and other payload files. So, we need to modify the header files to include our source and destination addresses. (Note that the IP addresses of machines have changed again.)

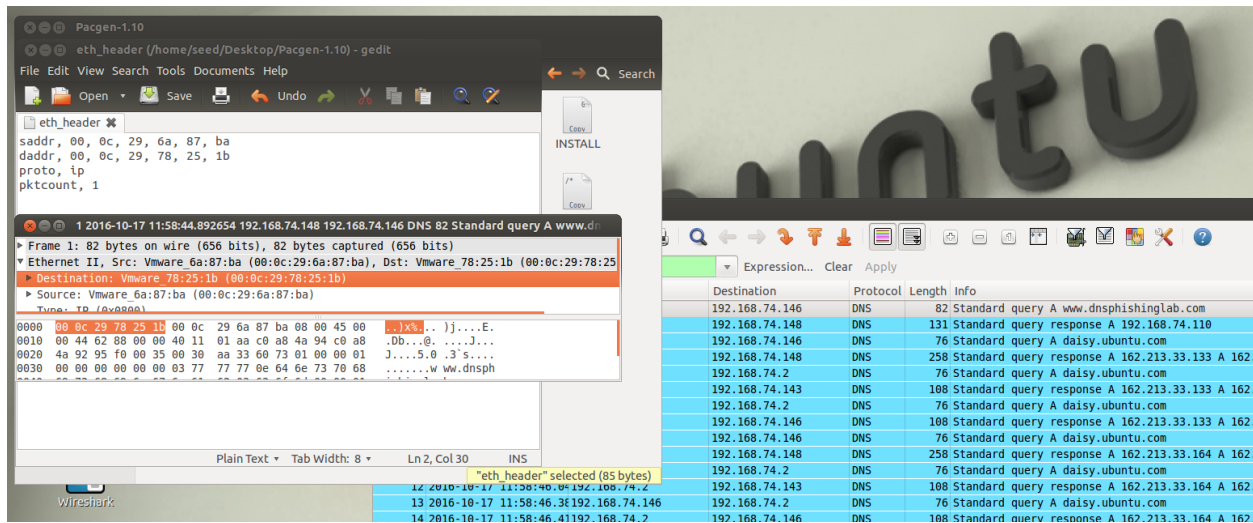


Figure 9. Changing source and destination addresses in ethheader file

2. The next step is to create a payload query file.
 3. Modify the pacgen.c file to add an infinite loop or a loop of some count that chooses random numbers between 0 and 65536, for transaction number prediction. So, the program first creates a DNS query using the payload generator file and then many DNS answers including random transaction ids.
- Unfortunately, the attack did not work for me i.e. the cache did not get poisoned.

4.3 Real-world problem and viability :

This attack is the most dangerous of all the attacks mentioned above, as it can be launched off-path. Also, it does not require the victim or DNS server to send a DNS query, in order to poison the cache, as it creates and sends a non-legitimate website query itself. This attack was deemed so catastrophic that experts predicted it would bring the entire internet down. A real-world attacker has to just keep sending fake DNS queries (*xyz.dnsphishinglab.com*) and appropriate responses including different transaction ID's and an authority section that says '*www.dnsphishinglab.com* maps to this spoofed IP', until one of them matches. This is very viable as the transaction id to be predicted has only 16 bits.

Source port randomization makes this attack harder to implement as it includes extra 16 bits to be predicted. The 0X20 encoding could be used to add more complexity of bit prediction. The best way to tackle Kaminsky attack is to use DNSSEC, but it is not very popular among local ISP's today. However, note that these bit randomizations will not help against an on-path adversary.