Playing with ARP - DRAFT
I repeat: It's a draft

Thomas Habets*

March 30, 2007

Contents

* *thomas@habets.pp.se*

1 Introduction

1.1 How ARP works

ARP, or Address Resolution Protocol [RFC826], is a protocol which is used to get the 48bit Ethernet address for a NIC (Network Interface Card) given an address for a higher level protocol, such as IPv4 [RFC791] (will hereby be referred to as IP)[1]. This is not a bit-counting specification, see the references for those.

To summarize the simple and traditional network, ARP for IP over Ethernet, here is a scenario:

- S has IP address 10.0.0.1 and MAC 08:00:00:00:00:01
- T has IP address 10.0.0.2 and MAC 08:00:00:00:00:02
- S wants to send T an IP packet.
- S and T are on the same IP subnet.
- Since S and T are on the same IP subnet, they are also on the same Ethernet segment[2]

1. S → broadcast: "who-has" 10.0.0.2, tell 10.0.0.1 (08:00:00:00:00:01)
2. S ← T: 10.0.0.2 "is-at" 08:00:00:00:00:02
3. S → T: IPv4 packet

1.2 Switches

Switches, or "smart hubs" as they are sometimes called, also cache the IP address and its MAC address. But unlike computers, switches don't normally send out ARP packets, instead they record which MACs lie behind each port. If there is no record of a MAC in the switch' memory, it broadcasts the packet to all ports.

[1] It can be used for other protocol pair as well, but I will not go into that
[2] Yeah yeah, I know. But for now, that's not relevant.

2 The nice things

2.1 ARP ping

Sometimes you want to see if a host is up, and for that reason, there is the ping utility. However, sometimes ping can't work. For example you don't have an IP address yet, or the target you are pinging is blocking every packet you can think of (yes, some people block 'ping'). What you'd *really* want is an Ethernet ping, but no one has thought of a way to do that (and I'd be very surprised if it were possible at all[3]).

What you do is simply send a who-has ARP packet asking what Ethernet address the IP you want to ping has. You'll get an answer if the host is up.

If you don't want people to do these things on your network, you'll have to hard-code Ethernet addresses on the hosts involved and block ARP packets with firewall software.

To do this to a Linux server with a bunch of clients you'll have to explicitly add the MAC/IP address pairs on each host that needs to communicate. So each computer will have to run[4]:

```
# ip ne change 10.0.0.1 dev eth0 lladdr 08:00:00:00:00:01
```

Where the address pair is that of the host that it wants to communicate with. For client/server-only communication with one server this will only mean one pair on each client, and one pair per client on the server. This is not that much but for a fully connected network (the norm) every host on the network will have to have the address pair of every other host hard-coded in its cache.

Once the ARP tables are set up properly, every host should run[5]

```
# ip link set eth0 arp off
```

to disable ARP, and thereby ARP pings.

I would like to point out that this is ugly and horrible. I would hate to admin this network. Let's say one NIC breaks on one host. This would require updating every other host with the new MAC. Well, either that or changing the MAC on the NIC, which is annoying at best and sometimes pseudo-impossible[6].

ARPing is a program I made to make these ARP pings. This is how it looks when I ping my home firewall:

```
# arping -c 1 piggy
ARPING 192.168.0.1
60 bytes from 00:60:97:43:91:88 (192.168.0.1):
                                index=0 time=531.912 usec

--- 192.168.0.1 statistics ---
1 packets transmitted, 1 packets received, 0% unanswered
```

And this is what tcpdump sees:

[3] If you've heard of a way, please enlighten me.

[4] Since I'm most familiar with Linux, every command in this document will be for Linux, unless explicitly stated otherwise.

[5] Again, this command will only work in Linux.

[6] If the NIC doesn't allow changing MAC, you can hack the OS kernel into thinking that it has the old MAC and run the NIC in promiscuous mode. But apparently not all NICs can be put in promiscuous mode. And besides, promiscuous mode puts more strain on the CPU.

```
# tcpdump -nec 2 arp
tcpdump:  listening on eth0
19:03:19.725338 0:10:a5:1e:a5:c5 ff:ff:ff:ff:ff:ff 0806 42:
                arp who-has 192.168.0.1 tell 192.168.0.2
19:03:19.725900 0:60:97:43:91:88 0:10:a5:1e:a5:c5 0806 60:
                arp reply 192.168.0.1 is-at 0:60:97:43:91:88
```

ARPing can be obtained from `http://www.habets.pp.se/synscan/` and is included in OpenBSD, FreeBSD and Debian GNU/Linux but works on many more systems. Practical use information about it can be found in the README file that comes with ARPing.

2.2 Reverse ARP ping

Although it technically doesn't involve the ARP protocol, I thought it'd be nice to put it here. If that doesn't suit you, then it's my doc and I'll put anything I want in it, so there.

Reverse ARP ping[7] is used to find the IP address that belongs to a NIC with a particular Ethernet address. One way to do this is with RARP [RFC903] (Reverse ARP) but in the real world, RARP is rarely used. And it's also not suited for our needs since it depends on a central server that knows about all address pairs. This makes it not a ping, since it would be the RARP server that responds, not the target host.

One way (the best way I've found) to ping an Ethernet address directly is by sending a directed broadcast ping to it. This means a normal IP ICMP ping to the global IP broadcast address (255.255.255.255), except that the target Ethernet address is not set to the broadcast Ethernet address (ff:ff:ff:ff:ff:ff) but the target Ethernet address.

This of course depends on the target to reply to a broadcast ping, something not all operating systems do, not by default anyway. Most UNIX dialects seem to reply to them, but Windows doesn't. Most operating systems should be able to turn this functionality on and off. With Linux you turn it off by setting /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts to 1.

If the target doesn't reply to broadcast pings you can try and guess the subnet of the target. For example if you think the target may be on the network 192.168.0.0/24 you can try to use the target IP address 192.168.0.255[8] instead. If that doesn't work either, you can brute-force all probable (or all possible) addresses until you get a reply.

Some example runs:

```
#  arping -c 1 00:60:97:43:91:88
ARPING 00:60:97:43:91:88
60 bytes from 192.168.0.1 (00:60:97:43:91:88):
                          icmp_seq=0 time=1.847 msec

--- 00:60:97:43:91:88 statistics ---
1 packets transmitted, 1 packets received, 0% unanswered
```

[7] Bad and misleading name, I know. I'm open to suggestions.

[8] Usually the broadcast address has all ones for the host bits, so let's just assume that, for now.

```
#  arping -c 1 -T 192.168.0.255 00:60:97:43:91:88
ARPING 00:60:97:43:91:88
60 bytes from 192.168.0.1 (00:60:97:43:91:88):
                          icmp_seq=0 time=1.490 msec

--- 00:60:97:43:91:88 statistics ---
1 packets transmitted, 1 packets received, 0% unanswered
```

The corresponding tcpdumps are:

```
#  tcpdump -nec 2 'icmp'
tcpdump:  listening on eth0
19:46:20.357208 0:10:a5:1e:a5:c5 0:60:97:43:91:88 0800 42:
                192.168.0.2 > 255.255.255.255:
                icmp:  echo request
19:46:20.358275 0:60:97:43:91:88 0:10:a5:1e:a5:c5 0800 60:
                192.168.0.1 > 192.168.0.2:
                icmp:  echo reply
```

And:

```
#  tcpdump -nec 2 'icmp'
tcpdump:  listening on eth0
23:59:22.424199 0:10:a5:1e:a5:c5 0:60:97:43:91:88 0800 42:
                192.168.0.2 > 192.168.0.255:
                icmp:  echo request
23:59:22.425337 0:60:97:43:91:88 0:10:a5:1e:a5:c5 0800 60:
                192.168.0.1 > 192.168.0.2:
                icmp:  echo reply
```

3 The naughty things

3.1 ARP poisoning

By sending ARP replies ("is-at") for a question that was never asked you can get other boxen to send data to you that was supposed to be sent elsewhere. Dsniff[**?**] contains a program called "arpspoof" that does this.

By relaying the traffic through your box you can sniff the traffic, and also modify it. Combined with other programs in the dsniff package it's quite nasty.

It can be prevented by locking MAC addresses to ports on the switch.

This ofcourse is the packet. It looks innocent, but it's very naughty. :)

```
#  tcpdump -nec 1 'arp'
tcpdump:  listening on eth0
12:44:50.873834 0:10:a5:1e:a5:c5 ff:ff:ff:ff:ff:ff 0806 60:
                arp reply 192.168.42.1 is-at 0:10:a5:1e:a5:c5
```

3.2 Switch flooding

A switch learns how the network looks by looking at the source MAC addresses of the frames coming to it on different ports. If a box sends a frame coming in to the switch on port N, then the switch knows that when it needs to send a frame to that box, it just has to send that frame out on port N, instead of all ports.

But the switch doesn't have infinite memory. If you send lots and lots of frames into the switch, with lots and lots of different source MAC addresses, then the switch will have to compromise. Different switches do different things. Some just shut down the flooding port, others shut down their switching capability, effectively turning them into hubs, which are much easier to sniff on. Also if the switch is turned into a hub then ARP poisoning can be used against the hosts themselves. Very sneaky.

3.3 Covert communication

It's almost silly to have a section for covert communication since there are so many ways to do it. Pretty much all protocols have a means for sending data [9] in innocent-looking packets. And since ARP is not routed outside an Ethernet segment it's pretty useless anyway.

Although it would be interresting to see an implementation of tunneling IP over a covert ARP network...

Anyway, ARP who-has packets have a field (arp.arp_tha) which for Ethernet contains 6 bytes of "don't-care" data. RARP request packets have a similar field.

[9] Even excluding covert channels using only the "was a packet received or not" timing-like covert channels that modulate a stream of innocent-looking packets.

References

[RFC826] ARP, `http://rfc826.x42.com`

[RFC791] IPv4, `http://rfc791.x42.com`

[RFC903] RARP, `http://rfc903.x42.com`

[DSNIFF] Programs for sniffing switched (and unswitched) ethernet networks, as well as doing man-in-the-middle attacks.
`http://monkey.org/~dugsong/dsniff/`