Program Tested on LAN 192.168.10/24

The two major parts to this final project are the ARP spoofer itself and the ARP spoofing detector. The ARP spoofing detector is broken down into 5 files.

**Main.py :** This file serves as the program entry. From the main file we launch 3 threads to perform work for the program.

Thread 1 will be the sniffer, we are capturing packets on a live network. So, we need a thread constantly listening on the wire for a specific ICMP message

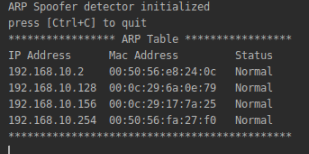
Thread 2 will be the dissector, we need to take a part the message we received as quickly as it is intercepted to then extract source/destination IP-address/MAC-address.

Thread 3 will be the detector, we need to employ the logic behind what constitutes the integrity of the address resolution protocol for IP addresses on the network. This is where we determine if duplicates are found in the IP/Mac table and determine a fan-out rate for a threshold set number of ARP requests per unit time.

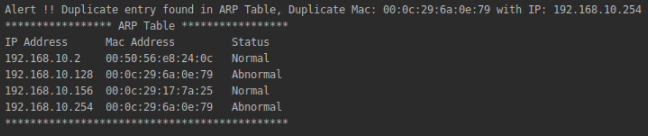
**Dissector.py:** This is the file which takes care of actually parsing the packet that was intercepted over the LAN. We need to first determine the protocol type for ipv4/ipv6 byte formatting. Then we unpack the following bytes to obtain the source ip address, destination ip address, source mac address and destination mac address. These items are all crucial components of the message that will play into our ARP detector logic. The current time is also logged for when the packet was first dissected, close enough to intercept time. This group of information for the packet is then bundled up into a tuple and placed into a shared queue amongst the threads. The queue is used so that, an intercepted message could be passed to the detector thread to be logically parsed on the basis of ARP spoofing.

**Detector.py :** This file is the heavy lifter for the program where the actual logic behind detecting ARP spoofing in the LAN. This portion of the program runs on its own thread to constantly check the message queue for newly intercepted messages. This thread first gets a baseline for what IP-addresses are on the network by sending out an ARP request to each IP on the LAN. This allows the program to collect the Mac Address for what is in the ARP table. The program then makes it first pass and scan of the newly created ARP table for a duplicate mac address. Since one of the major signs of ARP spoofing involves an attacker hijacking a user’s Mac address and saying that it belongs to them. So, if a duplicate is found in the table then the program will send an alert message to STD-OUT and update the table accordingly.

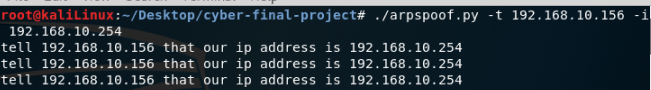
Below is an example of what the custom ARP table looks like when no spoofing is detected in the network. The three main portions to the table are the 1. IP address column, 2. Mac address column and Status Column. The first two columns are self-explanatory, but the status column is a Boolean condition for if the IP/Mac pairing is considered abnormal or abnormal. For an abnormal condition to arise there would have to be a duplicate entry in the table or a fan-out rate that has exceeded the threshold we placed upon it.



The photo below shows an IP/Mac pairing that was deemed abnormal. If you analyze the photo a little further, you will see that the status value for ip address 192.168.10.128 and 192.168.10.254 was triggered as abnormal since these two ip addresses have the same mac address and produce a duplicate in the table.



What’s going on behind the scene is that on the separate ARP spoofer program running on a separate node in the network, we are sending the ICMP message to tell 192.168.10.156 which is the machine that the spoofer detector is running on that our ip address associated with this mac address is 192.168.10.254. This makes an entry in the victims ARP table with the duplicate entry. Our program then parses the table checking for this duplicate.



Also, from the photo above you can get an idea of why we need to check for fan-out rate. In order for our spoofer to work, we would need to send the same message across the network saying, “hey you this is me” or “hey Ip 192.168.10.156 I am IP 192.168.10.254”. Or else the table would revert back to whatever mac address entry actual belongs to that IP-address or nothing if it is not on the LAN.

Fan out rate was then deemed useful since this could be a potential sign of ARP spoofing. This made sense for the research we did into the assignment and also for a countermeasure to what we had to reproduce in the spoofer itself. The Fan out rate calculating function simply steps through every unique IP to IP ARP message intercepted and compares each timestamp associated with the message to determine the threshold. So, for example say a threshold across a time span of 5 minutes with 10 occurrences. So, if we intercept the same IP – IP ARP message 10 times within a 5-minute period than we raise an alert.

**Sniffer.py :** This is a pretty simple file in which we actually listen on the socket created and intercept all packets within the LAN. Each packet is then placed inside a queue that is shared amongst the threads. So, we can now examine the packet intercepted and perform detecting logic with the packet.

**Psocket.py:** This is a pretty simple file in which we simply set up a socket to listen on.

**Exec Instruction:** To execute the program you would navigate inside the project directory to the detector folder. The program takes two command line arguments both arguments deal with determining the fan-out-rate.

*Python3 main.py arg1 arg2*

*Arg 1* – Time Range we are considering, so a value of 300 would be a 5-minute window in which we a looking for a threshold number of ARP requests between the same Source and destination IP-Address.

*Arg 2* – This would be the threshold for the number of requests, so a value of 10 would be 10 ARP requests have been detected across the time range passed in through Arg 1.

**Runtime Instructions :** No specific runtime instructions exist. The program simply prints to STD-OUT our custom ARP table and a separate alert message for when a threshold has been broken and a new suspicious IP-MAC pairing has been discovered. To exit program simply enter <CTRL+C>.