COMP 8506 Assignment 3

Perform and Analyze Active Sniffing of a Switched LAN

Xinghua Wei

A00978597

Task 1 – Password Sniffing

Introduction

Password sniffing is an attack on the internet that is used to steal usernames and passwords from the network. This method can be used during active network sniffing in a switched environment.

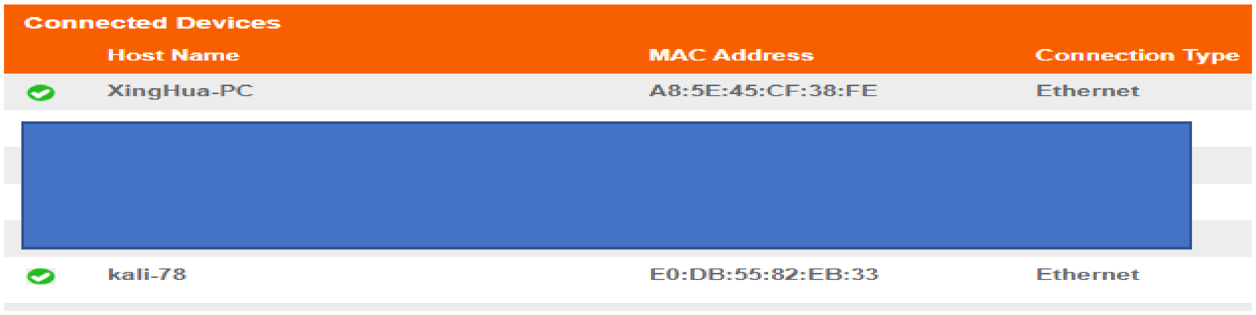
Most networks today work on switches. Unlike hubs, switches look at the Media Access Control (MAC) address related to each frame passing through it and send the data to the specified port. A MAC address may be a hardware address that uniquely identifies each node of a network.

For this exercise, I will perform a password sniffing by using ARP poisoning, an active sniffing technique. ARP Poisoning, aka ARP spoofing, is a type of cyber-attack carried out over a Local Area Network that involves sending malicious ARP packets to a default gateway on a LAN in order to change the pairings in its IP to MAC address table. The attacker sending a false ARP reply message to the default network gateway, informing it that his MAC address should be associated with the target's IP address. Once the default gateway has received this message and broadcasts its changes to all other devices on the network, all of the target's traffic to any other device on the network travels through the attacker's computer, allowing the attacker to inspect or modify it before forwarding it to its real destination.

Experiment

Step 1: Attacker connects to a port of the target system

My switch has 4 ports. I will use an internet cable to connect my attack machine to the switch where my target machine is already connected.



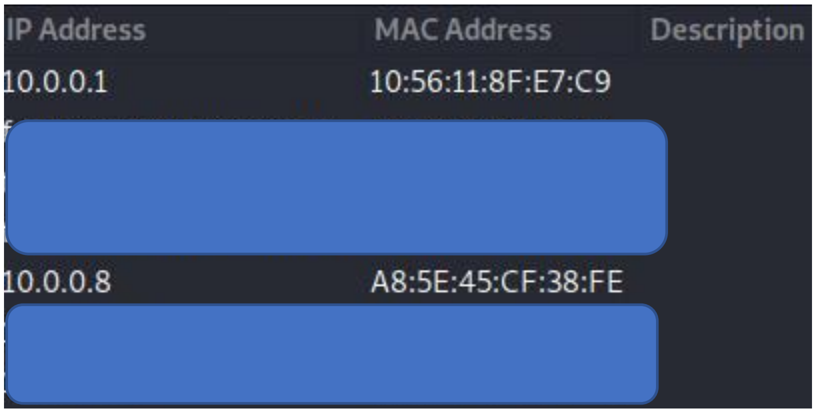
Connected Devices

	Host Name	MAC Address	Connection Type
✓	XingHua-PC	A8:5E:45:CF:38:FE	Ethernet
[redacted]	[redacted]	[redacted]	[redacted]
✓	kali-78	E0:DB:55:82:EB:33	Ethernet

Xinghua-PC is my target machine and kali-78 is my attack machine.

Step 2: Attacker who connects to the network scan hosts in the network

I will use Ettercap to scan my network and find out my target machine and the switch.

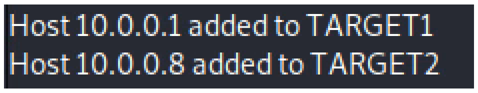


IP Address	MAC Address	Description
10.0.0.1	10:56:11:8F:E7:C9	
[redacted]	[redacted]	
10.0.0.8	A8:5E:45:CF:38:FE	
[redacted]	[redacted]	

After scanning the network, I know 10.0.0.1 is the switch and 10.0.0.8 is my target machine.

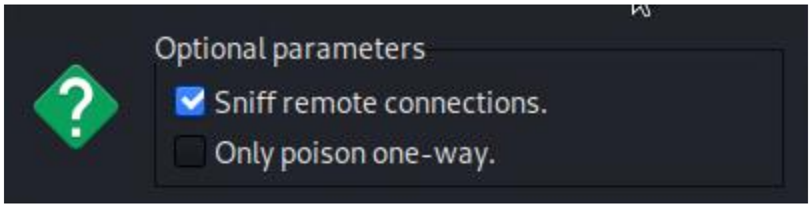
Step 3: Use ARP poisoning to send spoofed ARP messages

I will add my switch, 10.0.0.1, to target 1 and my target machine, 10.0.0.8, to target 2. Therefore, the target machine traffic to the switch will be modified.



```
Host 10.0.0.1 added to TARGET1
Host 10.0.0.8 added to TARGET2
```

Then I will use ARP poisoning to sniff remote connections.



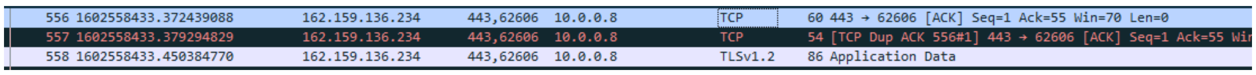
Optional parameters

[✓] Sniff remote connections.

[ ] Only poison one-way.

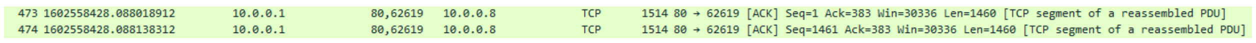
Step 5: Use Wireshark to monitoring all the traffic

By using Wireshark to capture the traffic. I can confirm the ARP poisoning has succeeded because I can see all the communications my target machine, 10.0.0.8, with other servers and my switch, 10.0.0.1.



```
556 1602558433.372439088      162.159.136.234     443,62606  10.0.0.8          TCP       60 443 → 62606 [ACK] Seq=1 Ack=55 Win=70 Len=0
557 1602558433.379294829      162.159.136.234     443,62606  10.0.0.8          TCP       54 [TCP Dup ACK 556#1] 443 → 62606 [ACK] Seq=1 Ack=55 Wir
558 1602558433.450384770      162.159.136.234     443,62606  10.0.0.8          TLSv1.2   86 Application Data
```

Target machine’s traffic with outside internet.

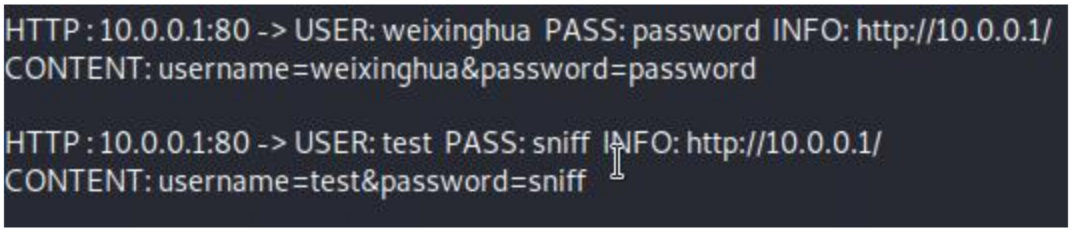


```
473 1602558428.088018912       10.0.0.1             80,62619   10.0.0.8            TCP      1514 80 → 62619 [ACK] Seq=1 Ack=383 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
474 1602558428.088138312       10.0.0.1             80,62619   10.0.0.8            TCP      1514 80 → 62619 [ACK] Seq=1461 Ack=383 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
```

Target machine’s traffic with my switch.

Step 6: Password Sniffing

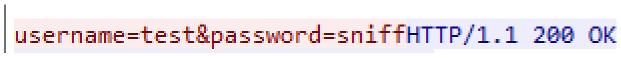
In this exercise, I use 10.0.0.1 as the server. Also, 10.0.0.1 is the website page for the Shaw network login page. So, if I use the target machine 10.0.0.8 and login to the 10.0.0.1 website. I could see its password on the attack machine.



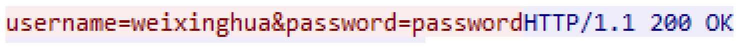
```
HTTP : 10.0.0.1:80 -> USER: weixinghua  PASS: password  INFO: http://10.0.0.1/
CONTENT: username=weixinghua&password=password

HTTP : 10.0.0.1:80 -> USER: test  PASS: sniff  INFO: http://10.0.0.1/
CONTENT: username=test&password=sniff
```

Ettercap detects the username and password input from 10.0.0.8 to 10.0.0.1. Also, I could also check the username and password in Wireshark captures. The difference is that Ettercap will automatically find and filter out the interesting information we want.



```
username=test&password=sniffHTTP/1.1 200 OK
```

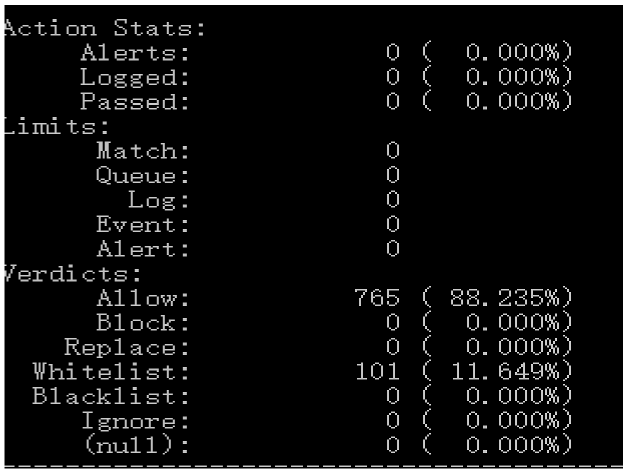


```
username=weixinghua&password=passwordHTTP/1.1 200 OK
```

After following the correct TCP stream, Wireshark also captures usernames and passwords.

Step 7: Checking Snort IDS and Victim Wireshark

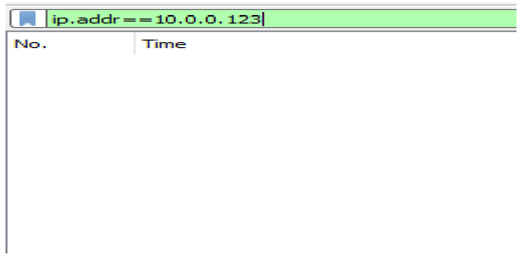
While I was doing password sniffing. I have Snort IDS running on the target machine with sense level sets to low.



```
Action Stats:
     Alerts:              0 (  0.000%)
     Logged:              0 (  0.000%)
     Passed:              0 (  0.000%)
Limits:
      Match:              0
      Queue:              0
        Log:              0
      Event:              0
      Alert:              0
Verdicts:
      Allow:            765 ( 88.235%)
      Block:              0 (  0.000%)
    Replace:              0 (  0.000%)
  Whitelist:            101 ( 11.649%)
  Blacklist:              0 (  0.000%)
     Ignore:              0 (  0.000%)
     (null):              0 (  0.000%)
==========================================
```

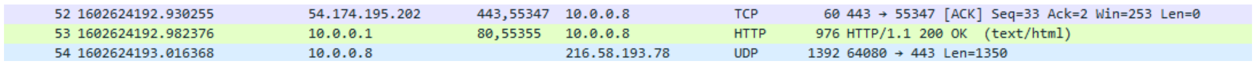
Snort does not catch anything. There are no alerts and no blocks. This means the attack is a success and the target machine can not notify the attack.

My attacker machine IP is 10.0.0.123. After filtering this IP in the victim's Wireshark capture, there is no communication with this IP.



```
ip.addr==10.0.0.123
No.         Time
```

Every communication is either with 10.0.0.1 or other websites.



```
52 1602624192.930255    54.174.195.202   443,55347  10.0.0.8       TCP     60 443 → 55347 [ACK] Seq=33 Ack=2 Win=253 Len=0
53 1602624192.982376    10.0.0.1         80,55355   10.0.0.8       HTTP   976 HTTP/1.1 200 OK  (text/html)
54 1602624193.016368    10.0.0.8                    216.58.193.78  UDP   1392 64080 → 443 Len=1350
```

Conclusion and Prevention

Password sniffing using ARP poisoning attacks occurs on a low level, users targeted by this attack rarely realize that their traffic is being inspected. Even IDS will ignore the attacks.

In order to protect me from password sniffing using ARP poisoning attacks, I need to first understand the spoofing process, so I can look at abnormal activities in my server and determine what information is attacker targeting. Therefore, I could store sensitive information in a safer place. Also, I could create a static ARP to reduce the risk of spoofing. If I have two hosts regularly communicate with each other, setting up a static ARP entry creates a permanent entry in my ARP cache that can help add a layer of protection from spoofing. Moreover, I could use a more powerful, robust IDS tool which dedicates to spot and prevent ARP spoofing.

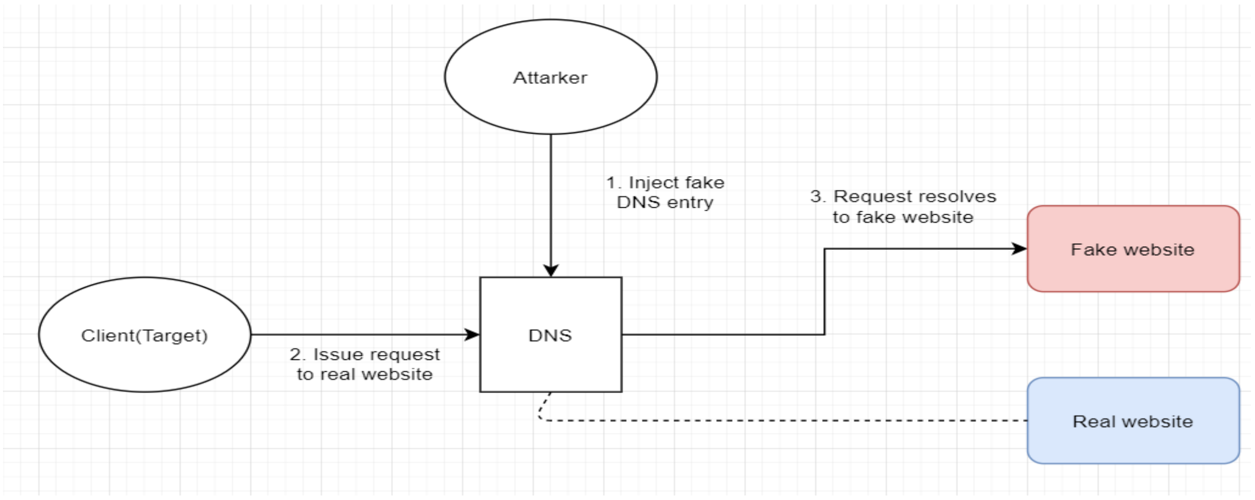
Task 2 – MITM - DNS spoofing attack

Introduction

Domain Name Server spoofing attack, aka DNS spoofing attack, is an attack in which altered DNS records are used to redirect online traffic to a fake website that resembles its intended destination. Often users are prompted to log in to their account which gives attackers the opportunity to steal sensitive information.

A DNS spoofing attack is a man in the middle attack. In this exercise, I will exam and illustrate the DNS cache poisoning attack.

Diagram

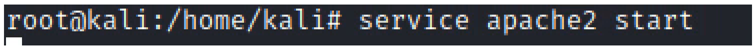


Experiment

In this exercise, the attacker IP is 10.0.0.123 and the victim machine is 10.0.0.8, the switch is 10.0.0.1. The attacker intercepts a communication channel between a target and a server belongs to a website. In this case, I will use a wildcard to redirect every website to the fake website except the ones who have protection on DNS spoofing.

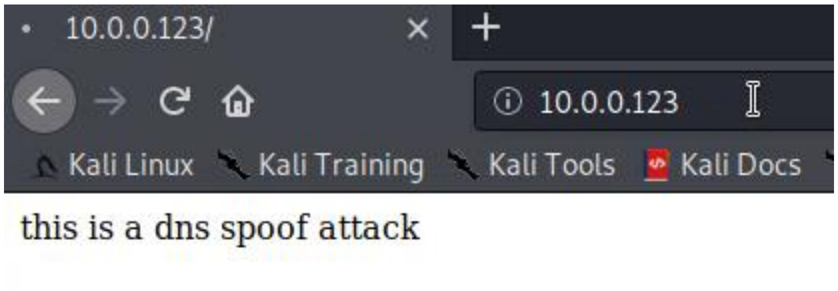
Step 1: Attacker Create and Publish a fake website

The attacker modifies index.html under /var/www/html and start apache server.



```
root@kali:/home/kali# service apache2 start
```

To exam the fake website, open a browser and go to the attacker's IP.



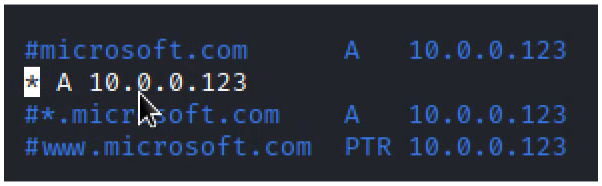
this is a dns spoof attack

This shows that the fake website is published.

Step 2: Attacker enable IP forward and set up redirect website

The attacker modify /proc/sys/net/ipv4/ip_forward to 1. As a result, IP packets sent between the target and server are forwarded to the attacker's machine.

Then attacker modifies/etc/Ettercap/Ettercap.dns to set up the website to be redirected.

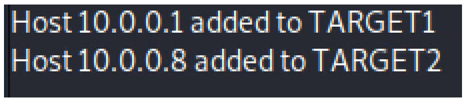


```
#microsoft.com      A   10.0.0.123
* A 10.0.0.123
#*.microsoft.com    A   10.0.0.123
#www.microsoft.com  PTR 10.0.0.123
```

In this exercise, I use wildcard and redirect every website to the fake website.

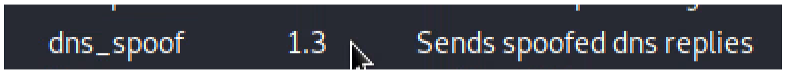
Step 3: Attacker start DNS spoofing

The attacker has connected to the target network and will scan the network hosts to determine the target hosts and server. In this case, the target host will be 10.0.0.8 and the server will be 10.0.0.1.

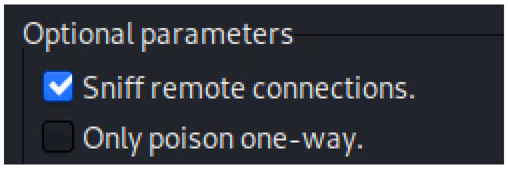


```
Host 10.0.0.1 added to TARGET1
Host 10.0.0.8 added to TARGET2
```

Then the attacker chooses the DNS spoof plugin in Ettercap and perform an ARP poisoning attack.



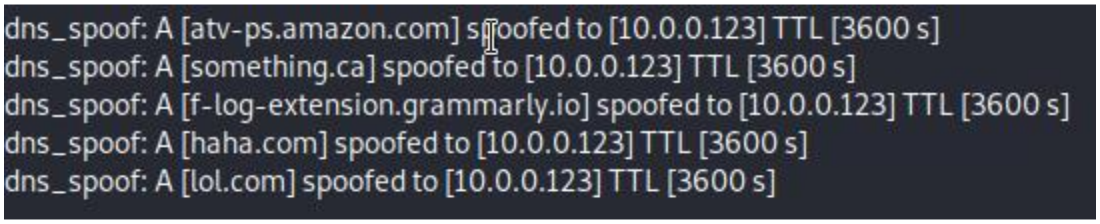
```
dns_spoof        1.3      Sends spoofed dns replies
```



Optional parameters
- [✓] Sniff remote connections.
- [ ] Only poison one-way.

Step 4: Attacker checks attacking the result

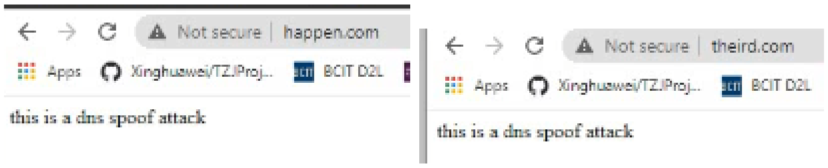
Once the attack succeeds, Ettercap will show which website has been redirected to the target machine's website.



```
dns_spoof: A [atv-ps.amazon.com] spoofed to [10.0.0.123] TTL [3600 s]
dns_spoof: A [something.ca] spoofed to [10.0.0.123] TTL [3600 s]
dns_spoof: A [f-log-extension.grammarly.io] spoofed to [10.0.0.123] TTL [3600 s]
dns_spoof: A [haha.com] spoofed to [10.0.0.123] TTL [3600 s]
dns_spoof: A [lol.com] spoofed to [10.0.0.123] TTL [3600 s]
```

Because I used wildcard, even browser plugin requests will be redirected to my fake website.

On the victim machine side, every website will be redirected to the fake website.



Not secure | happen.com

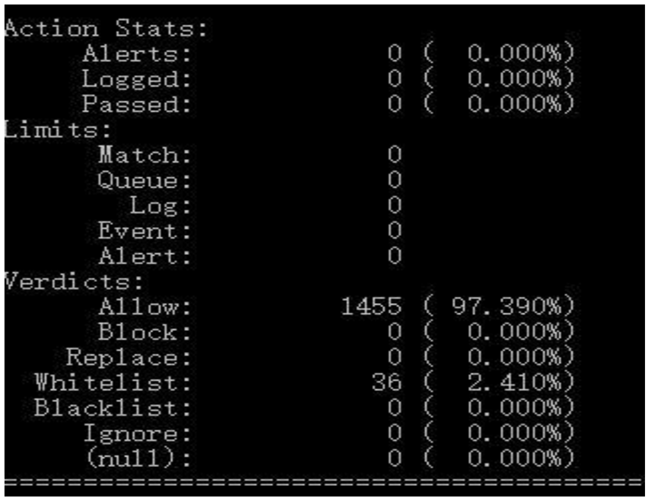
this is a dns spoof attack

Not secure | theird.com

this is a dns spoof attack

Step 5: Checking IDS and Wireshark

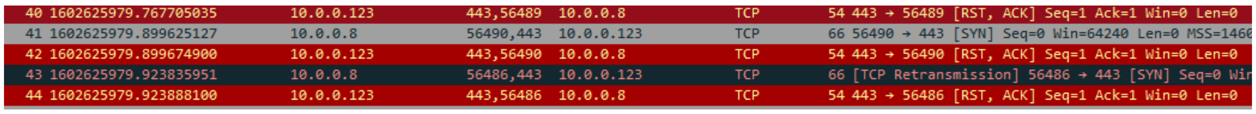
While I was doing a DNS spoofing attack. I have Snort IDS running on the target machine with sense level sets too low.



```
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:            0 (  0.000%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:         1455 ( 97.390%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:           36 (  2.410%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
     (null):            0 (  0.000%)
========================================
```

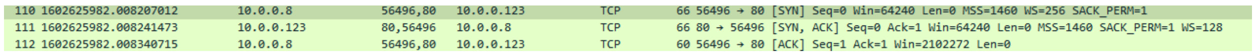
Snort does not catch anything. There are no alerts and no blocks. This means the attack is a success and the target machine can not notify the attack.

On the attacker side, we can tell from the Wireshark that communications are redirected to 10.0.0.123.



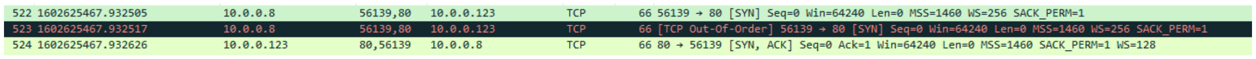
```
40 1602625979.767705035    10.0.0.123    443,56489  10.0.0.8      TCP    54 443 → 56489 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
41 1602625979.899625127    10.0.0.8      56490,443  10.0.0.123    TCP    66 56490 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
42 1602625979.899674900    10.0.0.123    443,56490  10.0.0.8      TCP    54 443 → 56490 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
43 1602625979.923835951    10.0.0.8      56486,443  10.0.0.123    TCP    66 [TCP Retransmission] 56486 → 443 [SYN] Seq=0 Wi
44 1602625979.923888100    10.0.0.123    443,56486  10.0.0.8      TCP    54 443 → 56486 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

Below shows a 3-way handshake from the target machine 10.0.0.8 to attack machine 10.0.0.123.



```
110 1602625982.008207012    10.0.0.8      56496,80   10.0.0.123    TCP    66 56496 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
111 1602625982.008241473    10.0.0.123    80,56496   10.0.0.8      TCP    66 80 → 56496 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
112 1602625982.008340715    10.0.0.8      56496,80   10.0.0.123    TCP    60 56496 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
```

On the target side, a 3-way handshake also happens.

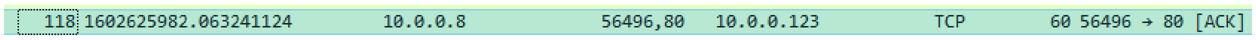


```
522 1602625467.932505    10.0.0.8      56139,80   10.0.0.123    TCP    66 56139 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
523 1602625467.932517    10.0.0.8      56139,80   10.0.0.123    TCP    66 [TCP Out-Of-Order] 56139 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
524 1602625467.932626    10.0.0.123    80,56139   10.0.0.8      TCP    66 80 → 56139 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
```

Conclusion and Prevention

A DNS spoofing attack needs a target. This could be an authorized name server or a weak point on the system DNS cache. The attacker can adjust the cache of that DNS server and begin redirect traffic from the actual website to the spoofed website. This could lead to password leaks, social engineering attacks and etc.

To prevent a DNS spoofing attack, we could filter our DNS server that does not allow the DNS server to answer traffic from high ports. For example, this traffic to 10.0.0.123 is sending from a high port.



```
118 1602625982.063241124    10.0.0.8    56496,80   10.0.0.123    TCP    60 56496 → 80 [ACK]
```

I could decrease the TTL values on my local DNS server. Notice that my default TTL value of the DNS server is 3600 seconds. I could decrease it to 600 seconds. In this way, if a cache poison does attack me, it will only bother me for a short duration of time and will not influence my network performance.

Also, use products like Zone, Stub Zone to enable my DNS server to resolve records in another domine. The information in the product allows my DNS to contact the authoritative DNS server directly. So commonly accessed domains would not be easily compromised.

Task 3 – URL Capture

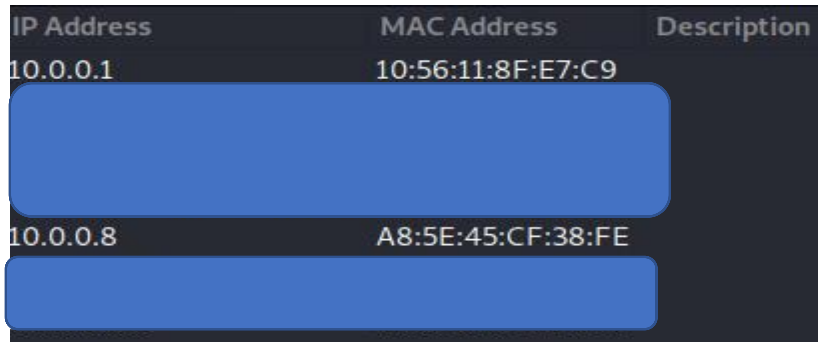
Introduction

URL capture is a MITM attack. It captures HTTP requests and logs the URL being referenced. In this exercise, I will use urlsnarf which outputs all requests URLs sniffed from HTTP traffic.

Experiment

Step 1: Attacker who connects to the network scan hosts in the network

I will use Ettercap to scan my network and find out my target machine and the switch.

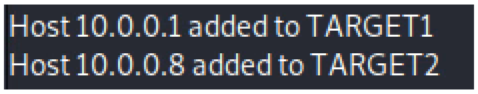


IP Address	MAC Address	Description
10.0.0.1	10:56:11:8F:E7:C9	
[redacted]		
10.0.0.8	A8:5E:45:CF:38:FE	
[redacted]		

After scanning the network, I know 10.0.0.1 is the switch and 10.0.0.8 is my target machine.

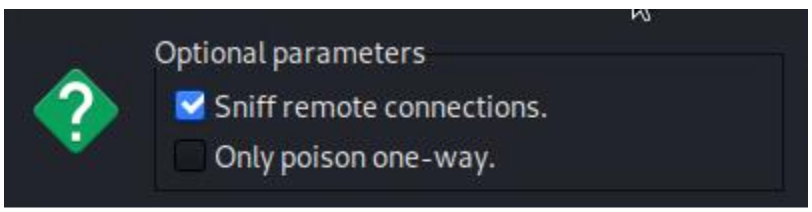
Step 2: Use ARP poisoning to send spoofed ARP messages

I will add my switch, 10.0.0.1, to target 1 and my target machine, 10.0.0.8, to target 2. Therefore, the target machine traffic to the switch will be modified.



```
Host 10.0.0.1 added to TARGET1
Host 10.0.0.8 added to TARGET2
```

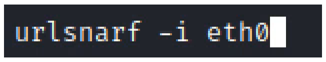
Then I will use ARP poisoning to sniff remote connections.



Optional parameters
[✓] Sniff remote connections.
[ ] Only poison one-way.

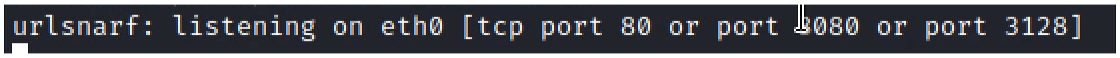
Step 3: Start URL capture

I will set urlsnarf listening to the same Lan where the target machine and server are.

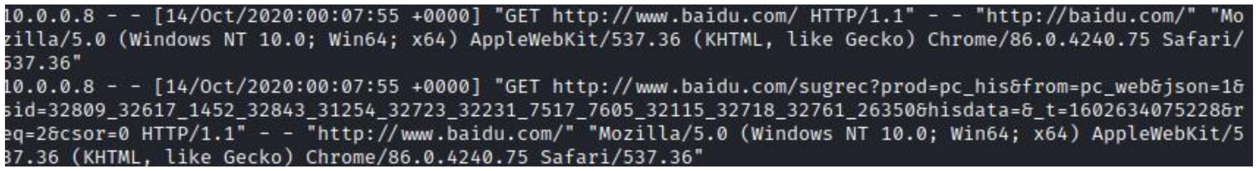


```
urlsnarf -i eth0
```

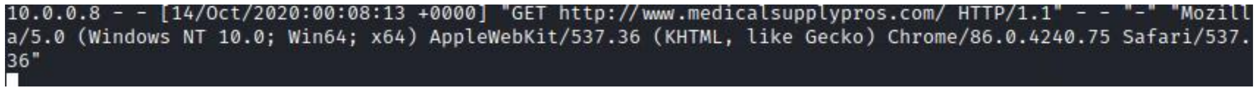
Urlsnarf shows that it is monitoring ports 80, 8080 and 3128. Therefore, I am going to access website pages using port 80 web design.



```
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]
```



```
10.0.0.8 - - [14/Oct/2020:00:07:55 +0000] "GET http://www.baidu.com/ HTTP/1.1" - - "http://baidu.com/" "Mo
zilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/
537.36"
10.0.0.8 - - [14/Oct/2020:00:07:55 +0000] "GET http://www.baidu.com/sugrec?prod=pc_his&from=pc_web&json=1&
sid=32809_32617_1452_32843_31254_32723_32231_7517_7605_32115_32718_32761_26350&hisdata=&_t=1602634075228&r
eq=2&csor=0 HTTP/1.1" - - "http://www.baidu.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5
37.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36"
```

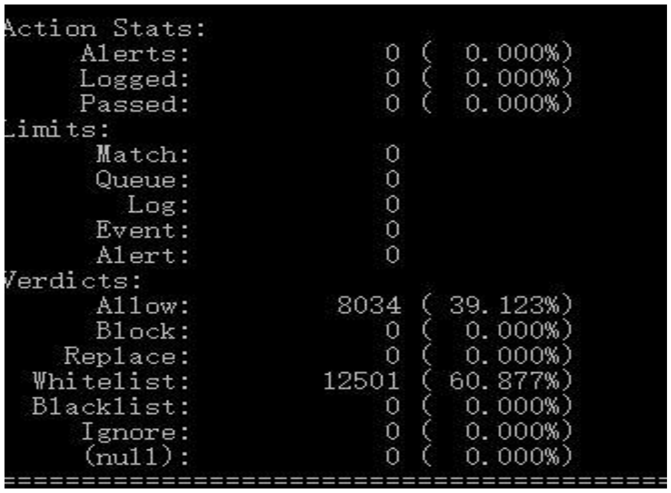


```
10.0.0.8 - - [14/Oct/2020:00:08:13 +0000] "GET http://www.medicalsupplypros.com/ HTTP/1.1" - - "-" "Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.
36"
```

Urlsnarf shows the websites the victim accessed. It also shows the target machine system, browser the target machine using, website IPv4 address.

Step 5: Checking IDS and Wireshark

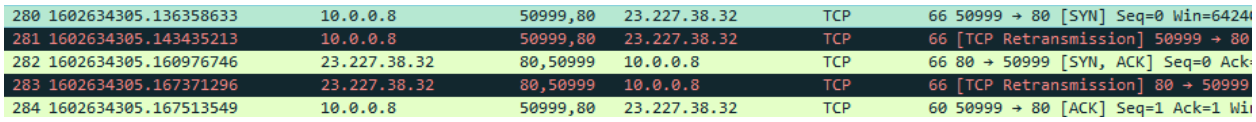
While I was doing URL capture. I have Snort IDS running on the target machine with sense level sets to low.



```
Action Stats:
     Alerts:           0 (  0.000%)
     Logged:           0 (  0.000%)
     Passed:           0 (  0.000%)
Limits:
      Match:           0
      Queue:           0
        Log:           0
      Event:           0
      Alert:           0
Verdicts:
      Allow:        8034 ( 39.123%)
      Block:           0 (  0.000%)
    Replace:           0 (  0.000%)
  Whitelist:       12501 ( 60.877%)
  Blacklist:           0 (  0.000%)
     Ignore:           0 (  0.000%)
     (null):           0 (  0.000%)
=========================================
```

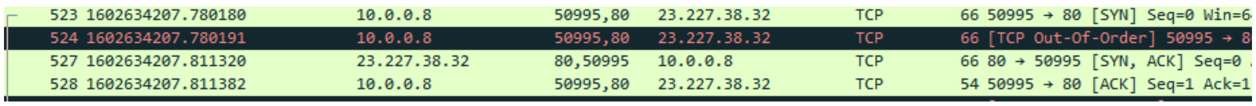
Snort does not catch anything. There are no alerts and no blocks. This means the attack is a success and the target machine can not notify the attack.

On the attacker side, we can tell from the Wireshark that communications are exchanged between websites and target machines.



```
280 1602634305.136358633   10.0.0.8       50999,80   23.227.38.32   TCP   66 50999 → 80 [SYN] Seq=0 Win=6424
281 1602634305.143435213   10.0.0.8       50999,80   23.227.38.32   TCP   66 [TCP Retransmission] 50999 → 80
282 1602634305.160976746   23.227.38.32   80,50999   10.0.0.8       TCP   66 80 → 50999 [SYN, ACK] Seq=0 Ack
283 1602634305.167371296   23.227.38.32   80,50999   10.0.0.8       TCP   66 [TCP Retransmission] 80 → 50999
284 1602634305.167513549   10.0.0.8       50999,80   23.227.38.32   TCP   60 50999 → 80 [ACK] Seq=1 Ack=1 Wi
```

Below shows a 3-way handshake from the target machine 10.0.0.8 to another website.



```
523 1602634207.780180   10.0.0.8       50995,80   23.227.38.32   TCP   66 50995 → 80 [SYN] Seq=0 Win=6
524 1602634207.780191   10.0.0.8       50995,80   23.227.38.32   TCP   66 [TCP Out-Of-Order] 50995 → 8
527 1602634207.811320   23.227.38.32   80,50995   10.0.0.8       TCP   66 80 → 50995 [SYN, ACK] Seq=0
528 1602634207.811382   10.0.0.8       50995,80   23.227.38.32   TCP   54 50995 → 80 [ACK] Seq=1 Ack=1
```

Conclusion

URL capture attack is useful that an attacker could monitor the target's website traffic. Then collect information about the websites the target commonly access. Then the attacker could also perform a DNS spoofing attack to extract sensitive information or perform other attacks.

To prevent URL capture and such man in the middle attack. I could set up a virtual private network. VPNs can be used to create a secure environment for sensitive information within a local area network. It uses key-based encryption to create a subnet for secure communication. Even if an attacker connects to the network, he will not be able to decode the traffic in the VPN.

I could also use HTTPS instead of HTTP. HTTPS can be used to securely communicate over HTTP using public-private key exchange. This prevents an attacker from data sniffing. And websites should only HTTPS and not provide HTTP choice.