**Lab 7**

**Testing the DNS Setup**

**Get the IP address of ns.attacker32.com**

When we run the following dig command, the local DNS server will forward the request to the Attacker nameserver due to the forward zone entry added to the local DNS server’s configuration file.

A screenshot of a computer

Description automatically generated

The answer is coming from the zone file (attacker32.com.zone) that we set up on the Attacker nameserver.

**Get the IP address of www.example.com**

Two nameservers are now hosting the example.com domain, one is the domain’s official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get.

We run dig to get the IP address of [www.example.com](http://www.example.com) by sending the query to the local DNS server, which will send the query to example.com’s official nameserver.

A screenshot of a computer

Description automatically generated

We now run dig to send the query directly to ns.attacker32.com

A computer screen shot of a computer code

Description automatically generated

We can see that the IP result given is from the attacker’s name server instead of the local DNS server.

**Task 1: Directly Spoofing Response to User**

In this task, we need to spoof the DNS response to the user.

Code

A screenshot of a computer program

Description automatically generated

The iface argument is the interface name for the 10.9.0.0/24 network.

Now we run the attack program on the user machine.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Task 2: DNS Cache Poisoning Attack – Spoofing Answers**

Here we need to poison the DNS cache to overcome having to send a spoofed reply every time a DNS packet is received.

Code

A screenshot of a computer screen

Description automatically generated

We can flush the cache using the following command.



We run the program as shown below and then we run dig example.com as shown below.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

This shows that the attack was successful because the answer section shows our spoofed reply.

We can dump the cache to check if it was poisoned as shown below.

A screenshot of a computer

Description automatically generated

**Task 3: Spoofing NS Records**

Here we need to poison the DNS cache and modify the NS record too. We modify the above program to include a spoofed NS record.

ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.com domain. **ns.attacker32.com** is controlled by attackers, it can provide a forged answer for any query as its cached by the local DNS server.

A screenshot of a computer program

Description automatically generated

We can flush the cache using the following command.



We run the program as shown below and then we run the dig command with any hostnames in the example.com domain as shown below.



we run the dig command with any hostnames in the example.com domain as shown below

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

We can see that the answer section is modified for any hostname.

We can dump the cache to check if it was poisoned.



**Task 4: Spoofing NS Records for Another Domain**

Here we need to poison the cache and in addition to example.com we add an additional NS record for google.com.

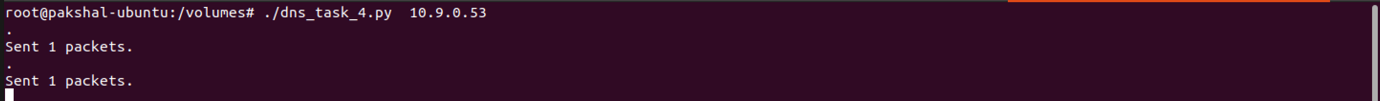
**Code**

**A screenshot of a computer program

Description automatically generated**

A new record is added and nscount I have set it 2 as shown above.

After flushing out the cache we run the program on the attacker machine

****

we run the dig command for any hostname in the example.com domain. Here I have used auth.example.com as shown below.

**A screenshot of a computer

Description automatically generated**

Cache dump

****

This outcome reveals that only the example.com record is present in the cache. This is due to the absence of any configuration information regarding google.com within the lab setup. In this setup, example.com is equipped with a configured zone, thereby establishing an authoritative server for that domain. Conversely, google.com lacks such configuration, preventing the DNS server from caching its information.

**Task 5: Spoofing Records in the Additional Section**

Here we need to spoof certain entries in the additional section and observe whether the target local DNS server caches them successfully.

Code

**A screenshot of a computer screen

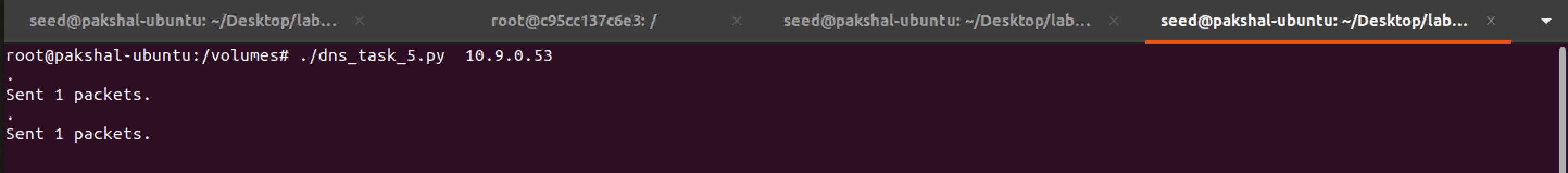
Description automatically generated**

Here I have 3 new records in the additional section.

We flush out the cache and then run the program on the attacker machine.



Now we run the program with 10.9.0.53 and run dig command and check.

****

**A screenshot of a computer

Description automatically generated**

Here we dump the local DNS Server cache and look at it to see if our attack was indeed successful

****

We've manipulated entries in the additional section, and the DNS server has cached them. However, it's evident that the entries for facebook.com are not cached because there are no configurations for domains other than example.com.