“Real World ARP Spoofing”

Raúl Siles

August, 2003

*GIAC Certified **Incident Handler** (**GCIH**) Practical*
*(Version **2.1a** – Option 1: **Exploit in Action**)*

GIAC Certification Administrivia Version 2.5b

Abstract

This paper pretends to explore ARP, from its design and specifications point of view, the Internet RFCs, to its real world implementations, that is, how the operating systems analyzed behave. It explains how when dealing with ARP works, how to manipulate and configure the elements that constitute the ARP modules inside the TCP/IP stacks of different OS and how the protocol can be exposed from a security perspective.

It describes the security vulnerabilities that could be exploited using ARP to take control over the network traffic that flows between two systems in a Local Area Network, called "ARP spoofing or poisoning", redirecting the traffic to a box owned by an attacker, and proposing some of the different advanced attacks that could be developed based on it.

The goal of this paper is trying to research and discover every small detail and component of the ARP protocol that will allow an attacker to get control over an unauthorized system, and to provide enough information for an administrator to be able to protect its network infrastructure.

The main motivation for this paper's research was originated after more than two years of internal Penetration Testing over production environments, meaning by internal the situation where the security auditor plays the attacker's role as an insider: employee, subcontractor, third-party support engineer or consultant…

Although the "ARP spoofing" technique is very simple in concept, in real world situations over heterogeneous networks, the obtained results are not always as expected, because both the operating system and network topology influences the way ARP behaves. Therefore, more information about how the ARP protocol and the "ARP spoofing" attack work should be obtained to be able to have as much control as possible over the ARP redirection games.

Layer 2 vulnerabilities are typically underestimated because they are associated with the attacker physically located next to the target system, but this is an incorrect approach. Once an attacker has got control over a system from outside, he is in the same situation as any insider.

From the author's point of view, it is a must to understand every detail about how the ARP protocol and every implementation work and to play the potential role of an attacker to be prepared to defend the network against the different ARP attacks and their security vulnerabilities. For this reason sometimes this paper will analyze a specific aspect to reach the attacker's goal, and sometimes it will focus on defending against the protocol exploitation.

Due to the fact that this is a very ambitious project, it will evolve and go into a deeper research of some areas in future versions, as, for example, covering additional operating systems and network traffic situations, such as those based on high availability solutions. The final goal will be to reach a similar work as the

one developed by Ofir Arkin about the ICMP protocol [OFIR1] but focusing on the ARP protocol. Sorry for being so ambitious, but using Ofir's paper as a reference is well worth.

This paper pretends to be the foundation of a future project called "**The SARP**: **The S**ecurity **A**RP **R**esearch **P**roject". My willing is to make this project available for the Internet community in the next few months.

To be able to agglutinate a huge knowledge around the ARP protocol, the Internet community should share information, so the new proposed ARP project could be a knowledge repository. Its main goal will be offering a database of the different ARP behaviours classified by OS. In the past there were similar projects, covering nearest information security areas, but they were unsuccessful [SSP1].

Some areas this project should include would be:

- Packet taxonomy: stimulus-response ARP traffic or how different OS respond to every possible ARP packet and how their ARP tables are populated, including big anomalies in packets.
- ARP table timeout behaviour: how each ARP timer work and how to configure it through OS kernel parameters.
- ARP bootstrap and shutdown times analysis.
- ARP behaviour when activating/deactivating network interfaces.
- ARP operating system fingerprinting.

Acknowledgements

"Mónica, there are no words to be able to express my feelings about you. Thanks so much for your support and help, and for reviewing this paper ;-)"

"To you, mum, to overcome any problem in this life with your energy and vitality"

"Marta, Jorge, David, thanks for your valuable contribution"

Revision

First version: **1.0** **August, 2003** – Author: Raúl Siles
Originally created for the SANS GIAC Practical paper needed to obtain the GCIH certification.

Table of Contents

Security disclaimer:

In most environments, before trying any of the practical research and tests proposed in this paper you must have written authorization to operate and play ARP games over the tested systems and network. The same kind of authorization must be obtained to use the sniffing techniques required to analyze the ARP protocol behaviour. For this reason it is recommended to have your own network and configure an isolated lab for all the proposed tests.

The author is not responsible at all about the consequences of emulating the actions described in this paper, so use them with caution and at your own risk.

Research disclaimer:

The test results obtained are not only influenced by operating system kernel version and patch level, they are also software and configuration dependent. Based on the applications installed, the system would try to contact other systems in local or remote networks, being needed to route network traffic through its default gateway and therefore using the ARP protocol when not expected.

All these elements will determine the system behaviour and the accuracy of the results, although all reasonable efforts have been made to minimize these factors as much as possible.

GIAC certification practical disclaimer:

Due to the extensive research associated to this paper, the structure proposed by the GIAC GCIH practical version 2.1a, "Exploit in Action", has been slightly modified to accommodate the research steps, a detail description of the tests made, the obtained results and the leading conclusions.

Besides, due to the fact that this document doesn't describe an actual incident in what I took part, the main focus has been centred in Part 2. Therefore, the Incident Handling Process, Part 3, will describe some recommendations that could be used to deal with the attack.

Part 1 – The Exploit

Name

The term "ARP poisoning" refers to an attack based on being able to introduce new information, ARP entries, in the ARP cache table of a given system, that is, poison its ARP table. From the ARP table point of view, this is the right term. Sometimes, the term "ARP spoofing" is used to refer to the same type of attack. At the ARP level, the attacker has spoofed the MAC addresses referenced inside the ARP packet, pretending to confuse other systems as if these packets had been sent by a different system. So, both terms can be used indifferently.

There are different security attacks based on the ARP protocol, Address Resolution Protocol, from now on ARP, and frequently the terms used to refer to them are confused or misused. First of all, we will focus in the one associated to this paper but all them are explained in the "Variants" section.

The idea behind this attack, "ARP spoofing or poisoning", is forcing a system to believe that the host I want to represent has a different MAC address. This MAC address can be the real attacker's MAC address or a fake one (owned by another host currently down, or just an inexistent one). So it is possible to mix both concepts or types of attacks: poisoning another system ARP table with a spoofed MAC address, not the real one.

This paper will focus on the "ARP poisoning/spoofing" attack, but sometimes "MAC spoofing" methods (see "Variants" section) will be referenced to complement the whole analysis.

As a curiosity, for some strange reason, let's call it "*the Google wave effect*", the term "ARP poisoning" is being associated nowadays more to wireless networks (802.11) while the term "ARP spoofing" is not. As it will be analyzed, this type of attack affects both network types, wired and wireless.

There are no official references, as CVE numbers [MITRE1] or CERT ids [CERT1], for this type of attack because this is a generic vulnerability/exploit based on the protocol design, as other similar networking attacks: IP spoofing, TCP session hijacking, network sniffing…

Operating Systems

The vulnerabilities shown are inherent to the ARP protocol design, so they are associated to any given implementation of this protocol. Every single device implementing the ARP protocol, typically included in every TCP/IP stack because it is needed to be able to map layer 3 protocol addresses (IP) to layer 2 ones (Ethernet), is potentially exposed to this attack.

Due to the extended use of the TCP/IP protocols in nowadays data communication networks, associated to the Internet boom, and considering Ethernet, and its variations like Fast or Giga Ethernet, as the most common technologies for LANs (Local Area Networks), almost every computer system needs to speak ARP to be able to communicate with the rest of the world.

This paper tries to develop a deep analysis of the ARP protocol behaviour in the most commonly used operating systems, covering (in alphabetical order):

- Cisco IOS 12.0(2)XC2 router and IOS 11.2(8)SA5 switch.
- HP-UX 10.20, 11.00 and 11i.
- Linux: kernel 2.4 (2.4.18-14).
- Windows 2000 without SP and with SP3.
- Windows NT 4.0 SP6a.
- Solaris 8.

For more details about the concrete versions used for this paper's research see "APPENDIX I: Operating Systems researched" section, where the network devices used are detailed.

As stated, the paper covers a common operating system forgotten in the security arena, probably because it is not as extended over Internet as other Unix variants, HP-UX. There is not so much information about how to configure it and the HP-UX internals, so the latest three releases that exist nowadays will be covered.

There are also other common and very relevant OS that should be analyzed in future versions of this paper, but every single OS that communicates through the TCP/IP protocol family could be analyzed if there is any interest in knowing its ARP behaviour:

- Cisco CatOS switches.
- Cisco PIX latest version.
- Cisco IOS other versions (latest one is 12.3), both routers and switches.
- HP-UX 11i Itanium versions.
- Linksys wireless access points.
- Linux: kernel 2.0 and 2.2, for historical reasons, and 2.6 (available soon).
- Linux different distributions.
- Nokia IPSO (latest version is 3.6).
- Solaris 2.5 and 2.6, for historical reasons, 7 and 9.
- Windows home versions: 9x (95 and 98 SE) and ME.
- Windows 2000 other SPs (latest one is SP4).
- Windows NT 4.0 older SPs.
- Windows XP different SPs.
- Windows 2003.

Protocols/Services/Applications

As its name indicates, the ARP poisoning/spoofing attacks affects the ARP protocol, one of the communication protocols associated to the TCP/IP protocol family.

ARP as its name denotes, was created to dynamically associate or translate protocol or layer 3 addresses (*IP addresses)*, to hardware or layer 2 addresses (*Ethernet addresses*), so any device could communicate with any other device just by knowing its layer 3 address.

ARP matches the Ethernet address associated to a host to its IP address in a transparent way from the point of view of the upper layer protocols and services. The ARP protocol is described in the RFC 826 [RFC826].

IP is the de-facto network protocol most extensively used in today's networks, not only Internet but also companies and organizations internal networks (Intranets) and private inter-company networks (Extranets), and Ethernet is the most commonly used technology for LANs.

Brief Description

The "ARP spoofing" attack is based on impersonating a system in the network, making the two ends of a communication believe that the other end is the attacker's system, intercepting the traffic interchanged.

To achieve this goal, the attacker just needs to send a previously modified ARP packet, method known as packet crafting, to the source system of a given communication saying that the destination IP address belongs to his own MAC address. In the same way, it will inform the destination system, through a second crafted ARP packet, that the IP address of the source is associated to his MAC address too. So from this moment, both systems will interchange information through the attacker's system and only because attacker's system has asked them politely to do so ;-).

The main motivation for this paper's research was originated after more than two years of Internal Penetration Testing on production environments, where the security auditor plays the attacker's role.
By "internal" it is stated any attack developed from inside the network, so the attacker could be any "insider": company employee, subcontractor, or third party support engineer or consultant working inside the network… or ¡¡ an external attacker having control of any internal system !!.

Although the ARP spoofing technique is very simple in concept, in real world situations over heterogeneous networks, the obtained results are not always as expected, because every OS and network topology influences the ARP protocol behaviour. Heterogeneous networks where not expected behaviours have been found contain every kind of computer systems: Windows desktop users and Windows servers (different versions and Service Packs), Unix servers (different

OS – different versions: Linux, HP-UX, Solaris…), firewall appliances (different boxes: Nokia, Linux based, Cisco PIX…), network equipment (Cisco, Nortel, 3Com, HP…).

The goal of this paper is to develop a deep analysis, from a real world perspective associated to the penetration tests experience previously mentioned, that would allow to have a complete overview about how every OS analyzed behaves in relation to ARP, its different possible stimulus and responses, and the ARP spoofing attack redirection games.

Finally, we need to emphasize that this type of attack must be developed from the internal portion of the network, but it is not only associated to internal attackers (as mentioned before), because every time an external attacker takes control of an internal system (by any other attacking method or vulnerability), he has the same access level as anyone on the internal network.

The CSI/FBI's security industry research provides statistics about the number of attacks developed from inside and outside the network, and, though the overall conclusion is that the internal thread is decreasing, since 1996, it is still quite alarming:

> (Year 2002) "For the fifth year in a row, more respondents (74%) cited their Internet connection as a frequent point of attack than cited their internal systems as a frequent point of attack (33%)." [CSI2002]
>
> (Year 2003) "Survey results illustrate that computer crime threats to large corporations and government agencies come from both inside and outside their electronic perimeters, confirming the trend in previous years. Forty-five percent of respondents detected unauthorized access by insiders. But for the fourth year in a row, more respondents (78 percent) cited their Internet connection as a frequent point of attack than cited their internal systems as a frequent point of attack (36 percent)." [CSI2003]

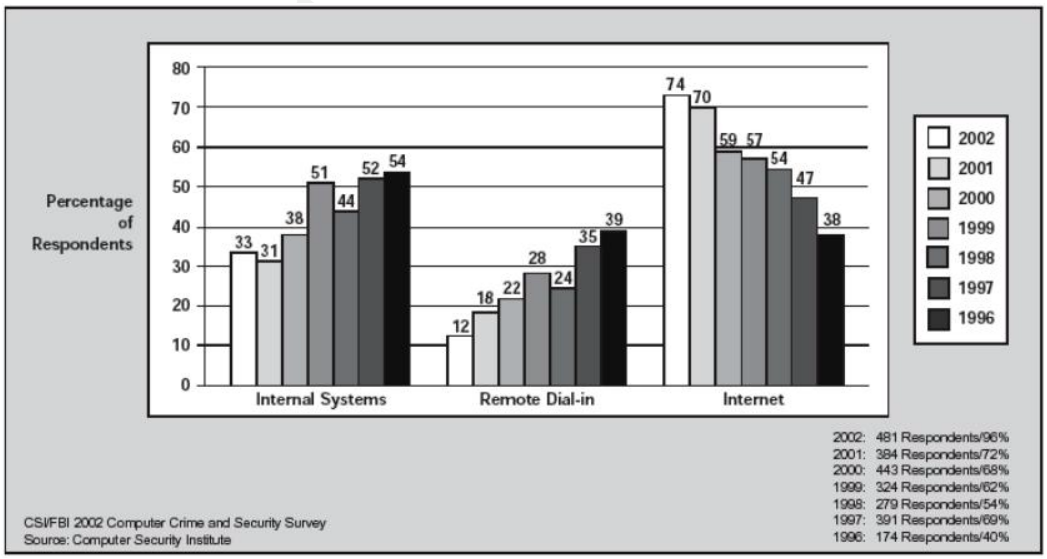


Figure 1.1. CSI/FBI statistics from 1996 to 2002

Some conclusions obtained from my personal experience are:

- Companies are more and more focused in protecting their infrastructure from outside threads, using firewalls, antivirus server solutions, DMZ IDSes and Web content filtering devices, so they are getting more and more capable of detecting and reacting against external attacks. However, lots of internal attacks are not detected at all and keep unnoticed.
- There are typically several elements that could increase the internal intruders risk against external intruders: best knowledge about the computer and network infrastructure, authorized access to some computer systems or a specific motivation or goal driving the attack.

Variants

Sometimes [SOLA1] the term "ARP spoofing" is used to refer to an attack where the intruder associates, through manual configuration, the IP address of another host to his own system. The attacker's system will impersonate the IP address of the host by responding with its own MAC address. Due to the fact that this is a mixture of "ARP spoofing" and "IP spoofing", and at the end, the purpose is to poison the target ARP table it is preferred the usage of the definition presented in the "Name" section.

It is possible to perform an attack based on forging the MAC address of a given Ethernet frame, pretending to be a different system, with the idea of bypassing the network devices controls based on the source MAC address. This attack is commonly confused with the previous one, and should be called "MAC spoofing", in the same way "IP spoofing" attacks refer to the possibility of sending IP packets with a forged source IP address.

This type of attack is worthless against network devices that use static physical security control by port, that is, that only accept Ethernet frames with a specific MAC address coming from a given port. This attack can also be used to confuse the switch (its CAM table) and get an association of the MAC address of another host to the port where the attacker is plugged in.

In a deeper level of complexity, there are different MAC addresses involved in the ARP packets that should be differentiated. On the one hand we have the destination and source MAC addresses inside the Ethernet frame header (manipulated in the "MAC spoofing" attack described previously) and on the other hand, we have the MAC addresses, sender and target, included in the ARP packet (involved in the ARP spoofing attack described previously). There is no reason why this pair of addresses must be the same in a given packet, although the correct usage of the ARP protocol would suggest so

It is possible for an attacker to craft ARP packets mixing all concepts analyzed, forging any MAC address, at the Ethernet frame or the ARP packet, using real or fake MAC addresses or combining all them.

References

They following references conform a basic set of sources to understand the ARP protocol and the way the "ARP spoofing" attack works. For a complete list of references see "List of References" section:

Plummer, David C. "RFC 826: An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware". Network Working Group. November 1982. URL: ftp://ftp.rfc-editor.org/in-notes/rfc826.txt (11 Jun. 2003).

Stevens, W. Richard. "TCP/IP Illustrated, Volume 1. The Protocols". Addison Wesley Longman, Inc, 1994. ISBN: 0201633469.

Yuri Volobuev. "Redir games with ARP and ICMP".
URL: http://lists.insecure.org/lists/bugtraq/1997/Sep/0059.html (17 May. 2003)

Sean Whalen (arpspoof@gmx.net). "An Introduction to Arp Spoofing". Revision 1.8. April, 2001. URL: http://chocobospore.org/arpspoof (1 Aug. 2003)

Search by the "arp" term in Sans Reading Room: URL: http://www.sans.org/rr/ (17 May. 2003)

Robert Wagner. "Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks". Practical Assignment GSEC Version 1.2f (amended August 13, 2001). URL: http://www.sans.org/rr/threats/address.php (10 May 2003)

For information about security tools that implement this exploit see "ARP spoofing tools" section.

Terminology and conventions

About the terminology used:

- Along the text the terms *Ethernet, MAC, hardware, HW* and *physical address* are used indifferently, that is, standing for the layer 2 addresses in the OSI model.
- The terms *protocol, network* or *IP address* refer to the layer 3 addresses in the OSI model.
- The terms *ARP table*, *ARP cache*, *ARP table cache* refer to the same concept.
- The terms *ARP spoofing* and *ARP poisoning* refer both to the attack previously described.
- *LAN*: Local Area Networks.
- *OS*: Operating System/s.

Several questions, that will be analyzed from a security point of view, will be raised all along the paper, so an identifier has been created to refer to them in different sections. The identifier is "[*Qn*]" where "*n*" is the question number.

Part 2 – The Attack

Description and diagram of network

The networks exposed to the described attack are any LAN, independently of the interconnecting network devices used: bridges, hubs, switches and layer-n switches. The local network topology doesn't affect the scope of the attack: "any system placed in a specific local network can potentially exploit any other system in the same LAN".

The usage of VLANs restricts the scope of the attack to an specific VLAN, but this is not different from the scope described before because the VLAN concept just expands the LAN concept between multiple physical locations (switch ports and switches), so it doesn't increase the security against this attack. The previous sentence could be rewritten: "any system placed in a specific local network (LAN or VLAN) can potentially exploit any other system in the same LAN or VLAN". So this attack can be consider an internal thread.

We will focus the analysis only in two of the most currently used protocols, even though the ARP protocol is totally generic and could be used over any pair of protocols. We will analyze the **Ethernet protocol** (layer 2), used in most of the local area networks and the **IP protocol** (layer 3), used all over Internet and most of its associated developments, the intranets and extranet networks.

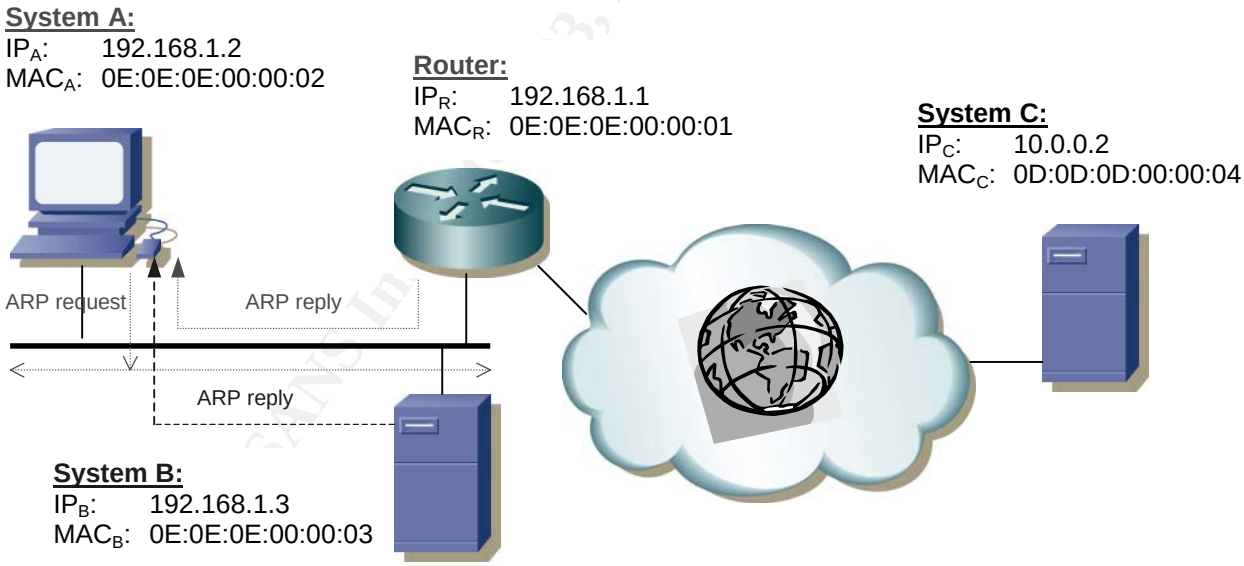


Figure 2.1. ARP protocol description: network diagram

The Ethernet concept includes all different possible and common speeds: Ethernet (10 Mbps), Fast-Ethernet (100 Mbps) and Giga-Ethernet (1 Gbps). It seems there is no real ARP implementation for other layer-2 protocols. Other

possible layer 3 protocols, although old and rarely used, could be CHAOS, Xerox PUP and DECnet.

The operating systems potentially exposed have been referenced in the "Operating Systems" section.

At the link layer level there are two main protocols for Ethernet technologies, the Ethernet protocol (RFC 894 [RFC894]) and the IEEE 802 protocols (RFC 1042 [RFC1042]). The later also covers non-Ethernet protocols as Token Ring. RFC 1122 [RFC1122] defines in its 2.3.3 section that a host connected to an Ethernet cable must speak Ethernet encapsulation and should speak IEEE 802, intermixed or not with Ethernet frames. This is the reason why we focused this paper on the Ethernet encapsulation frame, because it is the most commonly used implementation in Ethernet networks, from both a theoretical and a practical point of view.

There are other network technologies, not covered by this paper, like ATM, Asynchronous Transfer Networks, that doesn't have similar technologies as the associated to LAN networks natively, so they need to emulate LAN behaviour, including ARP, through special solutions, as LAN Emulation [RFC2225] [ATM1].

Protocol description

What is the purpose of the ARP protocol?

The ARP protocol [WIKI1] [HYPE1] was created to dynamically associate or translate *protocol addresses*, layer 3 addresses in the OSI model (network layer) or typically *IP addresses* in the data networks and Internet, to *hardware addresses* (also known as physical addresses), layer 2 addresses in the OSI model (data link layer) or typically *Ethernet addresses*, today's most commonly used technology for LANs.

This section provides a very detailed description of this protocol to be able to provide the reader a minimum knowledge require in further sections.

Due to the existence of the Ethernet protocol it is possible that several layer 3 protocols coexist in the same physical network, because every protocol will be identified by an Ethernet header field, called *protocol type* (see Figure 2.3). However, the Ethernet addresses are 48 bits long, but the protocol addresses may vary. For example, IP addresses are 32 bits long and have no relationship with the physical or Ethernet address.

ARP is an Address Resolution Protocol that matches the Ethernet address associated to a host to its IP address in a transparent way from the point of view of the upper layer protocols and services.

Although typically ARP is placed at the link layer in the communication stack (see Figure 2.2), and sometimes as a layer 3 protocol, we could specify it as a

"bridge" protocol between the Ethernet network driver (layer 2) and the IP protocol module (layer 3), integrating both layers.

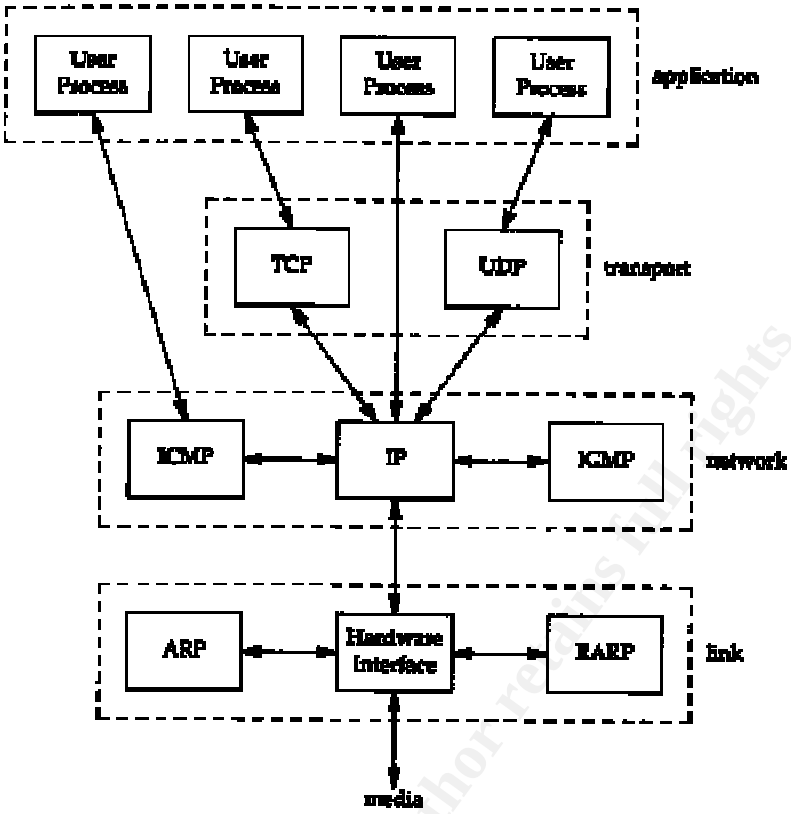


Figure 2.2. TCP/IP protocol suit layers ([STEV1], Figure 1.4)

MAC addresses: the lowest level network name

To understand the ARP protocol it is necessary to analyze what one of the two types of addresses used are, the MAC addresses, or Media Access Control addresses.

MAC addresses are also called hardware addresses or physical addresses. As the own names suggests, they are physically available in the network devices, that is, they were burnt into the network card's memory chipset when it was manufactured. The MAC address of a given network card is a unique universal value.

The reason why they exist is that every network device must be identified in the network somehow, for example, with the IP address at the layer 3 of the OSI model, or with the MAC address at the layer 2 of the OSI model. Before knowing which IP address belongs to which device (this is where ARP will help) we must be able to uniquely identify each and every device, and the only way to do this is through its layer 2 name, the MAC address.

Some devices have no IP address manually configured when they boot, so the only name they have (so other devices can identify them) to communicate with the external world is the MAC address. By using this address in conjunction with

other protocols, such as BOOTP or DHCP, they will obtain its protocol name, that is, their IP address.

What does a MAC address look like?

A MAC address is a hexadecimal string conformed by 6 hexadecimal pairs, that is, 6 bytes or 48 bits long. It is divided in 2 sections. First section contains the hardware vendor code, a specific identifier assigned by the IEEE group to every network card manufacturer (some companies have more than one id because they have built lots of different network equipment along time). This avoids the chance of collisions between MAC addresses associated to the network cards built by different vendors. The vendor's codes are defined in RFC 1700 [RFC1700] [IANA] [PATTON1] and the most recent database can be found in the IEEE web page [IEEE1].

MAC address example:	00-10-83-*0F-0F-0F*

As explained, the first portion of the MAC address is known as the OUI, Organizationally Unique Identifier, or generally speaking, the company identifier of the network card manufacturer. In this example, the vendor code is "00-10-83", which identifies "Hewlett-Packard" as the manufacturer of the network card.

The second portion is a number assigned by the vendor, and it must be different from every other network card manufactured by it. The pair OID plus card-code gives a unique network card identifier. In the example "0F-0F-0F" is the card code, assigned by the vendor.

MAC addresses types: Unicast & Broadcast & Multicast

All systems in an Ethernet network will listen for frames which contains their MAC address as the destination address field in the Ethernet frame header, but, to be able to optimize the network traffic and to increase the communications performance, there should be a way of sending packets not only to one system, but to several systems at the same time.

Due to this reason any system can send traffic to a set of targets based on the destination address used:

- Unicast: packets targeted to only one destination system.
- Broadcast: packets targeted to all hosts belonging to the same network. By design, all hosts belong to the broadcast group of their network.
- Multicast: packets targeted to a specific set or group of hosts in a network. Every host selects whether it wants to participate or not in a specific group.

The unicast address of a host is its MAC address, for example, 0x0010830f0f0f, and the broadcast address is always 0xffffffffffff. The most difficult addresses to manage are multicast addresses, where some systems should receive the data packets and others shouldn't [FIRE1].

To sum up, the Ethernet multicast addresses have the low-order bit of the high-order octet equal to 1, while unicast packets have it equal to 0. So, the following table shows the Ethernet unicast and multicast general rule. IEEE specifies the standard multicast addresses officially registered (IANA) [IANA] [IANA2].

Ethernet address	Range	Description	Bit
Unicast	X**Y**-XX-XX-XX-XX-XX	Y is an odd number: 0,2,4,6,8,A,C,E	0
Multicast	X**Z**-XX-XX-XX-XX-XX	Z is an even number: 1,3,5,7,9,B,D,F	1

IP multicast uses the Class D address range, 224.0.0.0 to 239.255.255.255, and it is usually combined with MAC multicast model to map IP multicast domains. A detailed explanation about the mapping algorithm between both can be obtained from [FIRE1]

RFC 1112 summarizes it as follows [RFC1112]:

> "An IP host group address is mapped to an Ethernet multicast address by placing the low-order 23 bits of the IP address into the low-order 23 bits of the Ethernet multicast address 01-00-5E-00-00-00 (hex). Because there are 28 significant bits in an IP host group address, more than one host group address may map to the same Ethernet multicast address."

The main IP multicast addresses are [IANA2]:

- 224.0.0.0 Base Address (Reserved)
- 224.0.0.1 All Systems on this Subnet
- 224.0.0.2 All Routers on this Subnet

ARP packet format

The following two figures show the ARP packet format, embedded in an Ethernet frame. The Ethernet frame is composed of two elements, the Ethernet header and the Ethernet body, just filled up with the ARP packet contents.

The Ethernet frame refers always to the Ethernet version II packet type, also known as Ethernet ARPA type [ETH2].

The Ethernet header has 3 different fields:

- Target or Destination Hardware Address (48 bits): system MAC address this frame is addressed to.
- Sender or Source Hardware Address (48 bits): system MAC address this frame was generated from.
- Protocol Type (16 bits): encapsulated next-layer protocol.

When sending ARP packets, the *Protocol Type* field is always set to the value representing the ARP protocol, "0x0806", defined in "Ether Types" [IANA].

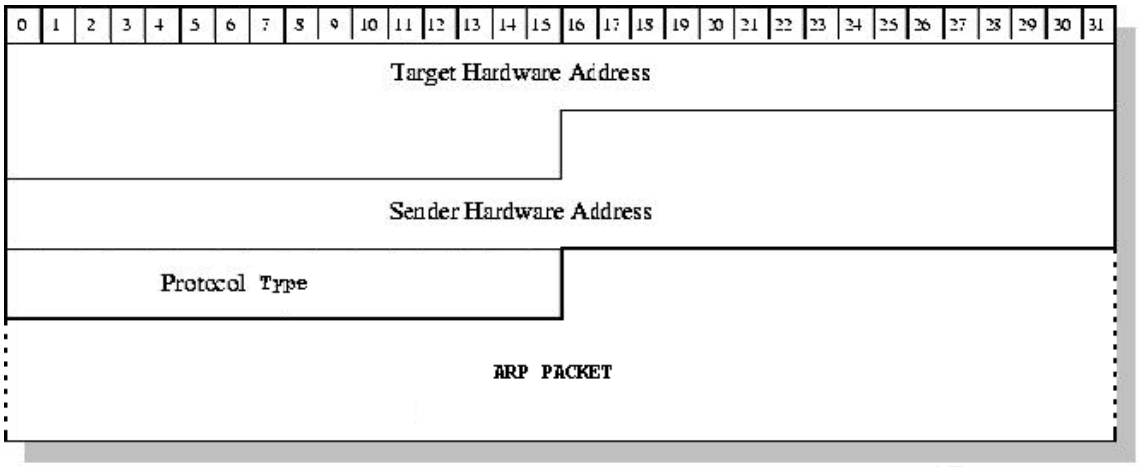


Figure 2.3. Ethernet frame layout

The Ethernet information is not always easily accessible at the user level when running some sniffing tools.

The ARP packet has 9 different fields:

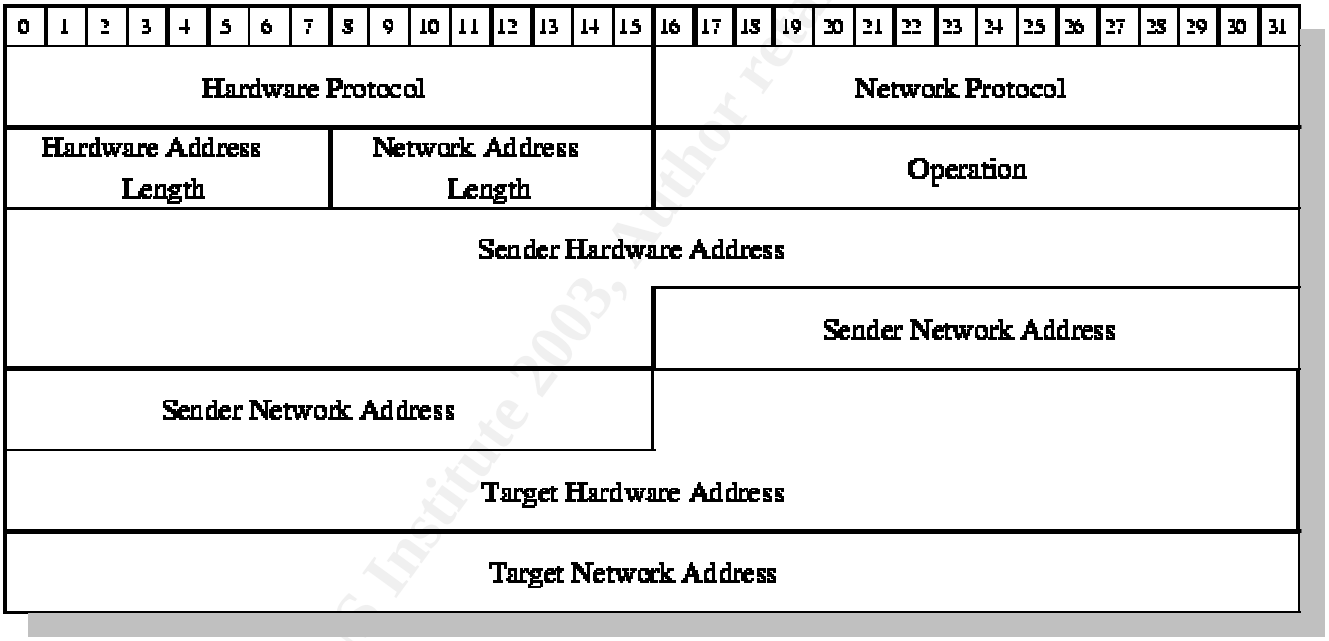


Figure 2.4. ARP packet layout ([SIL1], page 13)

Fields description:

- Hardware Protocol (16 bits): It specifies the hardware technology to be used, typically Ethernet, so it is set to "1".
- Network Protocol (16 bits): Layer 3 protocol to be used in the translation between hardware and protocol addresses, typically IP protocol, value "0x0800".
- Hardware Address Length (8 bits): length of the hardware addresses (in bytes). For Ethernet addresses its value is always "6".

- Network Address Length (8 bits): length of the protocol addresses (in bytes). For IP addresses its value is always "4".
- Operation (16 bits): type of ARP operation. The ARP protocol allows 2 types of operations:
  - Request: the value is set to "1".
  - Reply: the value is set to "2".
- Sender Hardware Address (48 bits): Ethernet address of this ARP packet sender.
- Sender Network Address (32 bits): protocol address of this ARP packet sender.
- Target Hardware Address (48 bits): Ethernet address of this ARP packet target.
- Target Network Address (32 bits): protocol address of this ARP packet target.

The 802.3 specification defines that Ethernet also supports 2-byte addresses, instead of the 6-byte standard addresses, but I'm not aware of any implementation of this [ETH1].

How does the ARP protocol work?

When a system, "System A" in Figure 2.1, needs to send information to another system, for example one or more IP packets, the operating system routing module evaluates if the information should be sent to the same subnet in the local network, "System B", subnet 192.168.1.0/24, typically the same Ethernet segment, or if it is addressed to a different remote subnet, "System C", subnet 10.0.0.0/8.

In both cases ARP needs to play its role because some work should be made to get the next Ethernet address to send the information to, also referred as the next hop. The only piece of data available to get the next hop is the IP address associated to it (destination host or router) that has been obtained from the routing table.

ARP is a data link only protocol, that is, it can be used only to translate IP addresses of hosts that are placed on the same subnet or Ethernet segment (bridges, hubs or switches) but it cannot go through routers between networks.

In the first case, communication with the same subnet, A, knows the IP address (provided by user or application) of the target system, "System B, 192.168.1.3", it wants to communicate with, but it doesn't know its hardware address, so it needs to send an ARP packet to obtain the target Ethernet address (0x0E:0E:0E:00:00:03) to be able to send the information, encapsulated in an Ethernet frame, to the final destination.

In the second case, communication with a remote subnet, "System A", given the final destination host, "System C", knows the IP address of next hop to which it should send the packet, "Router, 192.168.1.1", because its routing table provided it, so, given the fact A doesn't know the router's Ethernet address, it needs to send an ARP packet to obtain it (0x0E:0E:0E:00:00:01).

For this reason, ARP is a very important protocol, because it is often the very first packet generated before starting any other TCP/IP communication. As it can also be noticed, it is used very frequently, whenever any IP protocol communication takes place.

For example, some statistics were specifically taken for this paper to analyze how important the usage of this protocol is. Roughly speaking, about the 25% of the network traffic in a desktop Windows based LAN (21 bits mask) is associated to the ARP protocol. For a server class C LAN, Unix based, about the 15% has been measured. These are rough numbers and they are completely network dependent.

Also the importance of the ARP traffic resides in the fact that the packets of this protocol are typically the first traffic seen coming from a system from the network point of view (see "APPENDIX V: the "arp" command" section).

To optimize this lookup process, the ARP module maintains a resolution table where the pairs *(IP address, Ethernet address)* are kept. This ARP table, also called ARP cache, and its management should be analyzed from the security point of view because it directly affects the different systems ARP behaviour.

When a new communication is started, the destination IP address is searched in the ARP cache. If the entry is found, the packet is immediately sent to the Ethernet address extracted from the table. If not, one ARP request packet must be generated asking for the Ethernet address of the system that has the destination IP address of the destination host or the next hop (router). This packet is broadcasted to all systems in the local area network because the target system MAC address is unknown. System keeps waiting for a response from the system with the mentioned IP address.
It depends on the implementation to keep or not (throw it away) the higher network layer packet that has originated the ARP lookup process. Instead of keeping only this packet, it can have a buffer to store a set of packets.

- Example: ARP request packet from System A to System B (captured with "ethereal" [ETHR1])

```
Frame 1 (60 on wire, 60 captured)
    Arrival Time: Jun 13, 2003 13:49:47.974304000
    Time delta from previous packet: 0.038706000 seconds
    Time relative to first packet: 4.168291000 seconds
    Frame Number: 1
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II
    Destination: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source: 0e:0e:0e:00:00:02 (0e:0e:0e:00:00:02)
    Type: ARP (0x0806)
    Trailer: 000000000000000000000000000000...
Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
```

```
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: 0e:0e:0e:00:00:02 (0e:0e:0e:00:00:02)
    Sender IP address: 192.168.1.2 (192.168.1.2)
    Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.3 (192.168.1.3)
```

The main points about this packet are:

- It is addressed to everyone in this subnet, "Destination MAC Address" in Ethernet header is the broadcast address, 0xffffffffffff.
- Sender doesn't fill the "Target MAC Address" in the ARP header because it is the value it is trying to obtain.
- Sender fills up the "Sender MAC and IP Addresses" in the ARP header with its values and sets the "Target IP Address" with the only data it has available about the destination, the IP address.

The destination system will receive the ARP request and, given the fact that the requested IP address belongs to itself (based on the ARP "Target IP Address" field), it replies with a unicast Ethernet packet containing an ARP reply, directed to the system asking for it. All the other systems will discard the ARP request because it is a unicast packet not addressed to them.

When the ARP reply is received, the system will store the information in its ARP table allowing future communications with the same destination host without the need of further ARP packets, neither broadcast nor unicast.

- Example: ARP reply packet from System B to System A

```
Frame 2 (42 on wire, 42 captured)
    Arrival Time: Jun 13, 2003 13:49:47.974341000
    Time delta from previous packet: 0.000037000 seconds
    Time relative to first packet: 4.168328000 seconds
    Frame Number: 2
    Packet Length: 42 bytes
    Capture Length: 42 bytes
Ethernet II
    Destination: 0e:0e:0e:00:00:02 (0e:0e:0e:00:00:02)
    Source: 0e:0e:0e:00:00:03 (0e:0e:0e:00:00:03)
    Type: ARP (0x0806)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: 0e:0e:0e:00:00:03 (0e:0e:0e:00:00:03)
    Sender IP address: 192.168.1.3 (192.168.1.3)
    Target MAC address: 0e:0e:0e:00:00:02 (0e:0e:0e:00:00:02)
    Target IP address: 192.168.1.2 (192.168.1.2)
```

The main points about this packet are:

- It is addressed to the inquiring system, the one that sent the ARP request, so it is NOT a broadcast packet.
- Sender has filled the "Sender MAC Address" because it is the value it was asked for.
- Sender fills up the "Target MAC and IP Addresses" in the ARP header with the values associated to the system that placed the request and sets the "Sender IP Address" with its IP address, previously known by the requester.

The entries placed in the ARP table have an expiration time, so they are removed after a period of time. The ARP table timeouts should be analyzed from the security point of view because they also affect the ARP table management.

Initially, the ARP tables of both systems, A and B, don't have any entry related to each other. The "arp" command allows the visualization of the ARP table in the most common OS, Windows families and Unixes (see "APPENDIX V: the "arp" command" section).

Initially both systems have its ARP table empty:

```
SystemA# arp -a
SystemA#

SystemB# arp -a
SystemB#
```

After starting a new IP communication, for example, verifying the availability of one system through the "ping" command, using the ICMP protocol, both ARP tables are modified as follows:

```
SystemA# ping -c 1 192.168.1.3
PING 192.168.1.3 from 192.168.1.2 : 56(84) bytes of data
.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=255 time=0.918 ms

--- 192.168.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% loss, time 0ms
rtt min/avg/max/mdev = 0.918/0.918/0.918/0.000 ms
SystemA#
```

Network traces summary associated to the "ping" command:

```
No. Time       Source                Destination           Protocol       Info
1   0.168291   0e:0e:0e:00:00:02     ff:ff:ff:ff:ff:ff     ARP
Who has 192.168.1.3?  Tell 192.168.1.2
2   0.168328   0e:0e:0e:00:00:03     0e:0e:0e:00:00:02     ARP
192.168.1.3 is at 0f:0f:0f:00:00:03
3   0.168728   192.168.1.2           192.168.1.3           ICMP     Echo (ping) request
4   0.168791   192.168.1.3           192.168.1.2           ICMP     Echo (ping) reply
```

Each system has populated their ARP table with the new entry. A, the one making the request, populates its table when receiving the ARP reply packet from "System B", but B adds the new entry when it receives the ARP request packet. This behaviour increases the protocol performance and to reduce the overload, because it is assumed by B, that if A is asking for me is because it is going to communicate with me, so in a near future I will need to respond, and

will need to know its Ethernet address. Then, why not recording it right now, once I know this situation is going to happen? Acting this way, B is avoiding sending a new ARP request asking for "A's" Ethernet address.

```
SystemA# arp -a
192.168.1.3 at 0E:0E:0E:00:00:03 [ether] on eth0

SystemB# arp –a
192.168.1.2 at 0E:0E:0E:00:00:02 [ether] on eth0
```

The output for this example has been extracted from a Linux system running kernel version 2.4. For a more complete explanation about the "arp" command output formats see Appendix "Output format per Operating System" section.

When a system needs to communicate with a remote system located in another subnet, "System C", it needs to send the packets to the designated router. From ARP perspective, the Router is in the same situation as "System B" (see Figure 2.1) and the only difference is that A will learn the Router Ethernet address instead of the destination host "System C" Ethernet address.

To summarize, the typical ARP packet interchange is as follows:

ARP request packet:	"ARP: Who has <target_IP_addr>? Tell <sender_IP_addr>"

ARP request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr.		
Sender IP addr.		
<NOT DEFINED>		
Target IP addr.		

ARP reply packet:	"ARP: <sender_IP_addr> is at <sender_MAC_addr>"

ARP reply		
Target MAC addr.		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	2 (reply)
Sender MAC addr.		
Sender IP addr.		
Target MAC addr.		
Target IP addr.		

Relationships between ARP reply and ARP request packet:

- <sender_MAC_addr> is the value requested.
- <sender_IP_addr> is the <target_IP_addr> of the ARP request packet.
- <target _MAC_addr> is the <sender _MAC_addr> of the ARP request packet.
- <target _IP_addr> is the <sender _IP_addr> of the ARP request packet.

Multihomed systems, which are systems with more than one network interface, maintain an ARP table per interface, as shown bellow.

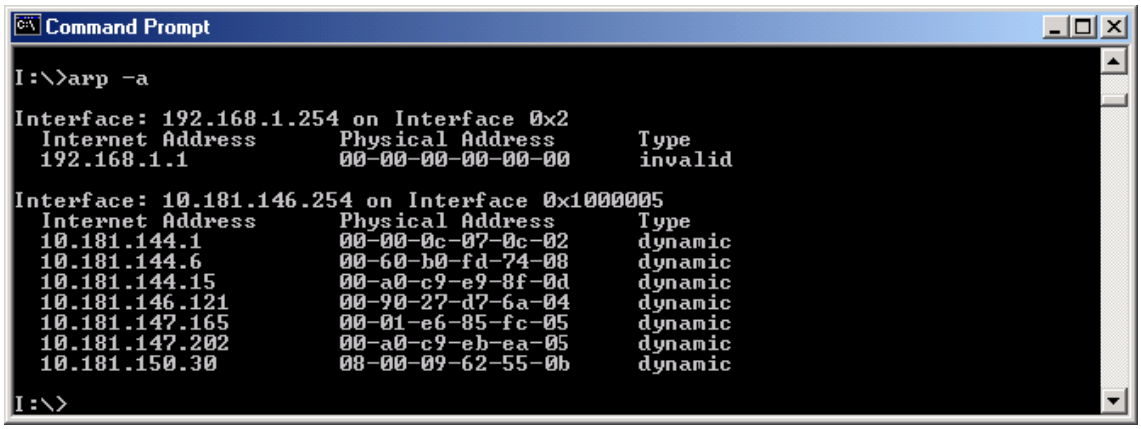


Figure 2.5. Two ARP tables for two NICs in different subnet

Typically there is a table per network because every interface used to be associated to a different IP subnet, but some load balancing solutions use multiple interfaces connected to the same network, although this can be problematic in terms of ARP [LVS1]. Some OS, as Windows 2000 SP3, maintain different ARP tables independently of the IP subnet they are associated to.

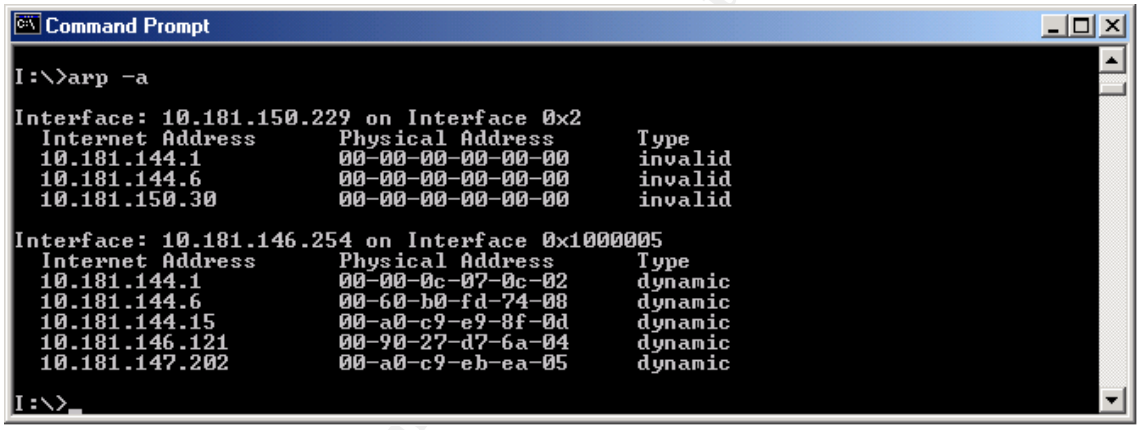


Figure 2.6. Two ARP tables for two NICs in the same subnet

Other OS, although they internally have an ARP table per interface, only show one global ARP network, as for example, HP-UX 11i:

```
system:/>arp -an
 (10.181.132.1) at 0:0:c:e7:ec:e3 ether
 (10.181.132.11) at 8:0:9:e6:e4:e3 ether
 (10.181.132.96) at 8:0:9:e2:e4:ea ether
 (192.168.1.254) at 0:0:86:e5:e5:e8 ether
 (10.181.132.237) at 8:0:9:e0:ea:e1 ether
 (10.181.132.20) at 0:d0:b7:ef:e2:ec ether
 (10.181.132.15) at 8:0:9:ea:e0:ea ether
10.181.133.111 (10.181.133.111) -- no entry
system:/>
```

In a future research it will be interesting to see how the different OS analyzed manage the ARP tables in multihomed systems, checking both cases, two network interfaces in the same or in a different subnet.

RFCs security analysis

To be able to develop the deep analysis this paper tries to afford we should analyze every aspect related to the protocol exploited, and for that purpose we will play the role of an OS ARP module developer.

ARP is typically a module belonging to the TCP/IP stack implementation inside the operating systems. The programming teams get the software specifications to develop the ARP module from the IETF, the Internet Engineering Task Force [IETF1], a community that publishes the RFCs, Request For Comments [RFC], which are a set of technical and organizational documents about Internet, its protocols and services.

To complement the specification analysis and the security implications of the ARP implementations, we could perform a deeper code research based on two of the best example references:

- The Linux ARP module source code, freely available:
  Linux kernel source code file: "/usr/src/linux-2.4.0/net/ipv4/arp.c"
- The TCP/IP implementation Bible [STEV2].

RFC 1433 [RFC1433], "Directed ARP", has not being analyzed in this paper because a first analysis suggests it just defines the usage of ARP packets to help dynamic routing protocols as BGP or OSPF advertise information about "foreign" networks. A future version could try to examine it in a deepest level.

RFC 826: the ARP protocol

The RFC 826 [RFC826] is the one that defines the ARP protocol, and although it is more than 20 years old, it was used by all the operating system designers to develop a compatible and standard implementation of ARP, interoperable between all different manufacturers.

Let's read the RFC document deeply and point out some aspects that should be analyzed later from a security perspective.

The first security concern that appears in the ARP RFC is the one that introduces "The Problem" section that the RFC tried to solve 20 years ago: "*The world is a jungle in general, and the networking game contributes many animals*". This doesn't seem to be a very secure beginning ;-).

The RFC is divided in two main sections and both are involved in the security risks that will be exposed later: packet generation (ARP request packet) and packet reception (ARP reply packet and ARP table management).

RFC: Packet generation section (ARP request packet)

The RFC states that this packet must be sent to the broadcast hardware address instead of to a specific unicast MAC address. It must be taken into account that a unicast packet has no sense because the destination MAC address is the value the sender is trying to obtain.

RFC doesn't determine what value the "Target MAC Address" field in the ARP packet should have. This is an open option, the RFC says "it could be set…", so it should be analyzed how the different implementations use it. RFC doesn't state what could be the other value instead of the broadcast address, 0xff:ff:ff:ff:ff:ff.

The ARP request packet specification triggers the following question:
[*Q1*] What happens if the "Target HW Address" field in the ARP packet is set to broadcast address (suggested in RFC 826)? Or to all zeros? Or to any other value: sender/target (hypothetically unknown) MAC address?

Based on the RFC, a system should only generate an ARP packet when it needs to contact another system and it doesn't have a valid translation in its ARP table.

ARP request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr.		
Sender IP addr.		
<NOT DEFINED> or FF:FF:FF:FF:FF:FF		
Target IP addr.		

Figure 2.7. ARP request packet based on RFC 826

RFC: Packet reception section (ARP reply packet and ARP table management)

Once a system receives an ARP packet, the packet should be processed based on the proposed algorithm showed in the RFC:

```
?Do I have the hardware type in ar$hrd?
Yes: (almost definitely)
  [optionally check the hardware length ar$hln]
  ?Do I speak the protocol in ar$pro?
  Yes:
    [optionally check the protocol length ar$pln]
    Merge_flag := false
    If the pair <protocol type, sender protocol address> is
      already in my translation table, update the sender
      hardware address field of the entry with the new
      information in the packet and set Merge_flag to true.
    ?Am I the target protocol address?
    Yes:
      If Merge_flag is false, add the triplet <protocol type,
        sender protocol address, sender hardware address> to
        the translation table.
```

```
?Is the opcode ares_op$REQUEST?
 (NOW look at the opcode!!)
Yes:
Swap hardware and protocol fields, putting the local
  hardware and protocol addresses in the sender fields.
Set the ar$op field to ares_op$REPLY
Send the packet to the (new) target hardware address on
  the same hardware on which the request was received.
```

It is the manufacturer's particular implementation what constitute the key element of the different behaviours which can be seen in real life situations; for example, the use of the "Merge_flag" variable on each particular OS.

First two questions of the algorithm are not relevant here because they expect a mapping between Ethernet and IP protocols. The "Merge_flag" tries to specify if a given entry is already in the ARP table.

A deeper analysis of the algorithm triggers the following security questions that will be checked lately:

- [*Q2*] Does a given implementation check the hardware address length (it is an optional step)?
- [*Q3*] Does a given implementation check the protocol length (it is also an optional step)?
- [*Q4*] If a pair is already in the ARP table, does the implementation always update the sender hardware address with the new received information, overwriting the information previously received? Based on the RFC it doesn't matter if the packet is addressed to me or not, or if it is a request or a reply, it is supposed it should always update the ARP table.
- [*Q5*] If the packet is addressed to me and I didn't have the entry before, is it added into the ARP table (new entry creation). Does every implementation act this way? Does it follow the same behaviour if the packet is a request or a reply?
  RFC defines that if the response is not for me I will only update the table, that is, refresh a previous entry.
- [*Q6*] If the packet is a request, I should reply if I'm the destination system. How does every implementation respond to different request and reply packets in which they are, or not, the destination? When is the operation code checked?

To summarize the RFC algorithm raises 3 questions:

1. Is a pair associated to the received IP address already in the ARP table?
2. Am I the target IP address?
3. If so, is the packet an ARP request?

The conclusions obtained from the RFC algorithm are very simple:

- It can be assumed that the RFC specifies to always update the table with every ARP packet received (request or reply) if there was a previous entry for the same IP address, and then…

- If I am the target IP address add the entry to the ARP table if it was not there (new entry), with both, request and replies. Besides, if it is a request, reply to the sender with my own information.
- If I'm not the target IP address, then I don't need to do anything else.

So the information in the ARP table should be always updated with the data contained in the ARP packet sender addresses fields, being the update a modification of a previous entry of any system or a new entry to be registered if I'm the packet destination.

ARP reply		
Target MAC addr.		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	2 (reply)
Sender MAC addr.		
Sender IP addr.		
ARP request Sender MAC addr.		
ARP request Sender IP addr.		

Figure 2.8. ARP reply packet based on RFC 826

Additional RFC 826 security analysis

Some additional questions extracted from the RFC security analysis are:

It is supposed that the only defined value for the Hardware address space field is 1, Ethernet networks. [*Q7*] How does a given implementation respond when it receives a packet with a different value?

[*Q8*] How does a given real implementation respond when it receives a packet with a Protocol address space field different from 0x0800, the one associated to the IP protocol?

The length fields in the ARP packet are redundant as the RFC denotes, because the number could be extracted from the address information simply by knowing the Hardware and Protocol address spaces. They were defined for consistency checking and to help parsing the addresses. [*Q9*] How many implementations really check the values placed in these fields?

This RFC end section, called "Related issues", is very important from the security point of view. It talks about the ARP table entry aging policies and timeouts involved, explaining why the algorithm was defined as described.

- The reason why an entry is merged into the ARP table before looking at the operation code was improving performance. If A is trying to talk to B, B will probably talk to A.
- The reasons for the ARP table timeouts existence were:
  - IP addresses relocation, that is, assigning an IP address to another piece of hardware.
  - To not allow incorrect "routing" information to persist forever.
  - Two specific behaviours were proposed:

1) Method 1: the ARP table timeout for a specific entry could be reset when a packet is received for a host, meaning that it keeps alive and kicking, so its associated entry is valid.
2) Method 2: another option is having an ARP daemon performing the timeouts, so when the timer expires it removes the entry and sends (with retransmissions allowed) a unicast ARP request to the previous learned MAC address. If a reply is not received, the entry is deleted.

It must be analyzed which method is implemented by every operating system [*Q10*].

- The reason why a new MAC address received supersedes the entry with the information previously learned was to lessen the recovery time when a host moves.
  Due to the fact that the existing entry is superseded, if a host moves, its first ARP request packet broadcasted to all stations will allow them to get the new MAC address.

The reason why the RFC defined ARP in the way it did was because it was designed with two concepts in mind: simplicity and performance. The first concept complements the latest.

Let's indicate how these two concepts are reflected in the RFC:

- **Simplicity**: the main goal was "*one resolution per packet*". This helped to design the packet format and the overhead to process it.
  Trying to obtain an algorithm as simplest as possible, the operation code is checked at the end, so the "Target Network Address" is not analyzed or differentiated between request and replies. The "Target Network Address" field is only needed in the ARP requests to know the IP address that someone is interested in, so every system asks itself, "Am I this IP address?". It is not needed in the replies if one assumes that a reply is only provoked by a request, but in nowadays evil networks, this is not always true.

  The "Target Hardware Address" has no meaning in the request because it is the address the requester wants to obtain, and its meaning in the reply is just to denote the request was originated from. Some implementations can save processing load using this field when the request is received, instead of the "Source MAC Address" included in the Ethernet header to reply to a host. From the security point of view, it should be analyzed how every implementation behaves if this two sender MAC addresses are different in a request [*Q11*].

- **Performance**: this was one of the main aspects involved in its creation, because this protocol is invoked in **every** communication **every** system does when transferring data through the LAN. If the cache table didn't exist, an ARP packet per IP packet would be generated, more than duplicating all the network traffic (per IP packet, an ARP request and a reply would be needed, three packets instead of just one).

RFC also states that "*Periodic broadcasting is definitely not desired*", introducing a possible broadcast resolution protocol in which every system broadcasted all the others its address pair (MAC, IP) periodically, instead of distributing information on demand.
A clear example of how important this was is that the packet length is the same for both operations, with the idea of reusing the packet buffer in memory when replying, as well as several fields, which are kept the same.

The following figure shows the memory changes an ARP request packet suffer to be converted to an ARP reply by the destination host (the one replying):

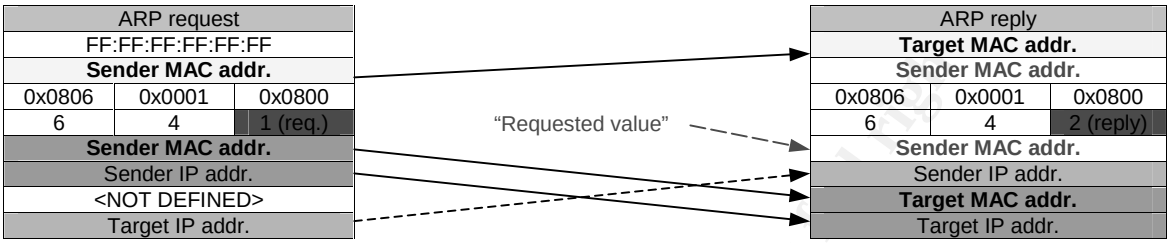


Figure 2.9. Memory swapping between ARP request and reply packets

Let's introduce a new attack method or tool not seen yet in the wild: the possibility of playing with the non-used fields in both types of ARP packets to carry on information belonging to a cover channel. For this purpose, the ARP requests "Target Hardware Address" field will be useful and, for ARP replies, the "Target Protocol Address" field could be useful as well. Besides, the "Target Hardware Address" field could be interesting for replies in some implementations. In the future version of this paper, a proof of concept code should be developed.

The RFC gives an example very similar to the one shown in the "How does the ARP protocol work?" section, but with the following shades:

- RFC doesn't determine the "Target HW Address" value in ARP request packets. It explicitly says "don't care".
- It denotes that a system will probably replace any existing ARP entry in the table when receiving a new ARP request packet.

Finally, to denote how simple, innovative and old this RFC was just figure out that it doesn't include any references.

It is necessary to analyze how every OS implements all the elements that belong to the original specification and to study the implications of all them from a system security perspective.

RFC 1122: ARP requirements for Internet hosts

The RFC 1122 [RFC1122] defines the specification associated to the Internet host's behaviour for the communication layers that conform the TCP/IP stack.

Section 2.3.1 defines the "Trailer Protocol Negotiation", a protocol to dynamically negotiate the use of trailers at the link layer based on the encapsulation used, with the idea of improving the network throughput, defined in RFC 893 [RFC893].
Trailer negotiation is based on the ARP protocol and performed when the ARP exchange takes place. This new feature adds new possible ARP packets.

A system that receives an ARP request and which wants to speak trailers will send its normal ARP reply plus an additional "trailer ARP reply", a normal ARP reply packet specifying the trailer encapsulation protocol type. It also affects the ARP table, because a trailer-speaker host will add a mark in the associated entry into its ARP table when receiving this type of reply.

If it is the requester who wants to speak trailers, it will send the same type of "trailer ARP reply" when it receives the normal ARP reply. So the extra "trailer ARP reply" may always be associated to a normal ARP reply. This extra reply is a gratuitous ARP packet.

From the security point of view this specification increments the stateless behaviour of ARP, allowing multiples ARP replies for a request. So it seems it could be easy for an attacker to simulate the second trailer ARP reply but modifying the "Sender MAC address" to poison the target ARP table.

For example, HP 9000 HP-UX 10.20 systems can receive trailer packets but do not send them. Setting the "trail" flag has no effect in this behaviour. At the kernel level the "ATF_USETRAILERS" flag is used to activate it. The same implementation constant is used in Solaris 8.

However, although this flag is available in the HP-UX 10.20 "arp" command, it is not in HP-UX 11 or 11i, and Linux "arp (7)" man page, in kernel 2.4, defines "ATF_USETRAILERS " as an obsolete option that should not be used. Due to this fact this paper version will not research this type of packets, leaving it for a future research.

Unix 4.2 BSD systems made use of the trailer encapsulation protocol to improve the VAX virtual memory architecture.

RFC section 2.3.2 defines two ARP requirements:

- Section 2.3.2.1, "ARP Cache Validation"
- Section 2.3.2.2, "ARP Packet Queue"

First section, 2.3.2.1, specifies that there must be a mechanism to flush out-of-date ARP table entries, manually or automatically. If a timeout is involved, its value should be configurable. Also it adds the necessity of having an anti flooding mechanism to avoid a high ratio of ARP requests per unit of time (1 per second per destination is the recommended rate).

Nowadays there is no doubt about why an ARP table expiration timeout is needed, but when this RFC was defined, Proxy ARP or bad ARP data were the main reasons. Let's say that security was not mentioned at all in the RFC!!

The defined mechanisms for this purpose were:

- Timeout: entries should be timed out periodically, even if they are in use. RFC states that the timer should be restarted when an entry is "refreshed", that is, when receiving a broadcasted packet containing it.
- Unicast Poll: RFC suggests sending a unicast ARP request to a given host and deleting the entry if no reply is received after "N" pools ("N" being 2).
  This specification introduces a new "allowed" ARP packet type, the directed or unicast ARP request, just suggested at the end of the RFC 826.
- Link layer advice: flush entries when the driver detects a delivery problem.
- Higher-layer advice: same as previous one when notification comes from higher-level protocols.

The suggested timeout for the first two options is 1 minute, but this could create a noticeable traffic overhead.

From the security side, the timeouts involved in every OS should be analyzed [*Q12*] and also the implementation of the "Unicast Poll" method [*Q10*].

Second referenced section, 2.3.2.2, defines that the link layer should save the latest packet received for a given destination not yet present in the ARP table, and transmit it when ARP had resolved the MAC address association. If this recommendation is not followed, the first packet of every communication will be lost, severely impacting performance. This design doesn't seem to affect security directly and several modern operating systems document they implement this type of buffering behaviour.

In section 2.5, the RFC specifies the following ARP features for hosts link layer and their requirements:

Flush out-of-date ARP cache entries	MUST
Prevent ARP floods	MUST
Cache timeout configurable	SHOULD
Save at least one (latest) unresolved packet	SHOULD

RFC 1812: ARP requirements for Internet routers

In the same way RFC 1122 specifies the requirements for Internet hosts, RFC 1812 [RFC1812] specifies requirements for IP version 4 routers. This RFC also covers both topics, trailer encapsulation and ARP behaviour.

Section 3.3.1 defines the "Trailer Encapsulation", denoting that routers may be able to speak trailers but should not generate them.

The ARP section, 3.3.2, specify that routers must be ARP compliant based on RFC 1122 and must not generate "ICMP Destination Unreachable" messages

for IP addresses with no entry in the ARP table, but they it should wait to complete the ARP exchange instead.

It also states that routers must not trust a host sending ARP replies claiming that its MAC address is a broadcast or multicast address. From the security point of view we will need to check if both, hosts and routers, deny ARP replies with this kind of MAC addresses [*Q13*].

This will involve the "Sender MAC address" for both, ARP request and reply packets, that is, the address field that is always used to learn MAC addresses.

RFC 1027: Transparent Subnet Gateways – Proxy ARP

During the old days, when Internet was developed, the concept of IP subnets was introduced, separating the networks into smaller different units and creating the concept of subnet masks and classless networks. There was a period in which some devices were subnet compliant and some others not. The Proxy ARP solution was defined in RFC 1027 [RFC1027] to solve this problem and hide the existence of subnets.

The Proxy ARP functionality is mainly associated to gateways, which will act as ARP intermediaries, representing other systems in a network from the ARP point of view. Gateways will respond to ARP requests in one of its network interfaces in behalf of other systems connected to another of its network interfaces. The gateway will responds with its MAC address and when packets are redirected to it, it will forward the packets to the final destination system.

This RFC also mentions the term "ARP hack", because, although probably not noticed at first instance, the gateway is hacking the network from the ARP perspective, in the same way an attacker does when performing ARP spoofing. Gateway is sending fake ARP replies, replacing the destination system identity, to be able to receive and redirect one side of the connection, forwarding the data to the destination, in the same way the attacker does. The only supposed difference is that the gateway is a trusted device.

The gateway advantage is that the other side of the connection, in its way back, works automatically, because the router is the official system to send the information to from destination to source. The only exception is when the gateway is acting as a Proxy ARP server for both sides (not always the case because it is possible to have several gateways between both networks), then, it plays the role of the "officially allowed ARP hacker" ;-), but without the evil component, manipulating or getting unauthorized data.

I'll just mention that this functionality has been widely implemented and almost all OS, mainly network devices as Cisco high-end switches and routers [CIPRO1], have options to set up a Proxy ARP server and publish other systems MAC and IP addresses pairs.

Nowadays all implementations are subnet compliant, so Proxy ARP functionality is used for dial-in remote host to connect to a network without consuming a

large IP address space, as they will have addresses in the same subnet although they are not directly attached to the network they are connecting to.

RFC 1868: ARP extension – UNARP

This RFC [RFC1868] introduces a modification in the ARP algorithms to allow a system to notify other/s that it is leaving the network, so that they can modify their ARP tables and remove the information associated to the leaving system.

The standard ARP protocol determines that nodes should not advertise their presence periodically for performance reasons. In the same way, it does not determine that a node needs to advertise its departure. Timers help in removing the obsolete information.

But there are some situations, mainly related to the Proxy ARP functionality, where a remote host can dynamically connect to a network through different paths. If for some reason the remote host, that was connected to the network through a gateway, disconnects and connects again through another gateway, any host in the network whose ARP entry has not expired will try to contact the remote host by using the first gateway instead of the second one. The communication won't be successful.

The RFC solves this problem by forcing the gateway to act as the remote host, notifying all other hosts that the previous ARP association is no longer valid.

This problematic situation, although not stated in the RFC, could also occur in some high availability solutions were an standby node becomes active, for example because the previously active node fails, and an ARP packet is not generated to notify all other systems that this node change has occurred (see "HA solutions" section). The same situation could happen in multihomed systems with active-standby network cards. If the working network interface fails the other becomes active, but although it will map to the same IP address, its MAC address will be different.

This paper proposes another solution for this problem from the "client" perspective: after "N" upper-layer protocol retries where it is not possible to communicate with a local network host, ask again for the MAC address associated to the destination using an ARP request packet.

Besides, this paper also suggests the idea of using the same solution when a DHCP client executes the "release" command to free its previously associated dynamic IP address, so that DHCP servers (acting as Proxy ARP servers) could send this type of packets too.

Finally, the RFC extends this solution to all ARP implementations, no matter whether the host is working as a Proxy ARP server or not.

So, based in this RFC, there is a new official ARP packet type belonging to the ARP reply unsolicited family, the UNARP reply, sent by Proxy ARP servers:

UNARP reply		
Target MAC addr.		
FF:FF:FF:FF:FF:FF		
0x0806	0x0001	0x0800
0	4	2 (reply)
<NOT_included> or 00:00:00:00:00:00		
Detaching host IP addr.		
<NOT_included>		
255.255.255.255		

Figure 2.10. UNARP reply packet based on RFC 1868

This packet should make nodes receiving it, if they are RFC 1868 compliant, remove any ARP entry associated to the "Detaching host IP address".

New security considerations this packet includes are:

- A "Hardware address length" value of zero, to avoid entries with a zero MAC value in those hosts receiving this type of packets but not being RFC 1868 compliant.
  It is supposed that nodes not supporting UNARP should reject this packet for being invalid for them [*Q2*].
- A "Target IP Address" with the value of the IP broadcast all.
- A "Sender MAC Address" with a zero value. This variation has not being included in the later research, so this option is kept for a future version.

Although the RFC expected a huge and broad implementation of this proposal (it even recommended a configuration switch to enable/disable this functionality in case of incompatibilities) none of the OS analyzed have this switch available.

A funny thing about this RFC is its section 5:

> "5. Security considerations
> Security issues are not discussed in this memo.
> …"

Let's say that at least the term security appeared in it ;-).

A quick security analysis about this proposal implications shows:

- It seems trivial that fake UNARP packets could be used to remove valid entries from systems ARP tables, allowing an attacker to remove a given ARP entry in all systems in a network.
  If systems are also capable of processing these UNARP packets even if they are specifically addressed to themselves, using Ethernet unicast packet instead of broadcast, then this technique will allow a selective ARP entry removal on specific hosts.

- This technique could be applied, for example, when a system doesn't allow updating an entry if it is already in the ARP table, as Solaris does, so an attacker could delete the existent entry, and then send an ARP packet to create a new one.

- It will also allow the usage of Denial of Service attacks in cases where the particular OS doesn't check the "Hardware Address Length" field, as it will update the entry associated to the packet IP address with a MAC address of zero. From now on the communication with this IP address won't be possible.

ARP packet types

This section has a summary of the different possible ARP packet types, based both on the described RFCs and in all the other references used along this paper:

ARP request:	**RFC**
Standard request to broadcast address.	826
Directed request to a unicast address to validate entry.	826,1122
ARP reply:	
Standard solicited reply to the host we received a request.	826
Standard solicited reply in behalf of other: Proxy ARP.	1027
Unsolicited Gratuitous ARP reply: host announcement.	N/A
Unsolicited UNARP reply: disconnecting from network.	1868
Unsolicited Trailer negotiation ARP reply.	1122, 1812

From all these packets the "Gratuitous ARP packet" is the one not explicitly studied in any of the RFCs analyzed.

Gratuitous ARP packet

A "Gratuitous ARP packet" consists of an ARP request or reply packet where the "Sender IP address" and the "Target IP address" are the same. Usually it is addressed to all hosts using the broadcast address for both, Ethernet "Destination MAC address" and "ARP "Target MAC address", although this is implementation dependent.

ARP gratuitous packet		
Target MAC addr.		
FF:FF:FF:FF:FF:FF		
0x0806	0x0001	0x0800
6	4	**1 or 2**
Sender MAC address.		
Host IP addr.		
FF:FF:FF:FF:FF:FF or implementation		
Host IP addr.		

Figure 2.11. ARP gratuitous request and reply packets

This type of packet is typically used in two situations:

- ARP reply [GRAT1]: when a host wants to announce its own IP-MAC addresses pair, for example in high availability clustering solutions (see "HA solutions" section).
  Due to the fact that the ARP request packets are also used to learn new IP-MAC addresses associations, an ARP request packet could be used for the same purpose.

- ARP request [STEV1]: when a host wants to check if there is a conflict with its IP address in the network, probably because another host is already configured with the same IP address. This situation generates a "Duplicated IP address message" when a reply is received (see ""Duplicate IP address" message" section).
  This type of checking is commonly done at bootstrap time (see "Bootstrap and shutdown times research" section), when they are initializing their IP stack.

How the exploit works

One of the reasons why ARP must be exposed is due to the open and flexible elements specified in the RFC and its interpretation in every implementation. All the questions and shapes stoop out along this paper may introduce potential vulnerabilities.

So the reason why this exploit could work lays on the ARP protocol design, and given the fact that the design is driven by the RFC, the simplicity and performance goals influenced the protocol security. It must be considered that security was not one of the main goals by the time when ARP was designed.

The ARP poisoning attack has been described in several papers [SEAN1] [ROB1] [VISV1] [DATA1] [SANS1].

For completeness we are going to include a brief description of how it works.

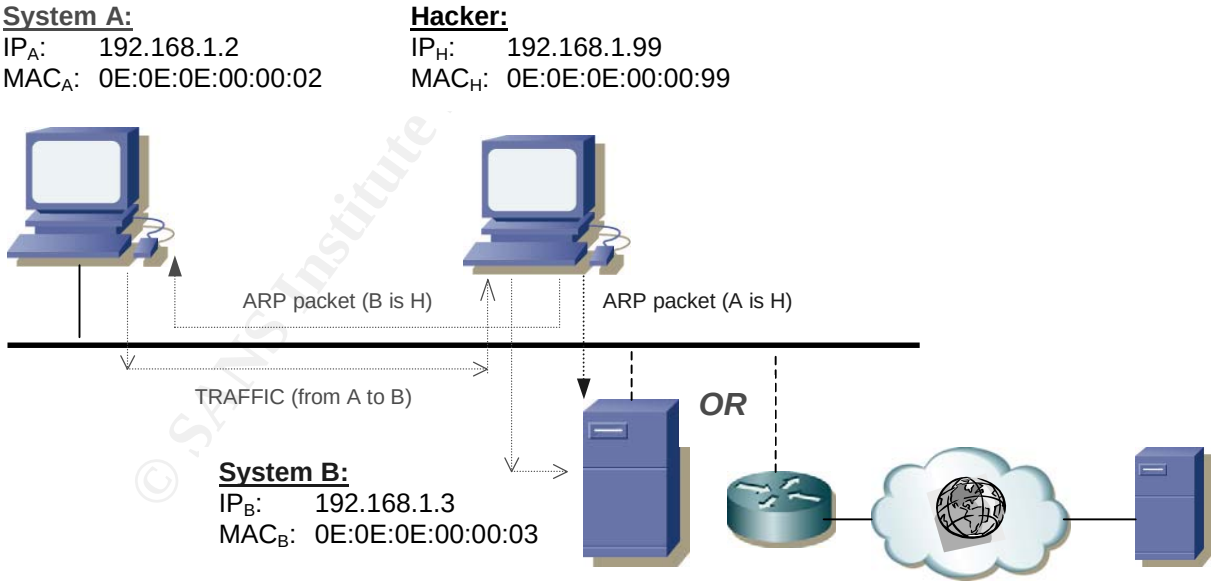


Figure 2.12. ARP spoofing attack: network diagram

The attacker's, "Hacker" system, goal is redirecting traffic going back and forth between both target systems, A and B, to be able to inspect it or develop more advanced attacks (see "Advanced attacks based on ARP Spoofing").

The first step the attacker must accomplish is to send to both systems an ARP crafted packet, typically an ARP reply (although this paper's research will evaluate other types), notifying that the IP address of the other end is at its MAC address, with the idea of poisoning target system's ARP tables:

- "Hacker" to "System A" (*red line*):
  ARP reply packet saying 192.168.1.3 is at 0E:0E:0E:00:00:99.

ARP reply		
0E:0E:0E:00:00:02		
0E:0E:0E:00:00:99		
0x0806	0x0001	0x0800
6	4	2 (reply)
0E:0E:0E:00:00:99		
192.168.1.3		
0E:0E:0E:00:00:02		
192.168.1.2		

- "Hacker" to "System B" (*blue line*):
  ARP reply packet saying 192.168.1.2 is at 0E:0E:0E:00:00:99.

ARP reply		
0E:0E:0E:00:00:03		
0E:0E:0E:00:00:99		
0x0806	0x0001	0x0800
6	4	2 (reply)
0E:0E:0E:00:00:99		
192.168.1.2		
0E:0E:0E:00:00:03		
192.168.1.3		

Later on, if A needs to send information to B, it will query its ARP table to obtain the associated B MAC address, but instead of getting the real B's MAC address, "0x0E:0E:0E:00:00:03", the ARP table contains the fake information introduced by the previously described packet: 0E:0E:0E:00:00:99.

The same situation occurs at B, so also the target system will redirect its network traffic to the attacker's system, believing it is the destination system, and only because attacker's system has asked them politely to do so ;-).

The attack is based in forging the other end system's identity at the layer 2 level, that is, at the physical or MAC addresses place.

In case the attacker wants to intercept the communication between a system placed in its local network, A, and a remote system, it only needs to substitute the destination target system (previously B) by the local network router. All communication between local system, A, and a remote destination system will need to travel through the local router. So this time A will think that its local router is the "Hacker" system and the router will think that A is the "Hacker" system.

A remote system will not be conscious at all about the redirection. From its point of view, all traffic is coming directly from A through out all the intermediate routers, and there is almost (see "TTL signature") no way for it to know the attacker is in the middle.

There are several free available tools to develop this attack. See "ARP spoofing tools" section.

Description and diagram of the attack

The attacker can poison the systems involved in a given communication (see Figure 2.12) in several ways, but all three combinations have value from the attacker's side in order to retrieve important information:

- Poisoning both systems, client and server, so every packet interchanged between them will be forwarded to the attacker's host. This is the most powerful attack from the attacker's perspective.
- Poisoning the client, so only the traffic from client to server will be redirected to the attacker's host. Typically client requests include usernames and passwords, commands to be executed and confidential information owned by the users. This and the next one can be considered as half-duplex poisoning attacks.
- Poisoning the server, so only the traffic from server to client will be redirected to the attacker's host. Typically the server responds with the confidential data associated to a given service.

Apart from that, the attack can be focused on two specific targets or all systems in the LAN can be poisoned, using unicast or broadcast packets.

We will focus on the attack details from a Linux operating system, because there are multiple tools to craft ARP packets and deliver ARP spoofing attacks, and it is possible to get a very deep control of the forwarding capabilities built-in inside the kernel. Therefore, lets suppose the attacker owns a Linux (kernel 2.4) box in the local network.

There are two main considerations that an attacker needs to take into account to run a successful attack. See an example script in "ARP spoofing preparation script" section to enable both:

- IP forwarding: once both systems have been poisoned, the traffic interchanged between them is redirected to the attacker's system. If this host has not enabled some forwarding capability, the traffic will be blocked after being received because it is not addressed to itself (destination IP address is one of the target hosts) and won't be deliver to the final destination.

  There are two general ways of enabling IP forwarding:
  - At the kernel level: in Linux activating the "ip_forward" TCP/IP stack kernel parameter is enough:

    ```
    # echo 1 > /proc/sys/net/ipv4/ip_forward
    ```

  - At the user level: through some kind of forwarding tool, as "fragrouter" [FRAG1].

- ICMP redirects: the attacker must disable ICMP redirects packets on the attack host to disturb the network as less as possible.
  This type of ICMP traffic is used by forwarding devices, typically routers but could be the Linux box with IP forwarding functionality, to inform some hosts about a more efficient network path to route packets through.

  When running an ARP attack, all three systems involved are placed in the same LAN, the two targets and the attacker's box. When the attacker's system receives the redirected traffic addressed to the other target it sees that both, source and destination are in the same network, so there is a more efficient way for the source host to send the packets to the destination rather than sending them through itself: sending the traffic directly between them.
  So, unless otherwise indicated, the attacker's system will politely inform the source system that it will be better (more efficient in routing terms) to send traffic to destination directly instead of to itself.

  To avoid this behaviour, this ICMP packet generation must be suppressed, using two TCP/IP stack kernel parameter, "send_redirects" and "secure_redirects":

```
# echo 0 > /proc/sys/net/ipv4/conf/*/secure_redirects
# echo 0 > /proc/sys/net/ipv4/conf/*/send_redirects
```

  The Linux "iptables" module can also be used for the same purpose:

```
# iptables -A OUTPUT -p icmp --icmp-type host-redirect -j DROP
```

An additional consideration in a smart ARP attack is determining the target OS as the first step. This information will be of high value to handle behaviour exceptions. Other tools as "nmap" or banner grabbing methods can be used to extract this data.

Trying to complement this study it must be clarified that most of the common OS in use nowadays are based in the Mentat TCP/IP stack implementation [MENTA1].

Mentat TCP is high-performance, fully compliant implementation of the TCP/IP protocol suite offering IPv6, IPsec, and other functionality. It is available as a source code product designed for easy integration into any operating environment, therefore it has become the leading portable implementation of TCP/IP. Mentat TCP can be found as an integral component of many computer and real-time operating systems including Hewlett-Packard HP-UX and Apple Mac OS. Originally written for Sun Microsystems, Mentat TCP forms the base of the native TCP/IP stack on Solaris. In addition, Mentat TCP has been ported to Linux, Microsoft Windows NT, Wind River VxWorks, SCO UnixWare, and many other OS.

How can the attacker verify if the attack was successful?

Once the attack has being launched, the attacker will need to check what the result was. There are different techniques that can be used for this purpose:

- Sniffing network traffic: When a given system sets its network card in promiscuous mode in a switched environment, it can only visualize the following traffic:
  - Unicast traffic addressed to it.
  - Broadcast traffic.
  - Multicast traffic bound to it, for example, addressed to all hosts or all routers multicast addresses.

  This is a similar scenario to sniffing in a hub environment in non-promiscuous mode.

  There is a frequent exception associated to the expiration time of the switches CAM table (see "Switches: advanced network devices" section). When a CAM entry is removed, the association between this MAC address and its switch port is momentarily removed, so in order to send traffic to this host the switch needs to act as a hub and send the next packet addressed to the host through out all its ports, until a new related entry is learn.

  If the attack was successful, the traffic interchange between the source and the destination will be captured.

- SNMP: Another helpful protocol for the attacker is SNMP. If the attacker can obtain the target system SNMP read community he could get the system ARP tables remotely by executing a simple "snmpwalk" command:

```
# snmpwalk 192.168.1.100 public at.atTable.atEntry.atPhysAddress | more
at.atTable.atEntry.atPhysAddress.2.1.10.0.0.30 =  Hex: 00 08 02 A1 A1 B3
at.atTable.atEntry.atPhysAddress.2.1.10.0.0.32 =  Hex: 00 08 02 E6 4F 9C
at.atTable.atEntry.atPhysAddress.2.1.192.168.1.1 =  Hex: E0 E0 0C CA AC 03
at.atTable.atEntry.atPhysAddress.2.1.192.168.1.2 =  Hex: E0 E0 34 CA BB 23
at.atTable.atEntry.atPhysAddress.2.1.192.168.1.3 =  Hex: E0 E0 34 CA BC 23
at.atTable.atEntry.atPhysAddress.2.1.192.168.1.10 =  Hex: E8 E0 09 CA BD EA
...
```

  Getting the read community is not a complex task due to the fact that SNMPv2, the most commonly SNMP version used, doesn't use encryption. Besides, it is amazing how many production networks use the default read and write SNMP communities, "public" and "private".

  The "arpsnmp" tool included in "arpwatch" [ARPW1] uses this method to monitor an alert about ARP anomalies.

As a curiosity, the Windows OS contributes to the attacker's interests by providing its MAC address through the "nbtstat" NetBIOS command, not only to systems in the same LAN but to those asking from a remote network:

Figure 2.13. Windows "nbtstat" command information

ARP spoofing persistence

Once the ARP table of both targets has been successfully poisoned, the attacker's goal is to hold control as long as possible to be able to compromise the system using advanced attacks (see "Advanced attacks based on ARP Spoofing" section).

Different traffic types could affect the redirection in place. We are going to analyze the main aspects that could affect it but, as a rule of thumb, the attacker will need to continuously send crafted packets to keep on populating the target's ARP table with the desired information.

Let's suppose the attacker, H, has poisoned both systems, X and Y. See scenario from section "ARP packet taxonomy tests".

From	To	Traffic	Description
ARP traffic			
Z	FF	ARP request asking for IP [X or Y].	[X or Y] will reply to Z with a unicast packet, the other target won't hear it.
Y	FF	ARP request asking for IP X.	X will see it and will respond. H must win X to be successful [1].
Y	H	ARP request asking for IP X.	H must reply this Y unicast request with H.
X	FF	ARP request asking for IP Y.	Y will see it and will respond. H must win X to be successful [1].
X	H	ARP request asking for IP Y.	H must reply this X unicast request with H.
X, Y	H	ARP request asking for IP H.	H replies with its MAC.
X, Y	FF	ARP request asking for IP H.	H replies with its MAC.
X, Y	FF	Gratuitous ARP request.	Host announces its IP [X,Y] in its MAC [X,Y] [1].
X, Y	FF	ARP request asking for IP Z.	Other target sees an ARP request for another system [2].
IP traffic			
X, Y	IP_Y	IP broadcast traffic.	Other target sees it [3].

X, Y	IP_ M	IP multicast traffic.	Other target sees it [3]. Same problem as with broadcast if the belong to the same multicast group.

NOTE:
Z is another host located in the same LAN but not poisoned, R is the LAN router, "FF" is the MAC broadcast address, "IP_B" is the IP network broadcast address while "IP_M" is an IP multicast address.

[1] ARP reply race condition: based on the packet reception and the algorithm applied (first or last packet wins) the redirection could be interrupted. Moreover, when Y asks for X, X should learn Y MAC address because it is a broadcast request addressed to it.
[2] If the OS learns from ARP requests addressed to other systems, which is a common situation, it will see a mismatch between the traffic and its ARP table.
[3] There is a potential problem when the OS learns or refreshes ARP information from generic IP traffic. In this case it will see IP traffic with a different MAC address as the one it is in their ARP table.

When IP generic traffic originated in a different network is addressed to the target systems, the router (R) will take the role of another host, it a similar way to Z's.

To sum up, all broadcasted traffic (including multicast when hosts belong to the same group) that reflects the real target system MAC addresses could potentially affect the redirection caused by the ARP spoofing attack. The most typical disturbing packet is a broadcasted ARP request coming from one target asking for another host, case [2].

One of the main points of interest to avoid this is the ARP table's expiration timeouts.

If the poisoning thread has only been targeted over one end, to capture only the traffic going from X to Y, the situation can be more easily detected. Apart from the previously described traffic flows, new ones would affect redirection:

From	To	Traffic	Description
ARP traffic			
Y	X	ARP request asking for IP X.	H cannot see this traffic cause Y has not been poisoned. X will reply with its MAC[2].
IP traffic			
Y	X	IP traffic not redirected.	X will see real Y MAC in the packet [1].

NOTE:
Z is another host located in the same LAN but not poisoned, R is the LAN router, "FF" is the MAC broadcast address, "IP_B" is the IP network broadcast address while "IP_M" is an IP multicast address.

[1] If the OS is able to analize MAC address information in generic IP traffic, it could detect a mismatch between the traffic and its ARP table.

[2] Y, not poisoned, asks X directly for its MAC. X should learn Y's MAC (different from actual H's MAC) because it is a request addressed to it.

To sum up unidirectional poisoning, all previous bi-directional cases affect this situation, plus all the unicast ARP traffic from Y to X. These packets will let X know that a problem exists because there is a difference in the Y MAC address. Besides, H cannot control Y timeouts because the attacker is not poisoning Y.

Network citizens

In real-world situations, this attack pretends to redirect the traffic going back and forth between target systems to the hacker's box. All the systems, targets and attacker, belong to the same local area network, and typically to the same IP subnet, so there are also some other devices, the ones that conform the network itself through which all them communicate. These network devices can be hubs or switches. Apart, the target systems can be the final hosts directly connected to the LAN or a local router that allows the remote connection to the final destination system.

To summarize, we are going to classify all different network elements in two types:

- Target systems: the systems establishing a communication, client or server, addressed or not to them. In this category we have two subtypes: end hosts and routers.
- Network devices: the systems that conform the local network, used to redirect the traffic at layer 2 (Ethernet) between target systems, mainly hubs and switches.

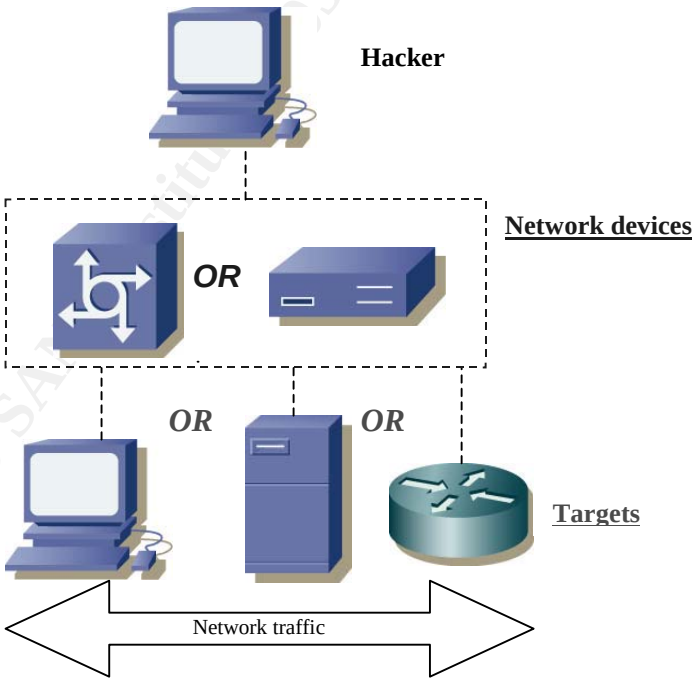


Figure 2.14. Network citizens

There is a special case in which hubs and switches can act as "target systems", not being considered as "network devices", when they are directly administered through a management connection. Commonly used protocols for this task are: ssh, telnet, ftp, tftp, rcp, snmp, ntp…

The attacker will reside in a target system, not having control of the network itself (network device), because in this case it will be "trivial" to redirect all the desired traffic to its box, for example, by using a switch monitoring (SPAN) port.

The goal of this paper is to cover as much different real world variants as possible, so we will need to research over different target systems and different network devices. The hacker's system will be always considered a Linux box (kernel 2.4) running the ARP poisoning tools.

The main idea is to cover as much operating systems as possible, including its different versions and distinct service packs or significant patch levels. See "Operating Systems" section for a detailed list of the different OS involved in this research and the interesting goals for future versions. In the same way, the same philosophy should be applied for network devices, to analyze its presence and how they influence ARP.

The OS used in the research include both, older version and variants in use these days, with the idea of covering both, old already fixed implementations and the latest infrastructure available in the field.

ARP spoofing tools

Arpplet

When this research paper was born, a small proof of concept code was developed, programmed in C language for the Linux platform. It was called "**arpplet**". There were two main goals in developing it:

- Learning raw socket programming and its application to the ARP protocol to better understand how the OS kernel generates ARP packets.

- Having a well know, not needing to inspect other tools source code, and simple tool. Simplicity was the main aspect desired: a tool was needed to simply generate ARP packets offering the chance of manipulating all the Ethernet and ARP fields, except the protocol types (remember we were interested only in Ethernet and IP protocols). Other additional functionality available in more advanced tools was not required, because unknown actions could alter the ARP behaviour analysis.

The "arpplet" source code has been included in "APPENDIX IX: "arpplet" source code" section. Its usage is very simple:

```
# arpplet -c 10 01:02:03:04:05:06 1.2.3.4 0708090A0B0C 5.6.7.8

 It sends an ethernet frame from 01:02:03:04:05:06 to 0708090A0B0C
 (both MAC address notations are valid) containing an ARP reply packet
 saying to 5.6.7.8 (0708090A0B0C) that 1.2.3.4 is at 01:02:03:04:05:06

 * ETHERNET/ARP PACKET: (42 bytes)

 * Ethernet Header: (14 bytes)
   - Destination address: 0708090A0B0C
   - Source address: 01:02:03:04:05:06
   - Frame type: 0x0806 (ARP)

 * ARP payload: (28 bytes)
   - Hardware type: 0x0001 (Ethernet)
   - Protocol type: 0x0800 (IP)
   - Hardware size: 6
   - Protocol size: 4
   - Op code: 2 (reply)
   - Sender MAC: 01:02:03:04:05:06
   - Sender IP: 1.2.3.4
   - Target MAC: 0708090A0B0C
   - Target IP: 5.6.7.8
```

Other tools available

There are several security tools that allow or are based on ARP spoofing methods. This section will simply reference them although a future research should analyze all them from the ARP perspective, checking what type of ARP packets they generate and how all they respond to different traffic stimulus.

arping	http://freshmeat.net/projects/arping/ http://www.habets.pp.se/synscan/programs.php?prog=arping
Broadcasts an ARP request packet on the network and prints answers. Very useful when you are trying to pick an unused IP for a net that you don't yet have routing to.	

arpspoof (dsniff)	http://www.monkey.org/~dugsong/dsniff/
arpspoof facilitates the interception of network traffic normally unavailable to an attacker (e.g, due to layer-2 switching)	

arptool	http://www.hotlink.com.br/users/lincoln/arptool (.C file) http://www.securityfocus.com/data/tools/arptool-0.0.1.tar.gz
With arptool you can send simple ARP packets, in this case just request and reply packets. This can be used together in a script to set up man-in-the-middle ARP-relaying or as denial of service attack. You can also reroute LAN traffic which would otherwise not be visible, to sniff and play dirty tricks on your machine.	

arpsend	http://www.net.princeton.edu/software/arpsend/
arpsend sends an IP ARP request or reply packet containing fields you specify. This is a diagnostic tool intended for use by network administrators.	

arp-fillup	http://www.althes.fr/ressources/tools/arp-fillup/README

This tool assist IP Smart Spoofing method populating the target host with all the valid IP-MAC address pairs of the systems in the network, in order to avoid the target broadcasting ARP request that could disturb the redirection attack. It uses unicast and broadcast ARP request and unicast ARP replies.

ettercap	http://ettercap.sourceforge.net/
Ettercap is a multipurpose sniffer/interceptor/logger for switched LAN. It supports active and passive dissection of many protocols (even ciphered ones) and includes many features for network and host analysis.	

hping2	http://www.hping.org/
hping is a command-line oriented TCP/IP packet assembler/analyzer. The interface is inspired to the ping(8) Unix command, but hping isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, the ability to send files between a covered channel, and many other features. hping3 will be released in December 2003.	

hunt	http://www.gncz.cz/kra/index.html (disappeared) http://packetstormsecurity.nl/sniffers/hunt/index.shtml
Hunt is a program for intruding into a TCP connection, watching it and resetting it. It can handle all connections it sees. Features: Connection Management - setting what connections you are interested in, detecting an ongoing connection (not only SYN started), Normal active hijacking with the detection of the ACK storm, ARP spoofed/Normal hijacking with the detection of successful ARP spoof, synchronization of the true client with the server after hijacking (so that the connection don't have to be reset), resetting connection, watching connection; Packet Engine - extensible packet engine for watching TCP, UDP, ICMP and ARP traffic, collecting TCP connections with sequence numbers and the ACK storm detection; Switched Environment - hosts on switched ports can be spoofed, sniffed and hijacked too; much, much more.	

Nemesis	http://www.packetfactory.net/projects/nemesis/
Nemesis is a command-line network packet injection utility for UNIX-like and Windows systems. You might think of it as an EZ-bake packet oven or a manually controlled IP stack. With Nemesis, it is possible to generate and transmit packets from the command line or from within a shell script. We are mainly interested in the "nemesis-arp" command.	

SNARP (Win)	http://packetstormsecurity.org/NT/snarp.zip
Snarp is a tool for NT 4.0 that uses an ARP poisoning attack to cause a host to redirect traffic to the attacking machine (the machine running Snarp), and thus allowing that host to sniff the data from the wire.	

Packit	http://packit.sf.net http://packit.sourceforge.net/
Packit (Packet toolkit) is a network auditing tool. Its value is derived from its ability to customize, inject, monitor, and manipulate IP traffic. By allowing you to define (spoof) nearly all TCP, UDP, ICMP, IP, ARP, RARP, and Ethernet header options, Packit can be useful in testing firewalls, intrusion detection/prevention systems, port scanning, simulating network traffic, and general TCP/IP auditing. Packit is also an excellent tool for learning TCP/IP. It has been successfully compiled and tested to run on FreeBSD, NetBSD, OpenBSD, MacOS X and Linux.	

Advanced attacks based on ARP Spoofing

The main goal for redirecting the traffic through ARP poisoning is being able to receive the network traffic interchanged between, at least, two hosts. These types of attacks are commonly referenced in the security area as "Man In The Middle Attacks", MITMA.

As a proof of concept code, "The Ultimate ARP Tool" could be developed: first of all, it will try to obtain the target operating system and based on this information will send the specific set of packets that for sure populate target system ARP table. To maintain the ARP table poisoned, the packets should be frequently sent based on the operating system timeout algorithms.

Sniffing

Once traffic is redirected, the attacker just needs to use a sniffer to capture all the data.

In section "How can the attacker verify if the attack was successful?" was analyzed what type of traffic is available in a standard switch environment and how it changes when the IP spoofing is successful.

Without having in mind the ARP poisoning attack, it is supposed that sniffing [SNIFF1] techniques are not possible in switched environments, because the attacker's system have access only to its own traffic and to the broadcast/multicast additional traffic. ARP poisoning changes this obsolete view, making possible to sniff traffic in any local network topology (hubs or switches).

Denial of Service

A DoS attack using ARP spoofing is a trivial task. The easiest way of isolating a host in a network is poisoning its ARP table with an unused forged MAC address.

Another possibility is redirecting all the traffic to the attacker's box to silently discard it without forwarding the packets. Linux example:

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

The Linux "iptables" module can also be used for this purpose [ARPSK1]:

```
# iptables –A FORWARD –s client –d server –j DROP
```

Transparent proxy

The attacker's host can act as a transparent proxy redirecting the traffic to the appropriate application, being able to modify at the application level any data transferred. This type of attack is also referred as session hijacking because a TCP connection can be owned and new traffic can be injected into it.

As an example, we will show how this method can be used to exploit a SSH vulnerability [SSHA1] based on a weak client implementation in the interoperability of the SSH versions, SSHv1 and SSHv2, and the flexibility associated to the server side. The exploit manipulates the initial SSH exchange, banners and keys, between client and server to impersonate the server.

To execute this exploit the attacker just needs to start a specially modified ssh daemon that executes the required steps, for example, in port 5000:

```
# sshd –4 –p 5000
```

The application level redirection must be implemented through the Linux redirection capabilities associated to the netfilter subsystem. Through the "iptables" netfilter user interface it is possible to specify that all traffic addressed to port 22, SSH, must be sent to a different port, let's say port 5000, where the evil sshd is listening:

```
# iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT \
                                      --to-port 5000 -i eth0
```

Then, using ARP spoofing the attacker will redirect all traffic from client to server to its box.

If the client originates a TCP connection to port 22, the ARP poisoned table will help to send the packets to the hacker's system and once received, the "iptables" module will redirect the packets to port 5000. The client will think he has established a connection with server in port 22 while in fact he has connected to hacker box at port 5000, where the evil sshd is waiting…

Smart IP spoofing

A new method has been presented [SMART1] in which it is possible to apply IP spoofing methods over any networking application. This attack exposes the commonly used filtering devices based on IP addresses, as firewalls.

As expected, this attack uses ARP spoofing, redirecting traffic between client and server. In this case, the attacker's goal is to spoof the client's IP address to bypass the security controls in the server side that only allow client connections. In this type of attack the attacker's box can be located in any intermediate network between the client and the server.

To impersonate the client at the IP level the attacker needs to use Source NAT, a method that allow setting the source IP address in the IP packets to a specific value, in this case, the client IP.

Again, using the "iptables" Linux module, the attacker could implement Source NAT. The Linux SNAT module will differentiate attacker's connections from client's ones automatically:

```
# iptables -t nat -A POSTROUTING -o eth0 -d server -j SNAT --to client
```

This method allows the use of any application to connect to the server using the client's IP address. Besides the existing traffic between the client and the server won't be perturbed. From the server side this is an undetectable attack, not being possible detect the real client from the evil box.

ARP protocol security research

For an administrator to be able to protect his network infrastructure and to defend against the ARP attacks and security vulnerabilities, he will need to understand every detail about how ARP and its implementations work. Helping him to acquire this knowledge is one of the goals of this paper.

For this purpose the document tries to sets the foundation of a group of tests that will allow getting as much information as possible about ARP security concerns. Thus, the following items should be analyzed:

- The ARP table behaviour based on different traffic stimulus. These stimulus consist of ARP packets generation modifying different packet fields. A packet taxonomy has been designed to cover all the meaningful possibilities.
- ARP table timeouts analysis: find out how they work, their default values, how to verify them, how to set/modify them, etc. per OS.
- Big anomalies in ARP packets: addresses length and spaces values.

By means of the above research, we'll try to test as much questions [*Qn*] arose along the paper as possible.

Additional analysis will be presented in this section, such as:

- OS fingerprinting based on the ARP traffic: analyzing how does every studied OS generate the ARP packets, both, requests and replies.
- Minimum packet size enforcement: analyze how every OS enforce this theoretical requirement.
- Using real or fake MAC addresses: pros and cons.
- ARP packets in bootstrapping and shutdowning processes.
- ARP packets when activating or shutting down interfaces.
- Analysis of the "arp" command, brief overview of the different options available in every analized operating system, and manual ARP table modification: how an OS behave when manually adding, modifying, removing ARP entries.

ARP packet taxonomy: analyzing all ARP packet variations

The designed packet taxonomy is based on a set of variables whose values define different types of ARP packets. By changing their values we intend to determine which packets allow the insertion of a forged entry in the ARP cache table for the OS.

It is important to remark that the impact of a change depends not only on the values being modified, but also on the previous state of the ARP cache; thus the same packet will provoke a different result upon distinct start points.

ARP packet variables

- Is it an ARP request or reply?
  - If it is a request, what is the value of the "Target HW Address" field? [*Q1*]
  - If it is a reply:
    - Does the ARP reply come after sending an associated ARP request? Remember that ARP is "stateless", so the cache table is updated when a response is received, not matter if a request has been previously sent on purpose.
    - Was it sent without a precedent corresponding response?

- Does the ARP cache have already a dynamic, previously learned entry associated to the IP address, obviously with a different MAC address value (probably the real one)? [*Q4*]
  - Without having a previous entry for the same IP address in the ARP cache table: new entry creation.
    It is supposed, based on RFC 826, that receiving an ARP reply associated to an inexistent previous entry should not update the ARP table.
  - Having an existing entry for the IP address: existent entry update.

  In the "ARP table and interface variables" section we will analyze what happens when the entry was statically created by the administrator instead of having been dynamically learned.

- Is the packet forwarded to a broadcast MAC address or to a unicast one? [*Q5*] What if it is being forwarded to a multicast MAC address?

  What are the MAC addresses inside the ARP packet? [*Q6*]

  a) Analyzing the ARP "op_code" field:
  - It is also a broadcast value for both, request/reply.
  - It is also a unicast value for both, request/reply.

  RFC 1812 states that routers must not trust ARP replies claiming to have a broadcast or multicast MAC address [*Q13*]. Check both targets, routers and hosts.

  b) Analyzing the ARP addresses field's contents:
  - It is addressed to the host MAC address?
  - It is addressed to another host different MAC address?

  If it is an Ethernet unicast frame, the MAC address in the Ethernet header is the one associated to the host, but:
  - Is the destination IP address in the ARP packet the host IP address?

  - Is the destination a different IP address belonging to another system?

  If it is an Ethernet broadcast frame, but targeted to only one IP address, should all host except the one the frame was addressed to (and those with its network interfaces in promiscuous mode) discard it? This is the typical ARP request packet: addressed to all at the Ethernet level (MAC broadcast address), asking just for one specific IP address.

  In a future revision it will be analyzed how each OS behaves when the network interface is in promiscuous mode: does it process the frames not intended to itself?

- Is the ARP packet a gratuitous ARP message (the same IP address for source and destination has been embedded into the ARP packet) or not?

- How does a given implementation behave if the two MAC addresses in an ARP request ("Sender HW Address" inside the ARP packet and "Sender MAC Address" in Ethernet header) are different? The ARP reply can be influenced [*Q11*].

ARP table and interface variables

The following ARP variables are related to the ARP table and the timeouts used to manage it, and to the network interfaces state:

- From the operating system point of view, we need to determine which ARP packet will be kept in the ARP cache table. The two possible options are:
  - The first packet to arrive will be kept.
  - The most recently received packet will win. This is the option defined in the ARP RFC.

  This test is referenced as the "Who wins?" test, checking which packet, the first or the last, wins and remains in the ARP table.

- The timeout that controls if an entry is kept in the cache could be influenced by its usage:
  - If it is used, is the timeout reset? Under what circumstances?
  - Although used, the timeout is a fixed value from the moment it was inserted in the ARP cache.

  Test if any OS uses recheck or validation ARP packets. These are ARP request packets asking for one IP address and addressed to the associated, already known MAC address, instead of to the standard MAC broadcast address. This is known as the "Unicast Poll" method.

- If and static ARP entry exists in the table, does the incoming ARP packet overwrite this entry?

What about if the interface is not supposed to be listening to ARP packets? In Unix systems (it doesn't apply to Windows boxes) this can be configured using the "ifconfig" command with the "-arp" option [YURI2]. This ARP suppression mechanism prevents the host from sending any ARP requests or responding with any ARP replies.

> "Comment applies to "IRIX 6.2, HP-UX or FreeBSD:
>
> It seems that operating systems update their table with new ARP information even when they had a static ARP entry associated to the same IP address. The static entry can be registered using the "-s" or "-f" arp options).
>
> Theoretically, the "-arp" ifconfig option defines that a network interface will not listen to ARP packets, but it seems that, used or not, the new ARP packets that refer to an IP address not set in the ARP table are also added."

This behaviour must be analyzed to reach a definitive conclusion about when "static" entries are updated.

ARP big anomalies

The following variations have been considered as big ARP packet anomalies and, although described under this section, they will be studied using a different test set from the one used for the ARP packet taxonomy classification.

They affect address length and spaces (or domains) inside the ARP packet:

- Is the HW length field modified (different from 6) in the ARP packet? [*Q2*] Is this value checked somehow? [*Q9*]
- Is the protocol length field modified (different from 4) in the ARP packet? [*Q3*] Is this value checked somehow? [*Q9*]
- Modified HW address space value. The value for Ethernet networks is always 1. [*Q7*]
- Modified Protocol address space value. The value for IP protocol is always 0x0800. [*Q8*]

ARP packet taxonomy tests

This section defines the whole set of tests according to all the ARP packet and table variables described before, called "Packet Taxonomy Tests". The tests design and goals will be explained. It constitutes one of the main elements of this research. ARP big anomalies tests have been developed as a separate set.

The taxonomy has been created based on the following scenario:

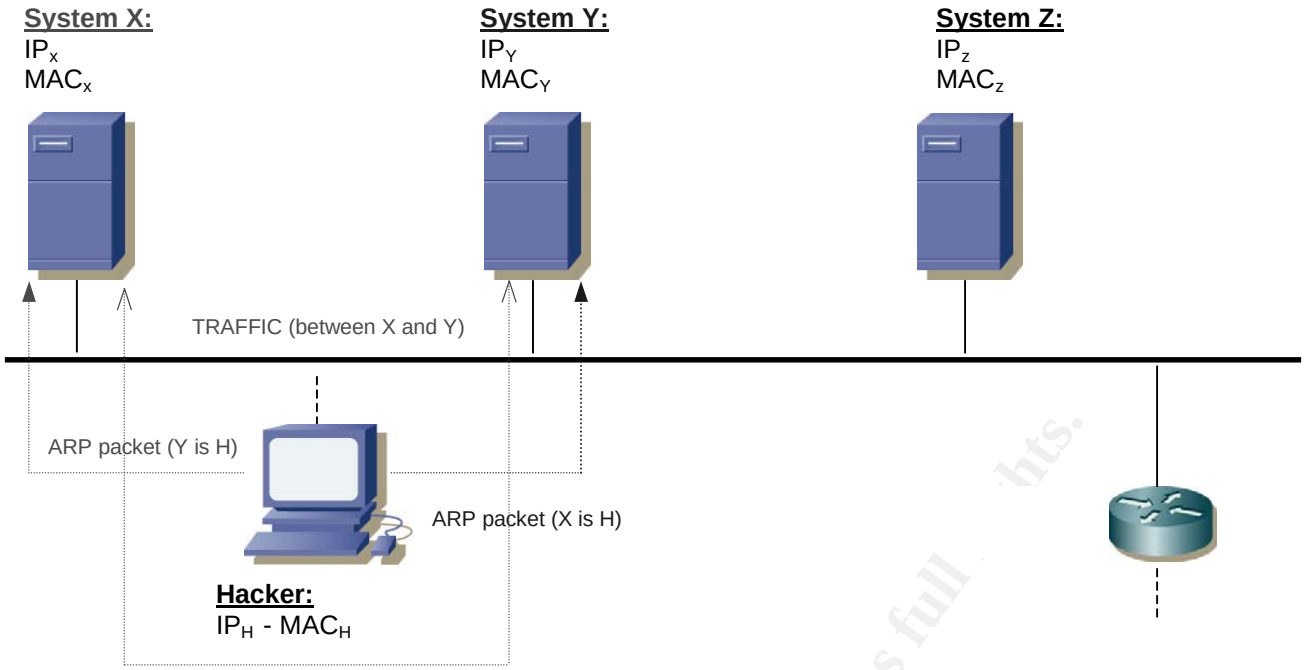
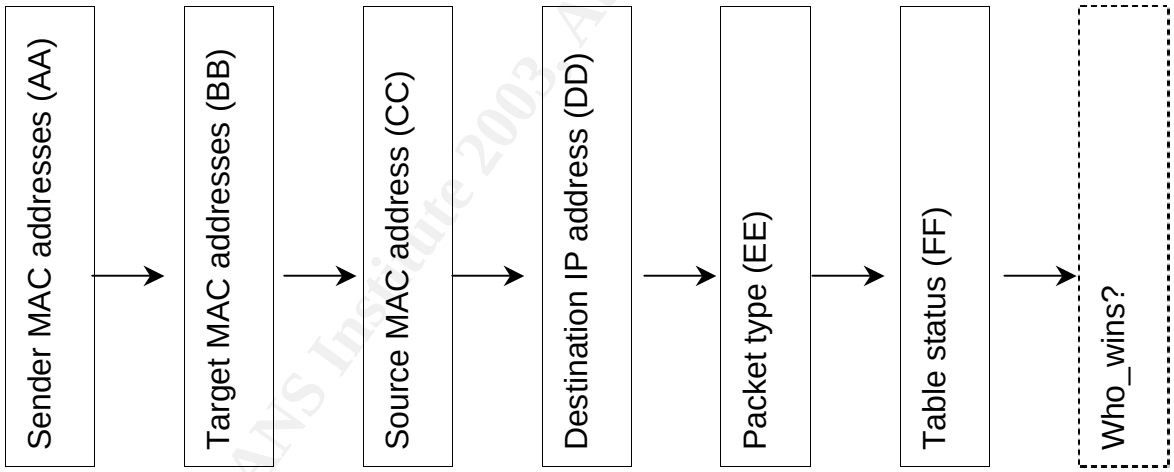


Figure 2.15. Generic networking scenario for ARP spoofing

An ARP packet can be identified based on all the different fields that both, Ethernet and ARP protocols define. A classification has been developed based on these fields:



The taxonomy has been designed with the idea of covering all the potentially relevant packets that could poison the target's ARP table. To do so a proof of concept design and code (see "ARP packet taxonomy scripts" section) were created. Every packet analyzed must have a specific value for any of the variables showed in the previous flow.

The main idea behind the Packet Taxonomy Test is being able to identify and classify every ARP packet that can be sent to a host, being the variations not only the packet fields but also the table and interface elements mentioned

before. The focus is centered in having a way of testing how a device behaves when a packet of a specific type is received.

For this reason every packet sent with a specific table and interface status in the target host has been identified through an ID, called "Test ID".

The "Test ID" value is conformed by all the six variables or steps involved in the packet reconstruction described before. The "who wins" is an optional step not influencing the "Test ID" value. Be careful not to confuse the "Test ID" with a MAC address value ;-):

TEST ID:
AA.BB.CC.DD.EE.FF

Every designed test should have a correspondent "Test ID", allowing it to be tested independently.

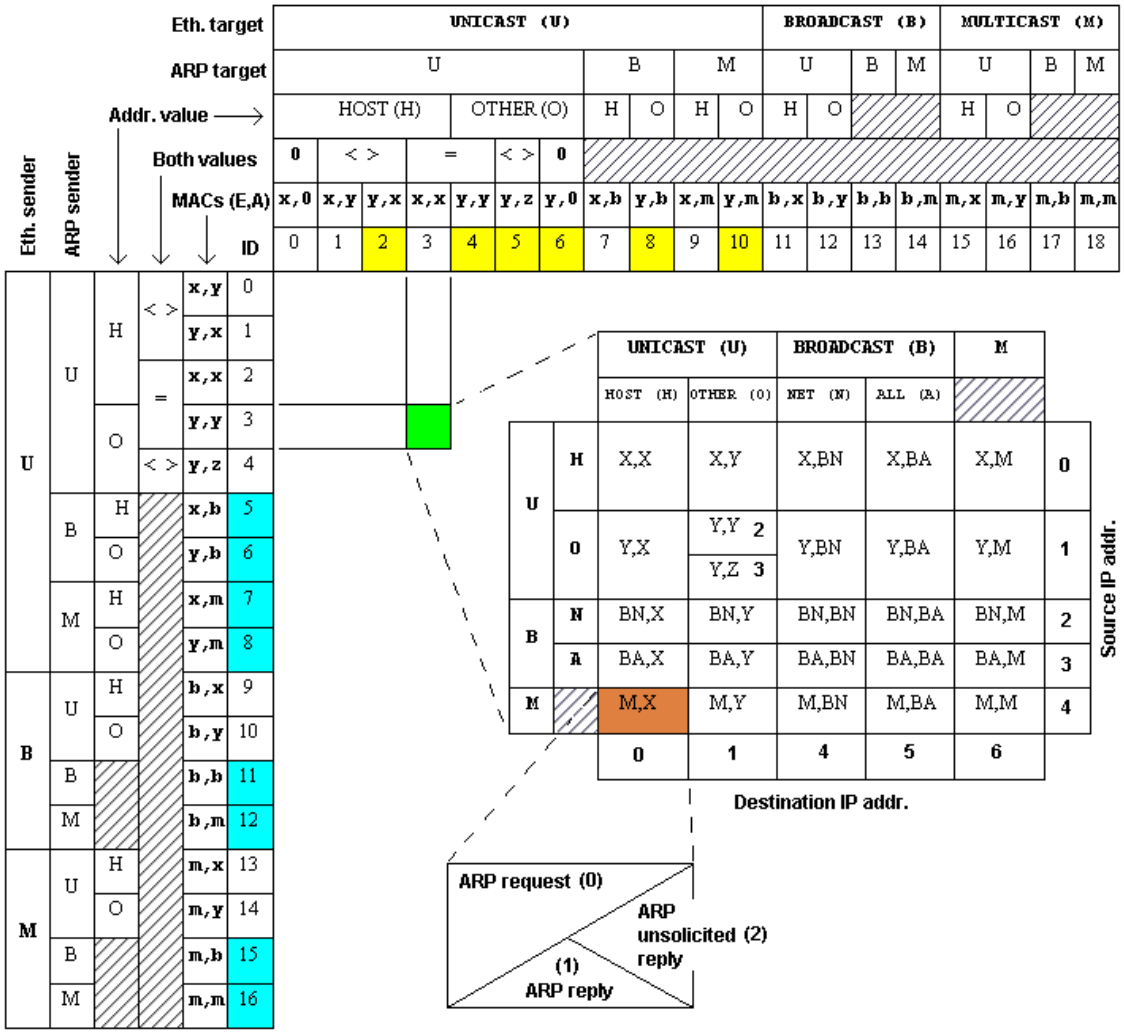


Figure 2.16. Packet Taxonomy Tests - packet variables

The figure shows how an ARP packet can be classified. The left column determines the values of the "Sender MAC Address" at the Ethernet header and

ARP packet, while the row on the top determines the "Target MAC Address" in both headers.

Both sender's MAC address can be unicast (U), broadcast (B) or multicast (M), and if they are unicast, they can have the target host MAC address (Host) or the MAC address of another system (Other). If both addresses, Ethernet and ARP header, are unicast, then they can have the same or a distinct value. All these combinations assign the ID for the "Sender MAC Addresses" between the 17 different possibilities presented. The "Sender MAC Addresses" in the ARP packet is typically the field used to learn new IP-MAC associations, therefore the most important field for this research.

The same reasoning is applied to the target's MAC addresses: they can be unicast (U), broadcast (B) or multicast (M); they can have the target host MAC (Host) or the MAC of another system (Other); and they can have the same or a distinct value.

The unicast values are driven by the hosts specified in Figure 2.15. "X" is the target system, while "Y" is the other end it is communicating with, probably another target in a real life scenario. "Z" is another system not involved in the ARP redirection games, it can be a host or the subnet router. "H" s the hacker's system from where the tests are launched.

An additional combination has been introduced giving the "Target MAC address" field inside the ARP packet the possibility of having a zero value. All these combinations assign the ID for the "Target MAC Addresses" between the 19 different possibilities presented.

Once the MAC addresses values have been established, a specific cell in the main matrix is selected. Then the second table denotes the values of the IP addresses in the ARP packet. The left column shows the "Source IP address" while the row on the top shows the "Destination IP address".

The IP addresses values can also be unicast (U), broadcast (B) or multicast (M):

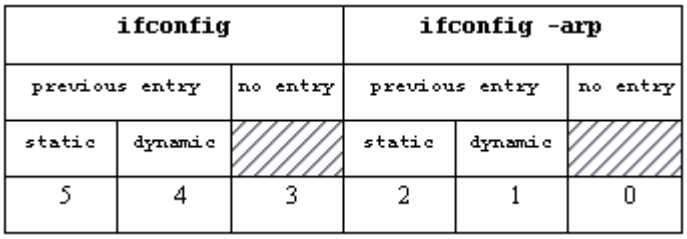
- If they are unicast they can have the target host IP address (Host) or the IP address of another system (Other).
- If they are broadcast, they can have the IP subnet broadcast value (Net), for instance, 192.168.1.255, or the global IP broadcast, 255.255.255.255 (All).
- If they are multicast only 1 address have been considered: 224.0.0.0 the base address for all devices.

All the combinations provide 5 options for the source IP and 7 for the destination IP. Due to the fact that the destination branch check if both, source and destination, have the same or a different value, this checking only applies to unicast addresses, so both IP steps generate a total of 26 possibilities.

Again, when the IP addresses values are known, a specific cell is highlighted, and then the packet type should be indicated. There are only 3 possible types:

packet can be an ARP request or an ARP reply, previously requested (solicited) or not (unsolicited).

Once the packet characteristics have been selected, the ARP table status of the target system must be analyzed. The following figure shows its possible states:



ifconfig			ifconfig -arp		
previous entry		no entry	previous entry		no entry
static	dynamic		static	dynamic	
5	4	3	2	1	0

Figure 2.17. Packet Taxonomy Tests - table status

The target network interface can be listening for ARP (ifconfig) or not (ifconfig –arp). Both cases are divided in three subcases related to the ARP table, providing 6 additional combinations:

- A previous entry didn't exist.
- A previous entry existed: it was created dynamic or statically.

The dynamic test can also be used to analyzed the "Who wins?" scenario, because two ARP packets are sent, being possible to check what information prevalence in the ARP table.

The designed packet taxonomy combines lots of different possibilities:

Total number of packet and state combinations:
AA=17*BB=19*(CC+DD=26)*EE=3*FF=6

To sum up, all the flow steps combined provide a total of **151164** possibilities, that is, different ARP packets, including the host state, that should be tested against any host that is going to be audited based on this research.

Then, for instance, to see if the target system is influenced by the reception of a standard ARP request, the following test must be selected:

TEST ID:
03.13.01.00.00.00

ARP request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr. (Y)		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr. (Y)		
Sender IP addr. (Y)		
FF:FF:FF:FF:FF:FF		
Target IP addr. (X)		

- Both "Sender MAC Addresses" must have the same value , being the MAC address of the other end (Y). {AA=03}.

- "Target MAC Addresses" in the Ethernet header must be broadcast, and "Target MAC Addresses" in the ARP packet must be all ones. {BB=13}.
- "Source IP Addresses" must be other end IP (Y). {CC=01}
- "Destination IP Addresses" must be target's IP (X). {DD=00}
- "Packet Type" is a request. {EE=00}
- The "Table Status" could have no entry associated having the network interface in normal mode, listening ARP. {FF=00}

There are some special cases that should be highlighted:

- "Sender MAC Addresses", blue colored tests (AA = 5,6,7,8,11,12,15,16), shouldn't be processed by target system, based on RFC 1812, because ARP "Sender MAC Address" claims to be broadcast or multicast. Although this RFC refers to routers, it should be applied to both routers and hosts.

- "Target MAC Addresses", yellow colored tests (BB = 2,4,5,6,8,10), shouldn't be processed by target system because the Ethernet header "Destination MAC address" is not the target host. There is a possibility in which the target host could have its network card configured in promiscuous mode and accept and process this packets. Promiscuous mode will be treated in a future revision. All proposed test refers to the interfaces in normal mode, not promiscuous.

To be able to run all these different tests a proposed research lab has been described in "APPENDIX II: Research lab description" section.

Conclusions

Due to the huge number of possible combinations, 151164, this document tests have been focused only over a very specific reduce set. The set was selected influenced by the only two previous ARP packet classification research found: [BH2001] and [ARPSK1], being the "ARP-SK" project the most complete. See "ARP packet taxonomy scripts" section.

All the tests have been focused on systems whose network interface is listening for ARP packets. On the one hand because it is the normal way systems work, and on the other hand, because not all OS can avoid this behaviour, only Unix variants can.

The following systems from the set considered in this research have been selected, with the idea of covering both, old and new versions and covering different manufacturers, and showing a proof of concept about how the proposed methodology could be used:

- Cisco IOS router 12.0
- HP-UX 10.20
- Linux: kernel 2.4
- Windows 2000 SP3
- Solaris 8

Test BH

First set of tests is focused on checking if target favors Ethernet "Source MAC Address" or ARP "Source MAC Address". See "Tests BH" section. Packets are like unicast request addressed to target but crafting "Sender MAC addresses".

Cisco IOS 12.0
It doesn't populate its table with packets with different Ethernet and ARP "Sender MAC Addresses". It enforces this supposed reasonable condition.

HP-UX 10.20
It doesn't populate the table when the ARP "Sender MAC Address" is its MAC address. Otherwise, this ARP value is the one chosen.

Linux kernel 2.4 and Windows 2000 SP3
It always populates its table with the ARP "Sender MAC Address".

Solaris 8
If packets seem to come from itself, it doesn't populate the table, but it they come from another system it does. It always introduces the "Sender MAC address" of the ARP header.

Summary

OS	Favors Ethernet or ARP
Cisco IOS router 12.0	N/A
HP-UX 10.20	ARP
Linux: kernel 2.4	ARP
Windows 2000 SP3	ARP
Solaris 8	ARP

This means that for these OS, an attacker can forge only the sender's MAC address in the ARP packet, being able to avoid access controls based in the Ethernet source address; he can set an spoofed MAC address there. This will confuse switches, and the CAM table won't help on detecting where the fake MAC address in system's ARP tables is.

Test SK

Second set of tests is focused on checking all the potentially valid packets from the MAC addresses perspective, trying all different unicast IP addresses combinations for ARP request and unsolicited replies. All the ARP table status should be checked. See "Test SK" section. Packets are like unicast request addressed to target but crafting "Sender MAC addresses".

Cisco IOS 12.0
If it sees an Ethernet/ARP packet coming from his MAC address it is discarded.
The table is populated with ARP reply packets coming from another host MAC address, where its MAC address is in the "Target MAC Address" in the ARP packet and both IP addresses are its IP address (gratuitous packet). A very concrete case that applies to new and dynamically learnt entries.

ARP reply		
Host MAC addr. or FF:FF:FF:FF:FF		
Another MAC addr.		
0x0806	0x0001	0x0800
6	4	2 (reply)
Sender: Another MAC addr.		
Sender: Host IP addr.		
Target: Host MAC addr.		
Target: Host IP addr.		

Static entries are never overwritten.

It detects duplicate IP address belonging to it when a standard ARP request coming from its IP address is received and the "Source MAC address" is its MAC address. When packets come from another MAC, duplicates are detected in these cases:

- There is no previous entry or there is a static entry, and the packet is an ARP request coming from and to its IP address (gratuitous ARP request).
- An standard ARP Request coming from its IP address and:
  - Addressed to its IP if it is a reply packet.
  - Addressed to another host IP if it is a request or reply packet.

HP-UX 10.20

If it sees an Ethernet/ARP packet coming from his MAC address it is discarded, same way Cisco IOS does. Also, if the Source IP address is its address, the packet never affects the ARP table.

Then, the table is only populated in the following cases:

- New entries are created only though ARP unicast or broadcast requests, coming from an IP address belonging to another host and addressed to it.
- Previously existent entries, no matter if they were learnt dynamic or statically, are always refreshed when packets specify another host IP address.
  Packet type and destination IP address don't affect this populating process at all.

No duplication detection found. Instead a bug was found, see ""Duplicate IP address" message" section.

Linux kernel 2.4

Linux populates its table in both cases, when the packet comes from its MAC address or from another MAC address.

To create new entries it is needed that the "Source IP Address" has the value of another host IP. "Destination IP Address" can be its IP or another IP address, it doesn't affect the table population for new entries.

ARP requests coming from another system IP address and addressed to the host IP also updates previously learnt dynamic entries.

Static entries are never overwritten.

Windows 2000 SP3
New entries are created independently of the "Source MAC Addresses" if packet comes from another host IP address to its IP address.

Dynamically learned entries are always overwritten when the ARP packet comes from another host IP address, no matter what the destination IP address is.

Finally, static entries are overwritten with the Ethernet "Sender MAC Address" if the packet comes from another host IP address. But there are some specific exceptions:

- Packets coming from another host MAC address being target's MAC its MAC address in the ARP packet:
  - A directed request that comes from IP address Y to Z.
  - A broadcasted request that comes from IP address Y to Y.
  - A broadcasted reply that comes from IP address Y to Z.

- Packets coming from its MAC address to itself, it doesn't matter if they are unicast or broadcast, from another host IP address to another host IP too:
  - If the two IP addresses are the same, gratuitous, a reply doesn't populate the table but the request is successful.
  - If they are different IP addresses, nor request neither replies populate the table.

Duplicate address messages are generated when the "Sender MAC Address" belongs to another host but the "Source IP address" is its IP address.

Solaris 8
Packets coming from its MAC address don't influence the table.

Packets coming from another host MAC address always populate the ARP table, no matter the source or destination IP address values neither if packets are requests or replies, nor the table status (in the three cases it is successfully compromised).

The only exception is when MAC destination is broadcast address in Ethernet and ARP headers, the source IP belongs to another host and packet is addressed to its IP address. Both, request and reply fail in populating the table.

Solaris detects duplicates IP addresses mainly through ARP replies packets where the "Source IP Address" is its IP address. Request don't seem to generate a syslog alert.

Summary

Due to the complexity associated to these tests it is not possible to show in a single table all the results obtained, so a detailed description per OS has been provided. Only the static entries analysis has been summarized:

OS	Static entries
Cisco IOS router 12.0	No
HP-UX 10.20	Yes
Linux: kernel 2.4	No
Windows 2000 SP3	Yes
Solaris 8	No

Under some circumstances all OS except Cisco IOS and Linux allow overwriting ARP static entries.

Future improvements

A future improved "Packet Taxonomy Test" classification should reflect additional values for some fields:

- "Destination MAC address" should analyze the case in which the "Target MAC address" in the ARP packet is zero when the "Target MAC Address" in the Ethernet header has a broadcast or multicast value, not only when it is unicast.
- "Source and Destination IP Addresses": when a multicast value is configured, additional multicast addresses must be analyzed, such as, 224.0.0.1 (all hosts in this subnet) or 224.0.0.2 (all routers on this subnet) or a specific multicast the target host is associated with.

ARP big anomalies tests

The variables involved in this test were defined in the "ARP packet taxonomy: analyzing all ARP packet variations" section, and apply to the first four fields of the ARP packet: Hardware and Network addresses lengths and Hardware and Network Protocols.

None of the ten operating systems analyzed produce any kind of response against this stimulus. The packets were silently discarded.

To run these tests a simple proposed research lab has been described in "APPENDIX II: Research lab description" section.

The initial tests only cover the length fields because this paper is focused in Ethernet and IP technologies, so protocol fields have a fixed value. It is supposed that OS should discard not available protocols in their stack, but due to the broad research nature of this paper, a future version should test these two fields too ("arpplet" tool should be modified to include new parameters for them). This is out of the current scope of this research.

ARP timeouts: analyzing the ARP cache table

To completely understand how ARP really works it is crucial to get detailed information about the different OS configurable timers used to operate with ARP and manage the ARP table. These timers and options involved are:

- [T1] ARP cache table entry persistence: this timer determines the timeframe an entry will be kept into the OS table [*Q12*]. Both ARP packet types should be checked: ARP request [T1a] and reply [T1b].
  Besides, once one of these two packets has been received:
  - It is possible this timer will have a different behaviour based on the entry usage, that is, it could be reset every time the entry is reused [T1ra, T1rb] when sending outbound traffic or…
  - It could be never reset; after being initialized the first time a new entry is registered in the ARP table that will keep decrementing until it reaches a value of zero, regardless of the traffic [T1-fixed].

  An entry is referenced (Windows terminology) or reused with any outbound packet to the IP address contained in it. Some operating systems could implement two different timers, one associated to reused entries and other one used for non referenced ones.

- [T2] ARP cache table entry blocking timeout: this timer determines the timeframe an entry recently registered in the ARP cache will be blocked for being updated when receiving a new ARP packet associated to the same IP address. For example, Solaris works this way.
  The entry could be learned due to an ARP request [T2a] or reply [T2b], and the packet received in the second place can also be an ARP request [T2xa] or reply [T2xb].

- [T3] What timeout algorithm (as stated in RFC 826) seems to be used by the operating system? [*Q10*]
  - Option 1: resetting the timeout when different inbound traffic is received:
    - ARP [T3a]:
      - Request [T3aa]
      - Reply [T3ab]
    - IP (ICMP, UDP or TCP) [T3b].
  - Option 2: reconfirm the validity of the entry by sending Unicast ARP requests, "Unicast Poll" method [T3-poll].

Every different timer and option has been given a timer identifier, [TNxy], that will be referenced in the test that analyzes it (see "ARP timeouts tests" section).

Timeout	**Packet that creates/ed the ARP entry (1):**		**Test**
T1	ARP request [T1a]		TestTR1
	ARP reply [T1b]		*TestTL1*
T1 reused [T1r]	ARP request [T1ra]		*TestTL2*
	ARP reply [T1rb]		*TestTL3*
Timeout	**First packet (1):**	**Second packet:**	**Test**
T2	ARP request [T2a]	ARP request [T2aa]	TestTR5
		ARP reply [T2ab]	TestTR6
	ARP reply [T2b]	ARP request [T2ba]	TestTR7
		ARP reply [T2bb]	TestTR8
Timeout	**Traffic type:**	**Subtype:**	**Test**
T3	IP [T3b]	ICMP, TCP, UDP	TestTR2

	ARP [T3a]	ARP request [T3aa]	TestTR3
		ARP reply [T3ab]	TestTR4

Figure 2.18. Different ARP table timeouts and options summary

Entries manually added are static and are not automatically removed from the ARP cache table, whereas dynamic entries are removed from the table. So ARP table timeouts don't apply to the first set.

ARP timeouts tests

A proposed research lab has been described in "APPENDIX II: Research lab description" section to help as much as possible with the information retrieval and data correlation. The diagram displayed in this section will be used to describe the different tests.

Two types of tests need to be carried on:

- Local tests: (outbound traffic) traffic is generated from the target system to see how it affects its ARP local table. Identified by "Test Timeout Local n" or "TestTLn".
- Remote tests: (inbound traffic) traffic is generated from the analyzer's (remote) system to see how it affects the target system ARP table. Identified by "Test Timeout Remote n" or "TestTRn".

This research mainly analyzes normal network traffic generated from and addressed to the target system, that is, without crafting packets. Only in those situations where normal packets are not possible, crafted packets have been used.

In a future research it will be interesting to analyze the ARP timeout behaviour with all the crafted packets described in the "ARP packet taxonomy: analyzing all ARP packet variations" section. For example, broadcasted ARP requests asking for someone else's IP address, which are typical in network traffic, or broadcasted ARP reply packets, not very common.

For a detailed description of all the proposed tests that will allow measuring all the different timeouts defined see section "APPENDIX III: ARP timeouts research".

Conclusions

The same considerations followed in the "Packet Taxonomy Tests" to select a reduce set of systems in which run the tests influenced the "Timeout Tests".

This section summarizes the ARP behaviour every OS presents based on the timeout tests. These tests allow analyzing how the table is managed and what kind of traffic and situations generate ARP events that affect the table entries.

The Solaris 8 timeouts were changed to the following values before running the tests:

```
# ndd –set /dev/arp arp_cleanup_interval 60000 (1 min.)
# ndd –set /dev/ip ip_ire_arp_interval 300000 (5 min.)
```

All the other OS were tested using the default values.

Cisco IOS 12.0
Cisco IOS has a default timeout of 4 hours per ARP entry, a huge value compared with other OS, whose values are between 1-20 minutes.

Cisco ARP algorithms are not influenced by the type of ARP packet, both, requests and replies are used to learn new entries and they use the same timer. Once and entry has been learned, generic IP traffic doesn't reset the timeout.

Cisco IOS is very dependent of external ARP traffic, and both directed and broadcasted packets, combining requests and replies, make the default 4 hours counter to start again.

HP-UX 10.20
HP-UX 10.20 uses a Unicast Polling method different for ARP request and replies.
When an ARP request is received a new entry is created and a 5 minute timeout is initialized. When the timer expires, it validates the entry using a directed ARP request. If a response is received, a 10 minute timeout is activated this time and when it finishes a new validation packet is sent. After 5 additional minutes, the entry expires. During all this period, 20 minutes, the entry is valid.

ARP request; 5 minutes; ARP validation; 10 minutes; ARP validation; 5 minutes; Entry expires

In case the entry has been learned through an ARP reply, the first period involved is the 10 minutes interval, therefore, the first 5 minutes period is not used.

ARP reply; 10 minutes; ARP validation; 5 minutes; Entry expires

So, both ARP packet types influence the ARP algorithms while generic IP traffic don't.

Linux kernel 2.4
Linux learn ARP entries and set them inside the table in the "reachable" state. After 30 seconds, see "ARP parameters by operating system" section, defined by the "base_reachable_time" parameter, the entry changes to the "stale" state.
When an entry is in this state and traffic is received from the associated host, Linux box replies using the entry but after 5 seconds (from the moment the traffic was received), defined by "delay_first_probe_time", it sends a unicast poll packet to confirm the entry validity.
Using the "arp" command it is not possible to visualize all these different states, but through the "ip neigh" command a detailed view can be obtained [MART1].
An entry is kept in the "stale" state during long periods of times, more than 50 minutes.

The packet type used to populate the table doesn't affect the timeouts.
The reception of generic IP traffic and ARP packets influence the timeout increasing the time the entry is kept in the "reachable" state.

Windows 2000 SP3
Windows timeouts are mainly based on entry reusing. When a new entry is created a default 2 minutes timer applies, "ArpCacheLife" parameter. If the entry is reused, the timer is increased, and the 2 minutes are applied since the moment the entry was reused.
This algorithm has a maximum of 10 minutes by default ("ArpCacheMinReferencedLife" parameter), after that time, the entry timeout is not reset and the entry must be renewed. It doesn't matter if the entry was learned through a request or a reply.
The reception of generic IP traffic or ARP packets doesn't change the timeout behaviour.
Invalid entries, when a reply has not been received, are kept for about 5 seconds.

Solaris 8
The expected Solaris behaviour, where an already learned entry cannot be overwritten for a specific period of time, works once the OS has used the entry.
Although the timers used were 1 and 5 minutes, as explained before, the ARP entries expired between 8.5 and 9.5 minutes depending on the test. Therefore it seems the internal ARP algorithm is based on some kind of random number generation that varies the timer delay.

It was observed that an entry is learned through ARP requests and replies but if it is not used in a period between 20 and 30 seconds, the entry is removed. During this period the entry can be relearned overwriting the previous MAC information.

If an entry is used to generate outbound traffic, then a blocking timeout gets involved, and the information cannot be overwritten during 9 minutes approximately. Solaris is the only OS analyzed that presents this blocking behaviour.

Summary

The following table summarizes which devices implement the "Unicast Polling" method described in the RFCs:

OS	Unicast polling
Cisco IOS router 12.0	No
HP-UX 10.20	Yes
Linux: kernel 2.4	Yes
Windows 2000 SP3	No
Solaris 8	No

OS fingerprinting based on ARP packets

Analyzing what types of ARP packets are generated by every operating system would allow to create an OS fingerprinting classification tree that will permit to find out what OS are participating in the network just by listening to their ARP traffic.

ARP standard request analysis

The most important field, not established at the RFC level, is the "Target MAC address".

The analysis performed has revealed that some OS set the "MAC Target Address" field in an ARP request to all ones, FF:FF:FF:FF:FF:FF, like Solaris, while others set it to all zeros, 00:00:00:00:00:00, such as Windows 2000, Linux kernel 2.4, Cisco IOS and HP-UX 10.20.

Gratuitous ARP packets

The initial gratuitous ARP packets also have a significantly different look and feel between different operating systems (see "Bootstrap and shutdown times research" section).

Windows OS set "Target MAC address" field to all zeros while Unixes variants set it to all ones.

Ethernet trailers and minimum packet size enforcement

Another element that can help to fingerprint a host, determining its OS, is the Ethernet trailer information appended to the Ethernet frames. This additional data is added to reach the standard Ethernet boundaries. Each operating system implements this fill up process in a different way, although RFC 1042, specifies that "the data field should be padded (with octets of zero)".

ARP Ethernet packets only use 42 bytes of data, although the smallest legal Ethernet packet is 60 bytes, not including CRC (4 bytes) [ETHS1]. The OS pads an extra 18 bytes on to the end of ARP packets to meet this minimum length requirement. You don't have to worry about adding padding when generating ARP packets it will be added by the OS. Some systems may not enforce the minimum packet size, while others will.

During year 2003 a new vulnerability was found, Etherleak [CERT2] [OFIR2], affecting lots of network interface device drivers. Vulnerable NICS incorrectly handle frame padding, allowing an attacker to view slices of previously transmitted packets or portions of kernel memory. Therefore the trailers visualize must be influenced by memory contents not being a fixed value.

Conclusions

When sending gratuitous ARP bootstrapping packets it is very rare to see memory data copied in the Ethernet frame because these are almost always the

first traffic generated by a host (see "Bootstrap and shutdown times research" section).

Solaris 8 always set the Ethernet frames trailers using the number 5 value; this is very helpful to guess the OS of a remote Solaris host. Other OS, such as Windows 2000 and HP-UX 10.20, suffer the described memory leak problem and garbage data can be seen inside the trailer portion. All these three OS use 60 bytes Ethernet frames, enforcing the minimum packet size.

Cisco follows the RFC recommendation and the trailer section is always set to zeros. Apart from that, its packet size is 60 bytes although some IOS versions generate 64 bytes frames.

Linux kernel 2.4 enforces the minimum packet size, using 60 bytes, and uses zero values for padding.

OS	Minimum size	Trailers
Cisco IOS	60 and 64	Zero's value
HP-UX 10.20	60	Memory data
Linux kernel 2.4	60	Zero's value
Windows 2000 SP3	60	Memory data
Solaris 8	60	Five's value

Figure 2.19. Minimum packet size and trailer values

Bootstrap and shutdown times research

Most hosts on a network will send out a gratuitous ARP packet (see "Gratuitous ARP packet" section) when they are initializing their IP stack, bootstrap time [STEV1]. This gratuitous ARP packet is typically an ARP request for their own IP address, which is used to check for a duplicate IP address. If there is a duplicate address, some stacks do not complete initialization, while others simply generate a warning message (see ""Duplicate IP address" message" section).

To analyze the bootstrapping and shutdowning processes the same lab environment as the one for timeout research (see "APPENDIX II: Research lab description" section) was used. This lab was also implemented to analyze the network interfaces manipulation (next section).

The process followed to test the ARP packets generated during the boot and shutdown time was:

1. Activate network traces, called "boot-traces" because they are related to the boot process, in the analyzer host to capture all traffic (promiscuous mode).
2. Wait for 2-5 minutes to check the existent traffic generated by the unique system already active in the network, the analyzer host. Traffic should be almost inexistent.
3. Boot the target system, the one to be analyzed: "**Boot Test**".

4. Once started, leave it up without any kind of user or external activity during 5 minutes.
   Do not log into the system, as this could launch lots of different communications (OS dependant), that could influence the obtained results.
5. Stop network traces: "boot-traces".
6. Connect to target system console directly, not through a network connection, in order to switch it off gracefully.
7. Start network traces, "shutdown-traces".
8. Shutdown target system, documenting the process followed to do so: "**Shutdown Test**".
9. Stop network traces: "shutdown-traces".

All the systems analyzed have a statically configured IP address (and the default router and DNS servers defined); DHCP is not used at all. An isolated environment where no default router neither DNS server were reachable was used, so it is common to see non responded ARP request packets for that systems.

A future research could incorporate more complex elements:

- A real environment in which both, the default router and the DNS servers were active could be tested: this probably won't provide additional useful information but will introduce lot of network noise.
- Instead of having a fixed IP address, we should test dynamically configured network interfaces (DHCP clients). See "DHCP systems" section. It will be interesting to investigate the situation where the DHCP server can be reached, and the opposite one.

In the initial tests design a power disruption was also simulated, by switching off the power button; as none of the systems generated any ARP traffic, those tests have been omitted. This test was considered initially because some hardware devices could run some actions, managed at the firmware level, without having an active operating system. A final comment about these tests is that based on the hardware platform, pressing the power button could be similar to a graceful shutdown, as for example in an HP Model 712/100 workstation or a Sun Ultra 5 workstation, so the simulation should be done by unplugging the power cable.

Conclusions

The conclusions obtained from the bootstrap and shutdown times have being summarized in the following table. Only the ARP traffic is shown.

OS	Boot Test	Shutdown Test
Cisco IOS router	No	No
Cisco IOS switch	No	No
HP-UX 10.20	No ARP gratuitous packets [3]	No
HP-UX 11.00	1 gratuitous ARP request. All ones in "Target MAC Addr." [2]	No

HP-UX 11i (11.11)	1 gratuitous ARP request. All ones in "Target MAC Addr." [2]	No
Linux: kernel 2.4	3 different gratuitous ARP packets: - "strange" ARP request with "Sender IP Address" all zeros instead of host IP. - 1 second later (delay) it responds to itself with a gratuitous ARP reply packet. - 2 seconds later, it sends a gratuitous ARP request. All ones in "Target MAC Addr." Finally it sends ARP requests for the default router.	No
Windows 2000 without SP	3 gratuitous ARP requests. All zeros in "Target MAC Addr." 1 second delay between packets.	No
Windows 2000 SP3	3 gratuitous ARP requests. 1 second delay between packets.	No Several ARP req. for default router.
Windows NT4.0 SP6	3 gratuitous ARP requests. 1 second delay between packets.	No. Several ARP req. for default router.
Solaris 8	4 gratuitous ARP packets. [1] 6, 4 and 2 seconds delay between them.	No

The ARP requests asking for the default router are software and configuration dependant, so they must not be considered as a common OS behaviour under any circumstances. For example, a system that has a NTP time server configured, will try to contact it at boot time. If the NTP server is placed in a remote network, it will resolve the default router MAC address through ARP request packets.

We must take into account that observed delays can be influenced by upper layer protocols, for example, the TCP retransmission algorithm:

[1] Solaris box asks for its default router using 30 ARP request packets divided in 6 sets of 5 packets each. Every packet belonging to the same set is sent every second, and there is a delay of 5 seconds between sets. Whenever it needs to resolve a MAC address, it uses a similar pattern, a set of 6 packets with one-second delay between them. The number of sets could vary.

[2] HP–UX 11.00 and 11i were asking for the default gateway by means of ARP request packets with the following time layout:
Both use sets consisting of 6 packets with a delay of one second between them. Three sets were sent with a delay of 10, 15 and 35 seconds respectively.

[3] HP-UX 10.20 sent two initial ARP request for the default router with a delay of two seconds. Then a similar packet is sent every 62 seconds.

HP-UX 11.00, 11i, Linux and Solaris 8 gratuitous ARP request packet:

ARP request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr.		
Host IP address		
FF:FF:FF:FF:FF:FF		
Host IP address		

Linux kernel 2.4 "strange" ARP request and gratuitous ARP reply:

ARP "strange" request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr.		
0.0.0.0		
FF:FF:FF:FF:FF:FF		
Host IP addr.		

ARP reply		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	2 (reply)
Sender MAC addr.		
Host IP addr.		
Sender MAC addr.		
Host IP addr.		

Windows 2000 (no SP or SP3) or Windows NT 4 SP6 gratuitous ARP request:

ARP request		
FF:FF:FF:FF:FF:FF		
Sender MAC addr.		
0x0806	0x0001	0x0800
6	4	1 (req.)
Sender MAC addr.		
Host IP addr.		
00:00:00:00:00:00		
Host IP addr.		

The default behaviour about the number of gratuitous packets sent can be altered modifying some OS parameters at the kernel level. For example, Microsoft KB Article 219374 [MS-KB1] defines how to do it for Windows systems. Other OS have similar parameters (see "ARP parameters by operating system" section).

The Windows NT 4.0 SP3 or higher gratuitous ARP behaviour is documented for static or DHCP addresses [MS-KB2]; despite the documentation says that delays are 5 and 1 second, the tests showed 1 second delay for all frames.

Shutdown Test

These were the procedures followed to shutdown every operating system.

OS	Shutdown procedure
Cisco IOS router: Cisco IOS switch:	Cisco#reload
Windows 2000	Go to "Start" – "Shut Down…", select "Shut down" option and press "OK"
Windows NT 4.0	Go to "Start" – "Shut Down…", select "Shut down the computer?" option and press "Yes"
Unix (all variants)	# shutdown –hy 0 ("Solaris doesn't accept "-h" option")

Activating/Deactivating network interfaces

Two methods to manipulate the status of a network interface have been considered: changing it manually or administratively and disconnecting the network cable to set the link down.

"**Interfaces Test 1**", changing the interface status gracefully.

Using the same lab environment as before, the following process was used:

1. Stop any user or external activity over the target system.
2. Start taking network traces.
3. From the target system console, shutdown the network interface, documenting the process followed to do so.
4. Leave the interface down for about 1 minute in order to differentiate activation and deactivation traces.
5. Bring the interface up again, documenting the process followed to do so.
6. Analyze network traces for about one minute.
7. Stop network traces.

"**Interfaces Test 2**", disconnecting the network cable.

The same test was developed by disconnecting the Ethernet cable from the target system and connecting it again.

OS	Interfaces Test 1		Interfaces Test 2	
	down	**up**	**down**	**up**
Cisco IOS router	No	No CDP and Loop traffic	No	No
Cisco IOS switch	No	No STP traffic	No	No STP traffic
HP-UX 10.20	No	No	No	No
HP-UX 11.00	No	1 gratuitous ARP req. "Target MAC Addr." all ones	No	1 gratuitous ARP req. "Target MAC Addr." all ones
HP-UX 11i	No	1 gratuitous ARP req. "Target MAC Addr." all ones	No	No
Linux: kernel 2.4	No	No	No	No
Windows 2000 without SP	No	3 gratuitous ARP req. 1 second delay between them.	No	3 gratuitous ARP req. 1 second delay between them.
Windows 2000 SP 3	No	2 gratuitous ARP req. 1 second delay between them.	No	2 gratuitous ARP req. 1 second delay between them.
Windows NT4.0 SP6	*NP*	*NP, Not possible without rebooting*	No	No
Solaris 8	No	3 gratuitous ARP req. "Target MAC Addr." all ones. 2 seconds delay between them.	No	No

All gratuitous packets generated by the analyzed OS keep the same format as the packet analyzed during the bootstrapping process.

Changing interfaces status

These were the procedures followed to activate/deactivate network interfaces:

OS	Interfaces administration procedures
Cisco IOS router: Cisco IOS switch:	Cisco>en Password: Cisco# Cisco#conf t Enter configuration commands, one per line. End with CNTL/Z. Cisco(config)#int eth 0/0 or int vlan 1 Cisco(config-if)#shutdown or Cisco(config-if)#no shutdown [1]
Windows 2000	"Start" – "Settings" – "Control Panel" – "Network and Dial-up Connections", select the network interface, press right mouse button and select "Disable" option.
Windows NT 4.0	"Start" – "Settings" – "Control Panel" – "Network": it is not possible (*NP*) to simply disable the network interfaces because when changing network properties or protocol to network card bindings you are required to restart the system.
Unix (all variants)	# ifconfig <NIC-X> down # ifconfig <NIC-X> up

[1] When a new port is used in a switch, it executes, unless otherwise configured, the STP protocol state phases before activating the port, to avoid loops and to resolve the root bridge.

ARP parameters by operating system

The operating system's configuration parameters allow setting up the timeout variables for the ARP algorithms and for other elements of the ARP module. All the ARP parameters related to the main operating systems analyzed by this paper will be described. Additional OS should be included in a future review.

Cisco IOS

The Cisco IOS allows changing the unique timeout associated to the expiration of ARP table entries. By default, an IOS router or switch maintains an ARP entry for four hours, 14400 seconds. This is an extremely high value compared with the timeouts used by the Unix and Windows OS.

Parameter:	arp timeout
Description:	It configures how long an entry remains in the ARP cache. Use the "arp timeout" command in interface configuration mode. To restore the default value, use the no form of this command. A value of zero means that entries are never cleared from the cache.
Actions:	GET: Router#show interfaces

	Ethernet0/0 is up, line protocol is up Hardware is AmdP2, address is 0003.f083.1b00 (bia 0003.f083.1b00) Internet address is 192.168.1.1/24 MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, reliablility 255/255, txload 1/255, rxload 1/255 Encapsulation ARPA, loopback not set, keepalive set (10 sec) ARP type: ARPA, **ARP Timeout 04:00:00** … SET: (seconds) Router#conf t Router(config)#interface ethernet 0 Router(config-if)#arp timeout 3600

Min.	Default	Max.
0 (never)	14400 (4 hours)	4294967 (~ 1190 h.)

HP-UX 10.20

HP-UX 10.20 allows the administration of the TCP/IP stack implementation through the usage of the "nettune" command.

Parameter:	arp_killcomplete
Description: # nettune –h	The number of seconds that an arp entry can be in the completed state between references. When a completed ARP entry is unreferenced for this period of time, it is removed from the ARP cache.
Actions:	GET: # nettune -l arp_killcomplete SET: (seconds) # nettune -s arp_killcomplete 2400

Min.	Default	Max.
60 (1 min.)	1200 (20 minutes)	3600 (1 h.)

Parameter:	arp_killincomplete
Description: # nettune -h	The number of seconds that ARP entries can be in the incomplete state. After this period of time, incomplete ARP entries will be removed from the ARP cache.
Actions:	GET: # nettune -l arp_killincomplete SET: (seconds) # nettune -s arp_killincomplete 300

Min.	Default	Max.
30 (0.5 min.)	600 (10 minutes)	3600 (1 h.)

Parameter:	arp_unicast
Description: # nettune -h	The interval in seconds used for sending unicast ARP packets to remote hosts for which there is an ARP entry in the completed state.
Actions:	GET: # nettune -l arp_unicast SET: (seconds) # nettune -s arp_unicast 150

Min.	Default	Max.
60 (1 min.)	300 (5 minutes)	3600 (1 h.)

Parameter:	arp_rebroadcast	
Description: # nettune -h	The number of seconds between broadcast inquiries for ARP entries in the incomplete or hold states.	
Actions:	GET: # nettune -l arp_rebroadcast SET: (seconds) # nettune -s arp_rebroadcast 120	
Min.	**Default**	**Max.**
30 (0.5 min.)	60 (1 minute)	3600 (1 h.)

HP-UX 11.00

HP-UX 11.00 and 11i allow the administration of the TCP/IP stack implementation through the usage of the "ndd" command.

Parameter:	arp_cache_report
Type:	Supported
Description: # ndd -h	Displays a report showing the ARP cache. It shows the static ARP entries meanwhile the "arp" command doesn't.
Actions:	GET: # ndd -get /dev/arp arp_cache_report

Output example:

```
ifname   proto addr      proto mask      hardware addr     flags
lan0     192.168.1.061   255.255.255.255 00:03:47:7e:7e:7e
lan0     192.168.1.057   255.255.255.255 00:90:27:8e:8e:8e
lan0     192.168.1.001   255.255.255.255 08:00:20:99:88:77 PERM PUBLISH LOCAL
lan0     192.168.1.061   255.255.255.255 00:03:47:9e:9e:9e PERM
lan0     224.000.000.000 240.000.000.000 01:00:5e:00:00:00 PERM MAPPING
```

Parameter:	arp_ cleanup_interval	
Type:	Supported	
Description: # ndd -h	Controls how long ARP entries stay in the ARP cache, accurately, the amount of time that non-permanent, resolved entries are permitted to remain in ARP's cache.	
Actions:	GET: # ndd -get /dev/arp arp_cleanup_interval SET: (milliseconds) # ndd -set /dev/arp arp_cleanup_interval 60000 (1 minute)	
Min.	**Default**	**Max.**
30000 (0.5 min.)	300000 (5 minute)	3600000 (1 h.)

Parameter:	arp_debug	
Description:	Controls the level of ARP module debugging.	
Actions:	GET: # ndd /dev/arp arp_debug SET: (boolean) # ndd –set /dev/ arp arp_debug 1	
Min.	**Default**	**Max.**
0	0	1

There are two additional ARP parameters not so relevant:

Parameter:	arp_dl_sap
Description:	Set the SAP when ARP binds to a DLPI device.

Parameter:	arp_dl_snap_sap
Description:	The SAP to use for SNAP encapsulation.

HP-UX 11i

The following options are exactly the same in HP-UX 11.00 and 11i: arp_cache_report, arp_cleanup_interval, arp_debug, arp_dl_sap, arp_dl_snap_sap. Some new options have been added in the new HP-UX version.

Parameter:	arp_defend_interval	
Type:	Supported	
Description: # ndd -h	Seconds to wait before initially defending a published entry. A value of zero prevents published entries from being defended.	
Actions:	GET: # ndd -get /dev/arp arp_defend_interval SET: (seconds) # ndd -set /dev/arp arp_defend_interval 10	
Min.	**Default**	**Max.**
0	0	999999999

Parameter:	arp_redefend_interval	
Type:	Supported	
Description: # ndd –h	Seconds to wait before redefending a published entry. This value is only meaningful if arp_defend_interval has been set to a non-zero value. A value of zero prevents a published entry from being redefended.	
Actions:	GET: # ndd -get /dev/arp arp_redefend_interval SET: (seconds) # ndd -set /dev/arp arp_redefend_interval 10000	
Min.	**Default**	**Max.**
0	5000	999999999

Parameter:	arp_resend_interval	
Type:	Supported	
Description: # ndd –h	Number of milliseconds between ARP request retransmissions.	
Actions:	GET: # ndd -get /dev/arp arp_resend_interval SET: (milliseconds) # ndd -set /dev/arp arp_resend_interval 5000	
Min.	**Default**	**Max.**
0	2000 (2 seconds)	60000

Value ranges for unsupported parameters are not documented. These are related to the Proxy ARP functionality.

Parameter:	arp_announce_count
Type:	Unsupported
Description: # ndd –h	Number of transmits used to announce a published entry.

Actions:	GET: # ndd -get /dev/arp arp_announce_count SET: (number) # ndd -set /dev/arp arp_announce_count 3	
Min.	**Default**	**Max.**
unknown	1	unknown

Parameter:	arp_probe_count	
Type:	Unsupported	
Description: # ndd –h	Number of address resolution requests to make before adding a published entry.	
Actions:	GET: # ndd -get /dev/arp arp_probe_count SET: (number) # ndd -set /dev/arp arp_probe_count 3	
Min.	**Default**	**Max.**
Unknown	0	unknown

Linux: kernel 2.4

The Linux ARP module supports a "sysctl" interface to configure parameters on either a global or on a per-interface basis. The sysctls can be accessed by reading or writing the "/proc/sys/net/ipv4/neigh/*/*" files or with the sysctl(2) interface. Information extracted from Linux "arp (7)" man page [ARPMAN1]. In Linux terminology the ARP table is known as the neighbour table.

Each interface in the system has its directory in /proc/sys/net/ipv4/neigh/. The settings in the "default" directory are used for all newly created devices.

Unless otherwise specified, time-related sysctls are specified in seconds. Some timer settings are specified in jiffies, which is architecture related. On the Alpha CPUs a jiffy is 1/1024 of a second, on in most of the other architectures (Intel) it is 1/100s.

Due to the open source nature of the Linux OS, the ARP internal algorithms are really well documented [MART1].

Linux ARP parameter	Description	Default value
app_solicit	The maximum number of probes to send to the user space ARP daemon via netlink before dropping back to multicast probes (see mcast_solicit).	0
mcast_solicit	The maximum number of attempts to resolve an address by multicast/broadcast before marking the entry as unreachable.	3
ucast_solicit	The maximum number of attempts to send unicast probes before asking the ARP daemon (see app_solicit).	3
base_reachable_time	Once a neighbour has been found, the entry is considered to be valid for at least a random value between: base_reachable_time/2 & 3*base_reachable_time/2.	30 seconds

	An entry's validity will be extended if it receives positive feedback from higher-level protocols.	
delay_first_probe_time	Delay before first probe after it has been decided that a neighbour is stale.	5 seconds
gc_interval	How frequently the garbage collector for neighbour entries should attempt to run. This parameter defines the Linux ARP table aging timeout.	30 seconds
gc_stale_time	Determines how often to check for stale neighbour entries. When a neighbour entry is considered stale it is resolved again (a new ARP request is generated after "delay_first_probe_time" seconds) before sending data to it.	60 seconds
locktime	The minimum number of jiffies to keep an ARP entry in the cache. This prevents ARP cache thrashing if there is more than one potential mapping (generally due to network misconfiguration).	1 second
proxy_delay	When an ARP request for a known proxy-ARP address is received, delay up to proxy_delay jiffies before replying. This is used to prevent network flooding in some cases.	0.8 seconds
proxy_qlen	The maximum number of packets which may be queued to proxy-ARP addresses.	64
retrans_time	The number of jiffies to delay before retransmitting a request.	1 second
unres_qlen	The maximum number of packets which may be queued for each unresolved address by other network layers.	3

As an example, these are the values extracted form a Linux system, kernel 2.4:

ARP parameter for "eth0": /proc/sys/net/ipv4/neigh	Default value
eth0/anycast_delay (1)	100
eth0/app_solicit	0
eth0/base_reachable_time	30
eth0/delay_first_probe_time	5
eth0/gc_stale_time	60
eth0/locktime	100
eth0/mcast_solicit	3
eth0/proxy_delay	80
eth0/proxy_qlen	64
eth0/retrans_time	100
eth0/ucast_solicit	3
eth0/unres_qlen	3

(1) The "anycast_delay" is the only ARP parameter available in the kernel and not documented in the "arp (7)" man page. It stands for the maximum value for random delay of answers to neighbor solicitation messages in jiffies (1/100 sec). Not yet implemented (Linux does not have anycast support yet) [OBS1].

The parameter values must be obtained by reading the file, for example, with the "cat" command, and changed using the "echo" command:

```
# cat /proc/sys/net/ipv4/neigh/eth0/app_solicit"
# echo 50 > /proc/sys/net/ipv4/neigh/eth0/retrans_time"
```

Linux also has another ARP configuration parameter related to the possibility of setting ARP filters, "/proc/sys/net/ipv4/conf/*/arp_filter", where "*" may be a network interface, as "eth0", "lo"… and "default" or "all". See "APPENDIX VII: ARP flux" section for a detailed explanation.

Finally, the kernel keeps the ARP table in the "/proc" pseudo file system, similar to the "arp_cache_report" parameter in HP-UX 11.x or Solaris 8:

```
# cat /proc/net/arp
IP address       HW type     Flags       HW address            Mask     Device
10.181.139.87    0x1         0x2         08:00:09:EE:E2:51     *        eth0
10.181.139.76    0x1         0x2         08:00:09:1E:E4:DA     *        eth0
10.181.139.9     0x1         0x2         08:00:09:0E:EC:03     *        eth0
```

Linux provides a set of tools, "iproute2" [ADVR1] [IPROU1], that allows the extraction of very detailed networking information. It was created for kernel 2.2 after the introduction of a completely redesigned kernel network subsystem. More data is available with the "ip neigh" command than with the "arp" command.

Linux documentation states its "RFC 1122: Timeouts and Unicast Poll" implementation:

> When there is no positive feedback for an existing mapping after some time a neighbour cache entry is considered stale. Positive feedback can be gotten from a higher layer; for example from a successful TCP ACK. Other protocols can signal forward progress using the MSG_CONFIRM flag to sendmsg(2). When there is no forward progress ARP tries to reprobe. It first tries to ask a local arp daemon app_solicit times for an updated MAC address. If that fails and an old MAC address is known a unicast probe is send ucast_solicit times. If that fails too it will broadcast a new ARP request to the network. Requests are only send when there is data queued for sending.

Windows 2000

The most relevant information has been extracted from the documentation about the internals of the Windows 2000 TCP/IP stack implementation [W2000].

To summarize, the default Windows 2000 algorithm states that if an entry is not used by any outgoing datagram for two minutes, the "ArpCacheLife" parameter value, the entry is removed from the ARP cache. Entries that are being referenced are given additional time, in two minutes increments, up to ten minutes. After ten minutes they are removed from the ARP cache, "ArpCacheMinReferencedLife" parameter. Entries manually added are not removed from the cache automatically.

The following ARP parameters are configurable through the Registry Editor (Regedt32.exe or regedit.exe). None of the ARP parameters are visible in the

registry by default, so they must be created to modify the default behaviour of the TCP/IP protocol driver.

All "keys" refer to the following registry branch: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters.

There are some ARP parameters associated to Token Ring networks, as "ArpTRSingleRoute" or "ArpAlwaysSourceRoute", and ATM networks, listed under "AtmArpC" keys, that won't be described and analyzed in this paper, because it is focused on Ethernet networks.

Parameter:	ArpCacheLife	
Description:	See "ArpCacheMinReferencedLife". In absence of an ArpCacheLife parameter, the defaults for ARP cache timeouts are a two-minute timeout on unused entries and a ten-minute timeout on used entries. This new registry parameter was added in Windows NT 3.51 Service Pack 4 to allow more administrative control over aging.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Number of seconds	
Min.	**Default**	**Max.**
0	120 or 600	0xFFFFFFF

Parameter:	ArpCacheMinReferencedLife	
Description:	ArpCacheMinReferencedLife controls the minimum time until a referenced ARP cache entry expires. This parameter can be used in combination with the ArpCacheLife parameter, as follows: - If ArpCacheLife is greater than or equal to ArpCacheMinReferencedLife, referenced and unreferenced ARP cache entries expire in ArpCacheLife seconds. - If ArpCacheLife is less than ArpCacheMinReferencedLife, unreferenced entries expire in ArpCacheLife seconds, and referenced entries expire in ArpCacheMinReferencedLife seconds. Entries in the ARP cache are referenced each time that an outbound packet is sent to the IP address in the entry.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Number of seconds	
Min.	**Default**	**Max.**
0	600 (10 minutes)	0xFFFFFFF

Parameter:	ArpRetryCount	
Description:	This parameter controls the number of times that the computer sends a gratuitous ARP for its IP address(es) while initializing. Gratuitous ARPs are sent to ensure that the IP address is not already in use elsewhere on the network. The value controls the actual number of ARPs sent, not the number of retries.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Number	
Min.	**Default**	**Max.**
1 (*)	3	3

(*) There was a bug fixed by Windows 2000 SP3 preventing the gratuitous ARP packets to be disabled at bootstrap time, because value "0" was not admitted [MS-KB5].

Parameter:	EnableBcastArpReply	
Description:	This parameter controls whether the computer responds to an ARP request when the source Ethernet address in the ARP packet is not unicast. Network Load Balancing Service (NLBS) will not work properly if this value is set to 0.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Boolean	
Min.	**Default**	**Max.**
0 (false)	1	1 (true)

And finally a not so relevant ARP parameter:

Parameter:	ArpUseEtherSNAP	
Description:	Setting this parameter to 1 forces TCP/IP to transmit Ethernet packets using 802.3 SNAP encoding. By default, the stack transmits packets in DIX Ethernet format. It always receives both formats.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Boolean	
Min.	**Default**	**Max.**
0 (false)	0	1 (true)

Windows NT 4.0

The most relevant information has been obtained from the documentation about the internals of the Windows NT TCP/IP stack implementation [WINNT1].

Again, the default behaviour is similar to the one described for Windows 2000: entries are aged out of the ARP cache if they are not used by any outgoing datagram for two minutes. Entries that are being referenced get aged out of the ARP cache after 10 minutes. Entries manually added are not aged out of the cache.

A new registry parameter, "ArpCacheLife", was added in 3.51 Service Pack 4 to allow more administrative control over aging.

These parameters normally do not exist in the registry. They may be created to modify the default behaviour of the TCP/IP protocol driver.

Parameter:	ArpCacheLife (in Windows NT 3.51 Service Pack 4)	
Description:	It controls the ARP cache life for entries NOT being referenced. Referenced entries (through outbound traffic) time out after 10 minutes, and this timeout value is not adjustable.	
Key:	Tcpip\Parameters	
Value type:	REG_DWORD—Number of seconds	
Min.	**Default**	**Max.**
0	120 or 600	0xFFFFFFF

A bug in Windows NT 4.0 [MS-KB6] describes this behaviour: OS doesn't allow referenced ARP entries to timeout until the 10 minutes fixed timer is completed. At least you need Windows NT 4.0 SP3 to apply the proposed hotfix. The hotfix introduces a new registry key and changes previous key meaning:

Parameter:	ArpCacheMinReferencedLife
Description:	ArpCacheMinReferencedLife controls the minimum time before a referenced ARP entry expires. This parameter can be used in combination with the ArpCacheLife parameter, as follows: - If ArpCacheLife >= ArpCacheMinReferencedLife, referenced and unreferenced ARP cache entries expire in ArpCacheLife seconds. - If ArpCacheLife < ArpCacheMinReferencedLife, unreferenced entries expire in ArpCacheLife seconds, and referenced entries expire in ArpCacheMinReferencedLife seconds. Entries in the ARP cache are referenced each time that an outbound packet is sent to the IP address in the entry.
Key:	Tcpip\Parameters
Value type:	REG_DWORD—Number of seconds

Min.	**Default**	**Max.**
0	600 (10 minutes)	0xFFFFFFF

So, with this new feature, the "ArpCacheLife" key changes its meaning to the Windows 2000 behaviour: by default, in absence of an ArpCacheLife parameter, the default for ARP cache time-outs is a 2 minute time-out on unused entries and a 10 minute time-out on used entries. Adding this parameter and setting a value in seconds overrides the default values described above.

Windows NT also defines the irrelevant "ARPUseEtherSNAP" parameter.

Solaris 8

The main two ARP parameters in Solaris are shown, and both are valid at least since Solaris 2.5. Take into account that Solaris keeps two ARP tables in memory, the ARP cache table and the IP routing table, that receive feedback from the ARP module. Each parameter affects one of these tables.

Parameter:	ip_ire_flush_interval (Previously to Solaris 8) ip_ire_arp_interval (Solaris 8: NEW parameter name)
Description:	ARP information lifetime in IP routing table: This option determines the period of time during which a specific route will be kept, even if currently in use, so it unconditionally flushes ARP info from IP routing table ARP entries not in use are flushed every 20 minutes. NOTE: "ire" stands for "internal routing entries".
Actions:	GET: # ndd [–get] /dev/ip ip_ire_arp_interval SET: (milliseconds) # ndd –set /dev/ip ip_ire_arp_interval 60000 (1 minute)

Min.	**Default**	**Max.**
60000	1200000 (20 minutes)	999999999

If this parameter is set to a value lower than the minimum, the "ndd" command generates an error message: "invalid argument".

Parameter:	arp_cleanup_interval		
Description:	ARP table cache lifetime: This option determines the period of time during which the ARP cache table maintains entries, so it discards an ARP entry from ARP cache after this interval. This is the time during which the cache will hold on to unsolicited information in case IP needs it. An ARP entry that is not completed within 5 minutes is removed.		
Actions:	GET: # ndd [–get] /dev/arp arp_cleanup_interval SET: (milliseconds) # ndd –set /dev/arp arp_cleanup_interval 60000 (1 minute)		
Min.		**Default**	**Max.**
30000		300000 (5 minutes)	unknown

If this parameter is set to a value lower than the minimum it has no effect but the "ndd" command doesn't generate any error message.

Although several Solaris documents [SUNS1] refer to the reduction of both ARP timers as a defensive method, this can also be considered as a facility for the attacker:

"ARP attacks may be effective with the default interval. Shortening the timeout interval should reduce the effectiveness of such an attack." [NDD1]

Reasons for decreasing or increasing these timers:

- Reducing the time an ARP entry remains in the table allows a faster removal of forged or spoofed entries, but, given the fact that Solaris sets a period in which the ARP table is not updated, this period can also be considered as a secure state period: the attacker could not poison the ARP table until the next timeout expiration, so system is "safe" until then.
- When the timer expires, a race condition takes place, and the first ARP packet that arrives will be kept in the table for another period of "N" seconds.
- If the timer is lower, the attacker could gain the system control sooner, and if timer is bigger, the attacker will spend more time until it gets the unauthorized control.
- A small timeout forces the attacker to continuously update the target ARP table to keep the fake MAC address, so a greater effort must be done to deploy a successful attack.
- Reducing these timers will also slow down an attacker, but won't stop him. On the other side, it will affect network performance by increasing the ARP network traffic between systems.

These are other important Solaris ARP related parameters:

Parameter:	arp_cache_report
Description:	This parameter shows the information contained in the ARP cache table.
Actions:	# ndd /dev/arp arp_cache_report

Output example: (read only parameter)

```
ifname    proto addr       proto mask       hardware addr     flags
hme0      192.168.1.061    255.255.255.255 00:03:47:7e:7e:7e
hme0      192.168.1.057    255.255.255.255 00:90:27:8e:8e:8e
hme0      192.168.1.001    255.255.255.255 08:00:20:99:88:77 PERM PUBLISH MYADDR
hme0      192.168.1.061    255.255.255.255 00:03:47:9e:9e:9e
hme0      224.000.000.000 240.000.000.000 01:00:5e:00:00:00 PERM MAPPING
```

Parameter:	arp_debug		
Description:	This parameter switches on ARP debugging information.		
Actions:	GET: # ndd /dev/arp arp_debug SET: (boolean) # ndd –set /dev/ arp arp_debug 1		
Min.		**Default**	**Max.**
0		0	1

These parameters are related to the Proxy ARP functionality.

Parameter:	arp_publish_interval		
Description:	A variable that defines how long the system waits between broadcasts of an Ethernet address that it is configured to be published.		
Actions:	GET: # ndd /dev/arp arp_publish_interval SET: (milliseconds) # ndd –set /dev/ arp arp_ publish_interval 4000		
Min.		**Default**	**Max.**
unknown		2000	unknown

Parameter:	arp_publish_count		
Description:	A variable that defines how many ARP broadcasts are sent in response to a query for an address that this system publishes.		
Actions:	GET: # ndd /dev/arp arp_publish_count SET: (number of times) # ndd –set /dev/ arp arp_ publish_count 5		
Min.		**Default**	**Max.**
unknown		3 (number of times)	unknown

Default size of the ARP cache

A not easy to answer question is what the default size of the ARP cache is in the different OS.

Windows NT 3.5x/4.0 and Windows 2000 adjust the size of the ARP cache automatically to meet the needs of the system [W2000] [WINNT1].

The Powertweak project [PWER1], a parametrization tool for Linux, documents a Linux 2.2 example. The values in kernels 2.2 and 2.4 are the same. Linux allows ARP table size parametrization through the following parameters:

Linux parameter	Description	Default value
gc_thresh1	The minimum number of entries to keep in the ARP cache. The garbage collector will not run if there are fewer than these number of entries in the cache.	128
gc_thresh2	The soft maximum number of entries to keep in the ARP cache. The garbage collector will allow the number of entries to exceed this for 5 seconds before collection will be performed.	512
gc_thresh3	The hard maximum number of entries to keep in the ARP cache. The garbage collector will always run if there is more than this number of entries in the cache.	1024

Other operating systems document that the ARP cache is administered as a linked list of dynamically allocated kernel memory buffers (HP-UX and Cisco). The only resource that might limit the size of the ARP cache is the size of memory available for the kernel. The size of an ARP entry is about 64 bytes.

HA solutions

There are different high availability and failover solutions in the computer industry, and most of them are implemented using layer 2 communications where ARP is involved.

The same problematic situation described in the UNARP RFC could occur in failover solutions where the active node is dynamically substituted by a standby node. Based on the notification packets generated, some client hosts will not refresh its ARP table.
As a brief introduction, some of the most frequently used failover solutions will be pointed out, but in a future version of this paper a deeper research should be accomplished:

- Cisco HSRP, Hot Standby Routing Protocol (defined in the RFC 2281), provides next-hop redundancy. The cluster uses a virtual MAC address for the management traffic, HSRP, with the following format (last number is the failover group, example, "02"): 00-00-0c-07-ac-02. This virtual MAC address can be changed through the following command:

  ```
  interface eth 0/0
  standby 10 mac-address <<NEW_MAC>>
  ```

  HSRP packets have the previous MAC address as source, and its destination address is the multicast MAC address associated to all routers in the same subnet: 01:00:5e:00:00:02 (224.0.0.2).

  When a router becomes active, the virtual IP address is moved to a different MAC address. The newly active router sends a gratuitous ARP

reply. Not all host implementations handle this gratuitous packet correctly [HSRP1].

Cisco HSRP implements some security features, as the usage of authentication based on a shared password:

```
interface eth 0/0
standby 10 authentication SECRET
```

- VRRP, Virtual Router Redundancy Protocol, is a failover protocol commonly used in Nokia security appliances. It is defined in RFC 2338 and it supports IP header authentication based on the MD5 algorithm.

- Windows NLBS:
  NLBS, Network Load Balancing Service, allows network load balancing between multiple interfaces in a multihomed system.
  The Windows "EnableBcastArpReply" parameter controls whether the computer responds to an ARP request when the source Ethernet address in the ARP is not unicast. NLBS will not work properly if this value is set to false, that is, not respond to these type of ARP packet.

- Microsoft Cluster Service:
  It uses ARP gratuitous requests in a failover situation [MS-KB3].

- LinuxVirtualServer:
  The LVS [LVS2] project development had to deal with special ARP situations associated to multihomed systems [LVS1].

DHCP systems

An special case are systems that don't have an IP address statically configured, so they need to dynamically interrogate a DHCP, Dynamic Host Configuration Protocol, server in order to get a valid network IP address.

A brief introduction is presented covering the ARP packets used by the most popular client and router OS, Windows and Cisco IOS. In a future version new operating systems should be analyzed.

Windows 2000 DHCP clients: Automatic Client Configuration an media sense

The DHCP Windows clients [W2000] execute some tests, implemented through gratuitous ARP packets, to make sure that the IP address that they have chosen is not already in use. If it is, the system informs the DHCP server about the IP address conflict and, instead of invalidating the stack, as it happens when a static IP address is configured (see ""Duplicate IP address" message" section), they request a new address from the DHCP server (up to 10 addresses) and they ask it to flag the conflicting address as bad. This capability is commonly known as DHCP Decline support. Once the DHCP client has received an address that not in use, it configures the interface with this IP address.

Cisco DHCP

Cisco devices implement a "show" command [IOS123] that displays address conflicts found by a Cisco IOS DHCP server when addresses are offered to the clients.
The server uses ICMP to avoid conflicts, while the client uses gratuitous ARP packets to detect other clients. If an address conflict is detected, the address is removed from the server pool and it won't be reassigned until an administrator resolves the conflict.

```
Router# show ip dhcp conflict

IP address    Detection Method     Detection time
192.168.1.2   Ping                 Feb 17 1999 12:08 PM
192.168.1.6   Gratuitous ARP       Feb 26 1999 09:54 AM
```

The IOS has the "update arp" command that secures dynamic ARP entries in the ARP table to their corresponding DHCP bindings. If not used the ARP entries are considered dynamic. However, existing active leases are not secured. These leases will remain insecure until they are renewed. When the lease is renewed, it is treated as a new lease and will be secured automatically. This command can be configured only under the following conditions:

- DHCP network pools in which bindings are created automatically and destroyed upon lease termination or when the client sends a DHCPRELEASE message.
- Directly connected clients on LAN interfaces and wireless LAN interfaces.

Signature of the attack

The most famous ARP monitoring and detection tools (see "How to protect against it" section) as the operating system ARP module itself, "arpwatch" or "snort", look for variations in the address pairs associations between MAC and IP addresses.

The typical signature is based on having some kind of stateful conception. It is needed to register all the associations seen in the network traffic and, when a given IP address appears with a different MAC address, an alarm is triggered, notifying that an ARP address change has occurred.

The high availability and fault tolerant solutions are the main source of false positives for this type of monitoring systems, changing associations between active and standby nodes. When there is a failure in the active node, the standby device takes its role, and a new association is seen in the network: the same IP address, previously associated to the damaged node MAC address, is now related to the standby node MAC address. Some failover solutions establish that, once the corrupted node is back to a normal state, it should take the control of the service, so the initial association will be seen in the network again.

This is the reason why advanced tools, such as "arpwatch" [ARPW1], have introduced a differentiated warning message called "flip-flop": When this situation is detected, it will generate a mail message called "flip flop" and a syslog message with the contents "reused old Ethernet address". The definitions for both messages extracted from "arpwatch" man page are:

Syslog: "reused old ethernet address"

> "The ethernet address has changed from the most recently seen address to the third (or greater) least recently seen address. (This is similar to a flip flop.)"

Mail: "flip flop"

> "The ethernet address has changed from the most recently seen address to the second most recently seen address."

At the packet level it is difficult to find a unique signature due to the different ARP packet flavors generated by each OS. There are two conceptual elements to check:

- anomalies in the ARP packets.
- anomalies in the ARP traffic flow: a traffic monitor should be mainly focused on checking the supposed ARP stateful behaviour, associating requests with replies.

Due to the lack of deterministic signatures, apart from the ones used by "arpwatch", the application of statistical analysis to ARP security [ABAD1] is proposed to find suspicious traffic. For example, an alarm could be triggered when an abnormally high percentage of gratuitous ARP packets is detected, or when a high percentage of certain MAC addresses is observed.

Using real or fake MAC addresses: pros and cons

As stated in the "Name" and "Brief description" sections, the attacker could poison other systems ARP tables with its MAC address or with a different one. This additional MAC address can be also associated to another system placed in the same LAN or can be randomly created. Let's analyze how each MAC address selection can increase the ARP spoofing thread or the network protection.

- 1. Attacker's real MAC address:

The easiest method, from the attacker's side, is redirecting other systems traffic to the attacker's real MAC address. This allows a real and direct detection of what type of network card, based on the OUI, and potential system is owned by the attacker and is sending the forged ARP packets.

- 2. Different MAC address:

The attacker can use a different address from the one hardcoded at the network card he is launching the attack from.

The simplest way to achieve this is by means of some OS dependent command or configuration file.

In this way, the attacker will "loose" the manufacturer MAC address and will use the new configured address instead.

The attacker might also want to keep both addresses: there are tools that allow so, for example, a simple sniffer which will get all the network traffic arriving to the attacker's system and which has some advanced capabilities that allow to respond to the ARP packets addressed to the new MAC address. In a switched environment, the attacker's host must inform the switch that this new address belongs to it, so it would need to generate some traffic having this new MAC address as the source address in order to allow the switch to learn that this MAC is associated to the physical port it is plugged into.

The attacker will need to send "x" packets per minute to maintain the switch poisoned, or if the CAM entry is removed, the traffic will be flooded to all ports until the next packet is received and the entry is created again. See "Switches: advanced network devices" section.

The "arpplet" tool used to craft ARP packets doesn't have the capability of listening in a different MAC address not configured at the OS level. It is only an ARP packet generation tool.

- 2.1. Other host real MAC address:

  If the attacker uses other existent host MAC address, this host must be currently down; if it were active, both, the attacker's system and the host will reply to the frames and packets addressed to its "shared" MAC, causing the described attack to fail. Probably "duplicate address" messages will be generated. See ""Duplicate IP address" message" section.

- 2.2. Randomly created MAC address:

  A random or "invented" MAC address allows the attacker to anonimyze its identity as much as possible. It also introduces the most controlled environment, because this MAC address will not conflict with any other address already used in the network.
  The main factor to take into account in this scenario is that the address must be a unicast MAC address (see the "MAC addresses types: Unicast & Broadcast & Multicast" section).

  Although the new MAC is totally unknown, it could be possible to figure out where the attacker is located inside the network in a switched environment.

Signatures based on MAC address selection

Based on the MAC addresses used by the attacker, the switch and host tables will present several scenarios with different signatures. To be able to analyze the following signatures it is necessary to understand how switches work and how they learn MAC addresses and port associations (see the "Switches: advanced network devices" section). In a hub environment only the host table's signatures will be available.

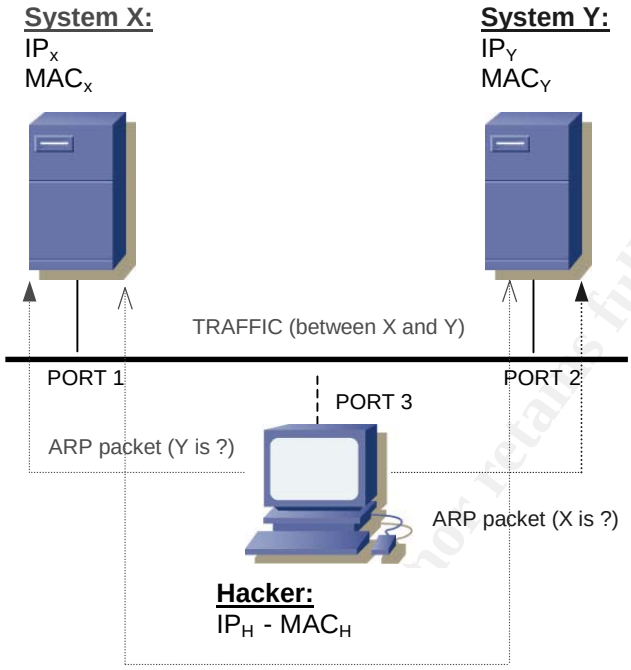


Figure 2.20. Switch scenario

In a future research it should be analyze how ARP tables in switches learn. Due to the fact that switches can be managed, traffic from and to them is possible, so they need to keep its own ARP table. It will be possible to see inconsistencies between the switch CAM and ARP tables that could help to determine an attack is in place.

Based on previous case, 1, where the attacker uses its own MAC address, this will be the ARP and CAM tables:

X ARP table
IPy is at **H**
IPh is at H

CAM table	H ARP table
X at port 1	IPx at X
Y at port 2	IPy at Y
H at port 3	

Y ARP table
IPx is at **H**
IPh is at H

Based on previous case, 2.1, where the attacker uses other host real MAC address ("O") located in switch port 4, keeping its MAC address too, this will be the ARP and CAM tables:

X ARP table
IPy is at **O**
IPh is at H
IPo is at **O**

CAM table	H ARP table
X at port 1	IPx at X
Y at port 2	IPy at Y
H at port 3	IPo at O
O at port 3 & 4	

Y ARP table
IPx is at **O**
IPh is at H
IPo is at **O**

This attack is not possible if switch cannot see the same MAC in different ports, or if switch cannot see two MACs associated to one port, port 3.
Switch will see same MAC in two ports and hosts will see two different IP addresses in the same MAC, only possible with virtual IP addresses in the same NIC.

Based on previous case, 2.2, where the attacker uses a new invented MAC address ("N"), keeping its MAC address too, this will be the ARP and CAM tables:

X ARP table
IPy is at **N**
IPh is at H

CAM table	H ARP table
X at port 1	IPx at X
Y at port 2	IPy at Y
H at port 3	
N at port 3	

Y ARP table
IPx at **N**
IPh is at H

This attack is not possible if switch cannot see more than one MAC in the attacker port, port 3. This situation occurs when hub or/and switches are connected between them. If X or Y need to communicate with H they don't see the same MAC address for different hosts.

Finally, the attacker can even use different MAC addresses ("DX" and "DY") for every target system:

X ARP table
IPy is at **DY**

CAM table	H ARP table
X at port 1	IPx at X
Y at port 2	IPy at Y
H at port 3	
DX at port 3	
DY at port 3	

Y ARP table
IPx at **DX**

Except in this case, in all others if ARP spoofing attack is delivered using broadcast packets, a third host will see the same MAC for 2 (both targets, cases 2.1 and 2.2) or 3 hosts (both targets and attacker, case 1), situation that will improve the chances of detecting the attack.

As can be seen, the different scenarios and the correlation between the systems ARP tables and the switches CAM tables will help to know what is happening in the network and if an attack is taking place.

How to protect against it

The scope of the ARP spoofing attack can be mitigated in several ways, being the main one the reduction of the local network, that is, limiting the environment a given host has access to.

All the explained countermeasures would allow IT personnel reduce the thread and avoid their systems to be compromised, but there is no too much OS vendors can do to fix the vulnerability analyzed, until a new secure substitution of the ARP protocol will appear. The 802.1x protocol could be considered as such.

It doesn't matter if this network segmentation is implemented through switches, new IP subnets (routers), VLANs, Private VLANs… an associated economical and administrative cost will be added, due to the fact that this solutions require more physical or logical elements, increasing the complexity of the environment.

Besides, there is a point in which it is impossible to segment the network in smaller units, because devices need to communicate and performance and management issues are hardly affected, therefore, every network that can be considered a LAN is exposed to this attack, no matter the networking technology used.

The countermeasures that can be applied will be explained from the most simple and obvious ones to the more complex and innovative. All them are focused on protecting one of the two elements affected by an ARP spoofing attack: the end hosts and the network.

Physical security

An ARP poisoning attack can be successful in two situations:

- When an external attacker has owned an internal box. In this case, the countermeasures to apply must be focused on avoiding the exploitation of vulnerabilities that allow external attackers to get into the network.
- The other case is when the attacker is physically placed on the local network. Physical security can help to mitigate this scenario.

Due to the fact that the attacker needs to be located in the same network as the target hosts, because ARP packet don't cross router boundaries, the first step is increasing physical security.

It is necessary to have a strong control about every physical network connector available in the building and security policies will ensure the staff security consciousness. People must be worried about who is sitting next to them.

It is frequent to find IT people thinking that having physical access to the company network is a difficult task. There are lots of examples that show how to do it, mainly using social engineering techniques, as the ones exposed in "The Promotion Seeker" or "The humiliated boss" [KEVIN1]. These examples explain

how easy is entering in a company room impersonating an employee, or how to get access to company faculties when you have previously been an employee.

The controls must focus on prohibiting unauthorized people accessing the building or the different rooms where network connections are available. As a recommendation it will be a good idea to disable all unused switch ports and even configure them in an isolated VLAN, only used for this purpose.

The same or even a worst situation is fight against when wireless networks [WIRE1] are used. The concept of physical network connector is distributed all along the air. The "Authentication: 802.1x" section will describe related solutions for both network types.

Static ARP entries

One way of being prepared against ARP spoofing attacks is through the configuration of system's ARP tables. Through the usage of static ARP table entries it is possible to substitute the dynamic functionality provided by ARP, therefore, avoiding attacks based on impersonation. This solution introduces a huge administrative overhead, presenting some scalability problems in large networks.

It is recommended to apply this method only to critical systems, typically placed in production and server networks, outside the user's desktop network environment. Not only critical host but also network devices are targets of this countermeasure, such as, routers, switches and firewalls.

All OS in the IT industry allow setting static ARP entries through the "arp" command or using similar commands. Some of them (HP-UX 11, Linux, Solaris…) allow the population of the ARP table when booting from a well-known file containing the "official" MAC-IP addresses pairs. For example, Unix variants use the "-f" option of the "arp" command. This option is not available in Windows 2000.

Apart from that, remember that static entries are not always permanent. There have been some system bugs along time that affect how static ARP entries behave when a dynamic packet is received. Sometimes, the entry is updated despite it was supposed to be static [YURI2].

To avoid this you need to apply a new concept, "immutable" entries. For example HP-UX 10.20 implemented it. The ARP cache static entries are modified when the ARP code detects a different MAC address. To avoid this and get profit of the new concept you need to apply the PHNE_18061 OS patch:

> "ARP has been enhanced to add immutable ARP cache entries that cannot be dynamically changed. Feature is called: it's called "immutable ARP".

It adds the "inmute" option to the ARP command.

In Windows a similar behaviour was detected [FLAK1] due to an operating system bug [MS-KB4], so static ARP entries are overwritten too.

Finally, some OS, as Unix, can be configured to remove the ARP processing on the network interfaces, using the "ifconfig –arp" command. In this case, ARP static entries are needed to be able to communicate with the rest of the world.

Encryption

To avoid traffic being successfully sniffed and analyzed once it has been redirected through the ARP spoofing method, or to stop more advanced attacks, as the one described in the "Smart IP spoofing" section, strong encryption techniques must be used.

There are several standard protocols that use cryptography to protect the data they transport, like IPSec, SSH or SSL. The usage of these protocols instead of clear text ones is totally recommended.

It is surprising to see how many unencrypted protocols are in use nowadays in production and critical networks although their encrypted equivalents are available for free.

Filtering devices

In several references the standard Linux packet filtering tool, "iptables", is used to denote a way of protecting against ARP forged packets. It is true that "iptables" can filter packets based on MAC address information but it should be explained that it is only capable of managing IP packets, so it cannot filter ARP packets.

```
# iptables -A INPUT -m mac --mac-source 0E:0E:0E:F1:F2:F3\
                    -s server -j ACCEPT
```

It must be clarified that "iptables" is based on the Linux netfilter module, and this is implemented as a series of hook points inside the kernel networking stack. Until now only three protocols are supported, IPv4, IPv6 and DECnet. Due to the fact that ARP is not mentioned at all, there is no ARP processor inside netfilter.

When an ARP request or reply with any MAC address arrives to an interface, the filtering module inspect the next protocol field, see it is ARP and deliver the packet to the networking stack without processing it.

However it can filter any IP packet where the Ethernet header has a specific source MAC address so that only the proper traffic gets through. Therefore, this filter will avoid advanced attacks based on ARP spoofing, that is, the targets ARP tables will be poisoned but any other IP traffic exchanged between them will be dropped.

The first thing a layer 3 filtering device, typically all firewalls, looks for is the Ethernet frame type field, that specifies the next layer protocol. Generally speaking, they check if the next protocol is IP to process the packet based on its security policy.

Cisco is working in a layer 2 ARP firewall for its high-end switches series [HACK1].

Switches: advanced network devices

Although switches are typically considered as security devices, they are useless against the ARP spoofing attacks. Let's analyze how they work, their features and why they cannot stop ARP poisoning.

Initially, networks were conformed just by cables, then bridges allow expanding the network distance and finally hubs were introduced, simplifying the network management and allowing different physical topologies, like the star topology.

Then, trying to improve the network performance switches were designed, isolating every port connection and creating several collision domains. After that, local networks evolved, and the VLAN concept was introduced. From a logical point of view each VLAN is as a different switch, and the VLANs were created to be able to expand a LAN over different switches and physical locations.

Therefore, in today networks, a group of ports, devices, are placed in a VLAN, that is, in the same broadcast domain, this is why typically every VLAN is a separate IP subnet.

A switch improves performance because it dynamically learns were every device is connected and when traffic must be sent to a specific device, the switch only sends the data to the port the device was plugged into. Other hosts, that is, switch ports, don't receive these frames.

To be able to learn were every device is located the switch has a table called the CAM table, Content Addressable Memory. This table contains the association between a device, identified by its MAC address, the port where it is connected and the VLAN it belongs to.

Let's explain how does the CAM table work [GILLI1]. When a device transmits traffic, the Source Address inside the Ethernet frame header contains its MAC address, so the switch is capable of learning that in this port there is a specific device. When some traffic must be sent to this host, the switch search its CAM table and only sends the traffic to the referenced port. While the switch doesn't know where a device is, because it has never generated traffic, it will send the traffic to all ports, acting as a hub.

There are not too much differences related to the ARP spoofing attack between a hub and a switch environment, being possible to run it in both scenarios.

Typically, the switch needs to manage two tables, the ARP and the CAM table, with different behaviours. The ARP cache timeout for Cisco IOS devices is 4 hours (see "ARP parameters by operating system" section). However Cisco switches CAM table times out in 300 seconds by default. This may result in some unicast IP traffic being flooded when the CAM entry is lost.

The CAM table

In the same way the ARP table can be manually managed, the CAM table can be manipulated. We will take Cisco switches as a reference to analyze what commands there are available to change the CAM table.

Both, CatOS [CATC1], Cisco Catalysts, and IOS [IOSC1], Cisco 2924XL IOS based, commands are showed. There is a Cisco document that details the main differences between layer 2 operations in CatOS and IOS devices [CAIO1], like the high-end switches family, Catalyst 6000/6500.

IOS command	mac-address-table dynamic mac-address-table secure mac-address-table static
CatOS command	set cam {dynamic | static | permanent}
Description	Use the set cam command to add entries into the CAM table.
Default value	N/A

Examples	`Console> (enable) set cam static 00-0c-0c-10-10-10 1/3`
	`Static unicast entry added to CAM table.`
	
	`Console> (enable) set cam permanent 01-40-40-aa-aa-aa 1/1,2/1,3/1-12`
	`Permanent multicast entry added to CAM table.`

CatOS CAM entries:

- dynamic: subject to aging.
- static: not subject to aging. They will remain in the table until the system is reset.
- permanent: these entries are stored in NVRAM until they are removed by the "clear cam" or "clear config" command.

IOS defines 3 types of CAM entries:

- dynamic.
- static: are not assigned to a port, but instead to the system. When a packet is received on the in-port, it is forwarded to each port in the out-port-list.
- secure: can only be assigned to one port at a time.

IOS command	mac-address-table aging-time
CatOS command	set cam agingtime
Description	Use the set cam agingtime command to set the aging time for the CAM table. It sets the period of time after which an entry is removed from the table. It applies to all VLANs by default. IOS defines it as the time a dynamic entry is in the table, from the time it was used or last updated.
Default value	300 seconds. Range: 0 (disabled) to 1,000,000.
Examples	`Console> (enable) set cam agingtime 1 600` `Vlan 1 CAM aging time set to 600 seconds.`

IOS command	clear mac-address-table
CatOS command	clear cam
Description	It deletes a specific entry or all entries (of an specific type) from the CAM table.
Default value	N/A

Examples	`Console> (enable) clear cam 00-44-44-aa-aa-aa`
	`CAM table entry cleared.`
	
	`Console> (enable) clear cam dynamic`
	`Dynamic CAM entries cleared.`

IOS command	show mac-address-table
CatOS command	show cam show cam agingtime
Description	Use the "show cam" command to display the CAM table. It accepts several parameters to filter the output displayed. Use the "show cam agingtime" command to display CAM aging time information for all configured VLANs.
Default value	N/A

Examples	`Console> (enable)` **`show cam dynamic 1`**
	`VLAN Destination MAC Destination Ports or VCs`
	`---- ----------------- -------------------------`
	`1 00-44-44-60-cc-99 4/1`
	`1 00-44-44-b0-bb-88 4/1`
	`1 00-00-0c-ff-ff-ff 4/1`
	`Matching CAM Entries = 3`
	`Console> (enable)`
	
	`Console> (enable) show cam agingtime`
	`VLAN 1 aging time = 600 sec`
	`VLAN 100 aging time = 300 sec`
	`VLAN 200 aging time = 300 sec`
	`...`
	`Console> (enable)`
	
	`IOS:`
	`Switch#` **`show mac-address-table`**
	`Dynamic Addresses Count: 19`
	`Secure Addresses (User-defined) Count: 0`
	`Static Addresses (User-defined) Count: 0`
	`System Self Addresses Count: 29`
	`Total MAC addresses: 48`
	`Non-static Address Table:`
	`Destination Address Address Type Destination Port`
	`------------------- ------------ ------------------`
	`--`
	`0000.0c5c.e176 Dynamic FastEthernet0/8`
	`0000.2424.96b4 Dynamic FastEthernet0/8`

Typically the CAM table has a limited size although it can be modified. This fact can be used to develop other attacks as the one implemented by the dsniff "macof" utility [DSNIFF1].

Port security

Port security [CISPO1] [CISPO2] is a switch feature implemented by different manufacturers, such as Cisco, that allow limiting the number of MAC addresses in a given port. It really specifies the maximum number of addresses the port can learn.

Apart from that, the specific MAC addresses connected to this port can be manually configured or dynamically learned.

When an invalid MAC is detected, a policy violation, the switch is able to alert and act through different methods: SNMP, blocking the port to only the invalid MAC or to all devices, shutdowning the port.

Cisco CatOS command:	set port security
Cisco IOS commands:	port security OR switchport port-security

The Cisco IOS has a command to visualize the CAM table and port security status:

```
Switch# show port security fa0/3
Secure Port      Secure Addr    Secure Addr  Security   Security Action
                 Cnt (Current)  Cnt (Max)    Reject Cnt
---------------  -------------  -----------  ----------  -------------
FastEthernet0/3  1              3            0           Send Trap
```

The Cisco CatOS uses the same command:

```
Console> (enable) show port security 3/20

Port  Security Violation Shutdown-Time Age-Time Max-Addr Trap     IfIndex
----- -------- --------- ------------- -------- -------- -------- -------
 3/20  enabled  shutdown          300       60       10 disabled     920

Port  Num-Addr Secure-Src-Addr   Age-Left Last-Src-Addr     Shutdown/Time-Left
----- -------- ----------------- -------- ----------------- ------------------
 3/20        3 00-e0-22-33-44-00       60 00-e0-22-33-44-00       no          -
               00-11-22-33-44-55        0
               00-11-22-33-44-66        0
```

There are some situations where this method cannot be applied, such as when a core switch port interconnects with another hub or switch port, creating a switch fabric. The traffic of all the devices connected to this other hub/switch must be allowed through the port, so it is not possible to limit the number of devices. As can be seen it has associated a big management and performance overhead.

This solution is only effective against the ARP spoofing attack when the attacker is using multiple MAC addresses (see "Signatures based on MAC address selection" section) and the switch policy limits to one host per port. If attacker is using its MAC address this feature won't stop the attack at all.

The Port Security feature is really effective against MAC flooding attacks [DSNIFF1].

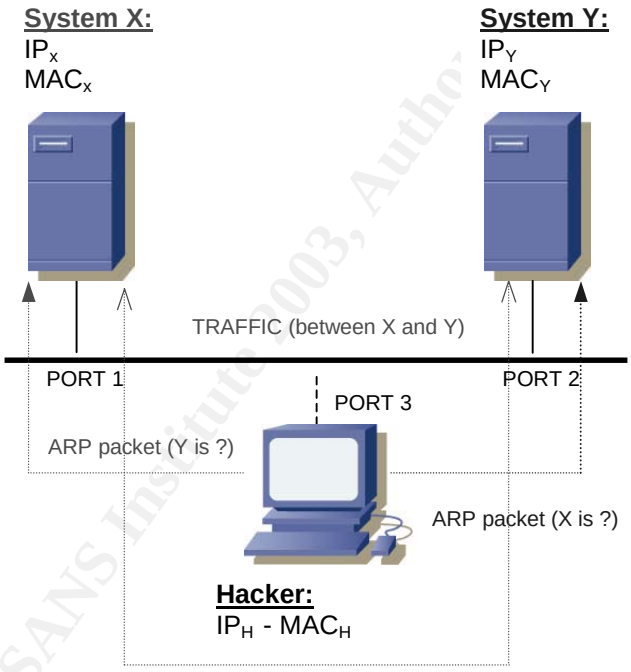
Blind ARP spoofing

There is an attack based on redirecting traffic forging the switch CAM table, called Taranis [TARA1], due to the tool name that implements it. It is based on changing the Source address of some Ethernet frames to populate the CAM table with the desired information.

Taranis has no relationship with the ARP protocol, but based on this idea, and how the CAM table is populated and the different MAC addresses the attacker can use (see "Signatures based on MAC address selection" section) a new attack concept is proposed.

This attack has been called "Blind ARP spoofing" because the switch is completely blind about from which ports the attack come from. An example will be developed based on the figure used in the previously referenced section, Figure 2.20, duplicated here to facilitate the exposition to the reader.

The attack is based on sending ARP packets where the MAC address inside the Ethernet header and inside the ARP payload are different, and in the fact that most OS learn new ARP entries based on the ARP payload information and not on the Ethernet header addresses.



Let's use ARP reply packets, although using ARP requests it will work as well if they populate the targets ARP tables. Attacker will use a different new MAC address in the Ethernet header, "A".

The attacker should never generate traffic using this new MAC address as the source address in the Ethernet frame header. In this way, the switch will never learn were the attacker is and its MAC address will never appear in the CAM table.

The attacker generates an ARP packet to populate X ARP table as if it was send by A:

ARP reply		
A		
X		
0x0806	0x0001	0x0800
6	4	2 (reply)
H		
IPy		
X		
IPx		

He also generates this traffic to the other end target. Therefore, X and Y will think the other end target is at H, but switch have not seen where H is, instead, he knows about A.

For example, let's analyze one way of the connection: when X sends traffic to IPy, it arrives at H. The switch doesn't know where H is, so it floods the packet to all ports, including the attacker port, port 3. When the attacker forwards the traffic it sends traffic from A to Y at the Ethernet header level, so the switch never sees H. The real association between a device and a port is at the physical level and not at the link, Ethernet, level.

X ARP table
IPy is at **H**

CAM table	**H ARP table**
X at port 1	IPx at X
Y at port 2	IPy at Y
A at port 3	

Y ARP table
IPx is at **H**

This attack generates too much confusion because when analyzing the ARP and CAM tables there is no way to associate A and H MAC addresses, and there is no way of knowing which port H is in.

The only way of detecting this attack is through the analysis of network traces and checking incongruences between Ethernet header and ARP payload MAC addresses. Tools like "arpwatch" can detect this anomaly.

The attack is not influenced at all based on how the switch manages its CAM table [TARA1] (last packet updates, a timeout must expire to update an entry or it has a fixed value), because the MAC address used, A, is totally new.

An additional analysis related to the signature of this attack, see "Signatures based on MAC address selection" section, reflects that if instead of using a new MAC address, A, the Ethernet MAC addresses forged would be the other target MAC address, X or Y, the attack will be almost evident cause the same MAC address will be seen in several ports and the attacker port would show both target MAC addresses:

X ARP table
IPy is at **H**

CAM table	**H ARP table**
X at port 1	IPx at X
Y at port 2	IPy at Y
X,Y at port 3	

Y ARP table
IPx is at **H**

Additionally, this attack, using the other end MAC, should suppose that the switch accept the same MAC in two different ports, a switch mode called "Limited Hub Mode" [TARA1].

"Duplicate IP address" message

One of the tasks performed by the ARP module is to passively detect other devices trying to impersonate the local host, that is, hosts claiming to have the same IP address as the local one. Typically it will respond to ARP requests, both standard and gratuitous, for this IP address.

This feature is commonly referenced as DAD, Duplicate Address Detection.

Some special cases visualized during this paper's test have been described under the "Conclusions" section in "ARP packet taxonomy tests".

Cisco IOS

Cisco devices save duplicate messages by default in an internal logging buffer and they are showed in the system console too. It is possible to configure routers and switches to send this messages to a syslog server.

The buffer can be inspected with the following command:

```
Router#sh logging history
Syslog History Table:1 maximum table entries,
saving level warnings or higher
 6 messages ignored, 0 dropped, 0 recursion drops
 3 table entries flushed
 SNMP notifications not enabled
   entry number 4 : IP-4-DUPADDR
    Duplicate address 192.168.1.1 on Ethernet0/0, sourced by 0010.aaff.2233
    timestamp: 126944
Router#
```

HP-UX 10.20

HP-UX 10.20 didn't generate any duplicate address message during the tests and even an ARP bug was found.

When a broadcasted gratuitous ARP request packet is received (asking for the IP address of the host and coming from the same IP address), instead of generating a "duplicate address" massage, the host reply with an ARP reply packet with spurious values.

```
Frame 1 (60 on wire, 60 captured)
    Arrival Time: Jun 10, 2003 13:49:47.974304000
    Time delta from previous packet: 0.038706000 seconds
    Time relative to first packet: 4.168291000 seconds
    Frame Number: 1
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II
    Destination: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
```

```
    Source: 00:0e:0e:00:00:02 (00:0e:0e:00:00:02)
    Type: ARP (0x0806)
    Trailer: 00000000000000000000000000000000...
Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: 00:0e:0e:00:00:02 (00:0e:0e:00:00:02)
    Sender IP address: 192.168.1.1 (192.168.1.1)
    Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.1 (192.168.1.1)
```

```
Frame 2 (42 on wire, 42 captured)
    Arrival Time: Jun 10, 2003 13:49:47.974341000
    Time delta from previous packet: 0.000037000 seconds
    Time relative to first packet: 4.168328000 seconds
    Frame Number: 2
    Packet Length: 42 bytes
    Capture Length: 42 bytes
Ethernet II
    Destination: 00:0e:0e:00:00:02 (00:0e:0e:00:00:02)
    Source: 00:0e:0e:00:00:03 (00:0e:0e:00:00:03)
    Type: ARP (0x0806)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Sender IP address: 0.96.176.32 (0.96.176.32)
    Target MAC address: 00:0e:0e:00:00:02 (00:0e:0e:00:00:02)
    Target IP address: 192.168.1.1 (192.168.1.1)
```

It is not clear enough from where this IP address has been extracted.

HP-UX 11.X

HP-UX 11.x ARP module watches passively for hosts impersonating the local device, like a system that responds to an ARP mapping request for the local host's address, and generates the following message:

```
duplicate IP address!! sent from Ethernet address: 00:10:10:01:02:03.
```

This message is printed on the console screen and into the syslog file.

Linux

Linux also generates "duplicate address" messages and saved them in the default syslog file. Syslog file will show a new entry:

```
"duplicate IP address!! sent from Ethernet address:
00:10:10:01:02:03."
```

Windows

Duplicate address detection applies to stations with either static or DHCP configured IP addresses, being Windows NT [WINNT1] or Windows 2000 [W2000]. Windows is probably the OS that has the more complex duplicate address detection mechanism and fortunately generates visible messages.

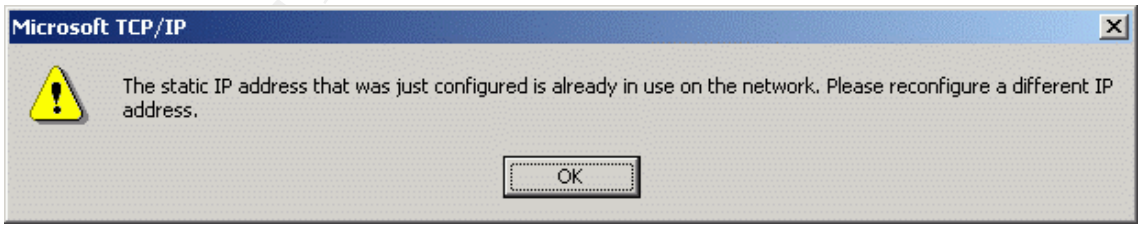
When the stack is first initialized or when a new IP address is added, by default, 3 gratuitous ARP request packets are broadcasted for the IP addresses of the local host.

If another host replies to any of these ARP packets, then there is an address conflict because this IP address is already in use. The node that was turn on first is called the "defending node" while the one that was powered on later is called the "offending node".

When this happens, the offending node still boots, however, it prevents its TCP/IP stack for being initialized for the interface containing the offending address and a Windows pop-up error message is displayed. If the host that is defending the address is also a Windows-based computer it will also display the same type of pop-up error message; however, its interface will continue working. This scenario is the same as when a duplicated IP address is configured in a network interface and it is activated (see "Activating/Deactivating network interfaces" section).

The defensive host generates the following message:

The offensive host displays a different message while it brings its interface down:



In order to minimize the damage provoked to the other host's ARP caches, the offending [W2000] computer resends an additional ARP broadcast to restore the original values in the ARP caches of the other systems.

After transmitting the ARP reply, the defending [WINNT1] system ARPs for its address again to make sure the rest of the devices composing the network will maintain the correct mapping for the address in their ARP caches.

The documentation shows a mismatch at this respect, as it is not clear enough who is in charge of resending the correct MAC notification. After testing it was confirmed that, in a Windows 2000 SP3 lab environment, it is the defensive host the one that sends a gratuitous ARP request message with its MAC address.

A more complex scenario is presented when a computer using a duplicate IP address is started before getting plugged to the network, so no conflict would be detected. However, as soon as the system is connected to the network and sends the first broadcast ARP request to ask for another IP address, the Windows NT/2000 system with the conflicting address will detect the problem. Every time a defensive host sees a standard broadcast ARP request coming from an offensive system, the same error window keeps on popping-up, almost continuously in normal situations.

When a pop-up duplicate address detection message is displayed a detailed event is added to the system log. A sample event log entry is shown below:

```
"The system detected an address conflict for IP address 192.168.1.100
with the system having network hardware address 0E:0E:0E:00:01:00.
Network operations on this system may be disrupted as a result."
```

The main problem in identifying an ARP attack through this method is that typically anyone notices this kind of messages, except in Windows boxes (you must be mad if you ignore them ;-)), because they are silently notified to the syslog service.

Solaris 8

Solaris, as its other Unix variants, generates messages into the syslog file. Its message is a bit different from the Linux and HP-UX OS:

```
WARNING: IP: Hardware address '00:02:03:03:02:01' trying to be our
address 192.168.132.145!
```

A different message will appear if ARP is used for Proxy ARP functionality, that is, it has been used to create a published entry, and some other host on the local network responds to mapping requests for the published ARP entry.

```
IP: Proxy ARP problem? Hardware address '00:02:03:03:02:01' thinks it
is `192.168.132.145'
```

NIDS

The Network IDS systems, NIDS, can be used to monitor the ARP traffic and alert about any strange situation observed. The most commonly used open source NIDS, Snort [SNO1], included a new preprocessor, called Arpspoof.

The Snort configuration file allow defining a MAC-IP addresses pair database reflecting the well-known associations:

```
[ArpSpoof]
#
# Search anomaly in ARP request.
#
# The "directed" option will result in a warn each time an ARP
# request is sent to an address other than the broadcast address.
#
# directed;
# arpwatch=<ip> <macaddr>;
...
```

There are other NIDS that implement similar capabilities, such as "prelude-nids" from Prelude-IDS [PREL1] which also defines IP associated with MAC, and other ARP attacks through a plugin called "ArpSpoof". It generates messages such as "[Attempted ARP cache overwrite attack...]".

Some IDS will alert if a huge amount of ARP traffic is seen in the network, a typical situation where the attacker needs to constantly refresh the poisoned ARP tables.

Some of the network management suites, as the HP Network Node Manager, NNM, or the Ciscoworks applications, that monitor and manage the whole network topology are capable of detecting strange ARP behaviours as the "duplicate address" problem. AS stated in the "HA solutions" section, the failover solutions may confuse the ARP monitoring applications [NNM1].

Therefore, integrating the management network tools with the security network IDSes is well worth.

HIDS

The most used and referenced application to detect ARP spoofing attacks is "arpwatch" [ARPW1], a host based IDS, HIDS, focused on detecting anomalies in the ARP traffic received by a system.

The "arpwatch" tool can notify protocol anomalies through e-mail and/or syslog. The tool needs a reference database (by default the "arp.dat" file) that contains the entire official IP-MAC addresses pairs. If it is not provided it will create one on the fly based on the information learned from the network traffic.

The tool has the possibility of reading network traces in "tcpdump" format for offline analysis. This is an awesome feature that would allow running security audits over all the network traces captured in a production network or when running other tests described in this research. This will allow to analize which packets will be detected by "arpwatch" and which won't.

The special cases this tool detects are:

- Host claiming to be the broadcast addresses, both, at layer 2 FF:FF:FF:FF:FF:FF or at layer 3, 255.255.255.255. Generates the messages: "Ethernet broadcast" or "IP broadcast".
- Source Ethernet or ARP address equal to all zeros or all ones. Generates the message "Ethernet broadcast" too.
- Different Ethernet and ARP addresses. Generates the message "Ethernet mismatch".
- Flip-flop situations. See messages bellow.

This is a basic list of the report messages generated by this tool, extracted from the tool documentation:

- New activity: This Ethernet/IP address pair has been used for the first time in six months or more.
- New station: The Ethernet address has not been seen before.
- Filp-flop: The Ethernet address has changed from the most recently seen address to the second most recently seen address.
- Changed Ethernet address: The host switched to a new Ethernet address.

Messages generated in the syslog:

- bogon: The source IP address is not local to the local subnet.
- Reused old Ethernet address: The ethernet address has changed from the most recently seen address to the third (or greater) least recently seen address. (This is similar to a flip-flop.)

"arpwatch" is included by default in several Linux distributions, like Red Hat: RPM packet in Red Hat 7.1 - arpwatch-2.1a11-17.7.3.2. The default configuration directory is "/usr/operator/arpwatch", but this is distribution and OS dependant, Red Hat 7.3 uses "/var/arpwatch".

"arpwatch" also includes a tool called "arpsnmp" that is able to deliver the same kind of monitoring service over devices and platforms where the tool has not been ported to, for example Cisco boxes. It can extract the ARP information of a Cisco device through SNMP and analyze its status. This tool also have a "-f" option to be able to run it over old captures for auditing purposes.

An advanced "arpwatch" version is proposed in which not only the ARP traffic will be analyzed, but the generic IP traffic, looking for MAC-IP pair mismatches. This will use IP broadcast and multicast traffic to visualize the real system MAC address, inspecting the "Ethernet Sender address" field and the "IP source address", and checking them against the IP-MAC reference database.

There is also a version ported to the Windows platform, called "Winarpwatch" [WINARP1].

There are other specific solutions, like a Linux patch [LXPA1] that defines the kernel behaviour when changes in correspondence between MAC and IP addresses are detected.

This patch applies to Linux kernels 2.4.18 and .19 and enforces resisting ARP spoofing. When applied it brings a new sysctl parameter:

```
net.ipv4.neigh.<interface name>.arp_antidote
```

Parameter value	Description
0	It corresponds with the Linux standard behaviour, where the ARP cache will be silently updated.
1	It just reports the attack and ignores the spoofing attempt.
2	ARP cache record will be marked as "static" to prevent attacks in the future.
3	ARP cache record will be marked as "banned", no data will be delivered to the attacked IP anymore, until system administrator unbans the ARP record updating it manually.

Values between 1 and 3 correspond to the new "verification" behaviour where the Linux kernel will send ARP request packets to test if there is a host at the "old" MAC address. If such response is received it lets us know than one IP pretends to have several MAC addresses at one moment, probably caused by an ARP spoofing attack.

TTL signature

From a remote system point of view and if there are no redundant paths through the intermediate networks, the TTL number in the IP packets coming from an specific sender system would have a fixed value.

If the attacker is in the middle of the connection, it will need to forward the packets to its final destination and the TTL should be decreased by one. This would allow detecting something wrong is happening.

TTL is decremented when attacker uses standard forwarding functionality, like the Linux kernel, but if a user level code is used instead, like "fragrouter" [FRAG1] the TTL can be modified too through the "ip_ttl" directive.

But this is not as easy as described in real world networks where redundant paths are needed for load balancing and high availability, so the TTL of the packets interchanged between two systems are rarely always the same.

Authentication: 802.1x

As explained in the "Physical security" section, it has not sense to provide access to unauthorized people to the network resources. In the same way security is always enforce at the application level, not allowing any user to access a given application without have being previously identified, at least with

a username and a password, there is no reason why the access to the network itself should be different.

In nowadays network infrastructures anyone can plug a new device into the net and be able to use the network resources: from bandwidth to the company Internet access. Besides, today networks are typically prepared to offer mobility and flexible services, such as DHCP, providing new users with IP addresses and other configuration elements.

In wireless networks [WIRE1] the situation is even worse because it is easier for an unauthorized entity to get network access even from outside the company facilities. However, thanks to wireless technologies, new security methods have been developed to enforce the network as an application, forbidding its usage to unauthorized people, such as the new standard IEEE 802.1x [SEC2005] [802.1X], defined in December of 2001.

The 802.1x is a link layer protocol that increases LAN security by deploying port based network access control, in order to effectively leverage identity based access control and policy enforcement, including AAA (authentication, authorization and accounting), and logging.

The 802.1x protocol is used to transport higher-level authentication protocols, as for example, EAP payloads, Extensible Authentication Protocol. EAP is specified in RFC 2284 and it is a flexible protocol used to carry any authentication information and is typically used with other protocols, such as 802.1x and RADIUS.

The protocol applies to both, wired (802.3) and wireless (802.11) networks:

- EAPOL: EAP over LAN
- EAPOW: EAP over wireless

The switch, or access point in wireless networks, becomes the man in the middle between the 802.1x/EAP clients and the authentication databases, like the RADIUS servers. It uses concepts previously used in wireless networks, such as network identifiers (SSID) or dynamic WEP keys.

The typical packet looks like the following:

Ethernet header	**802.1x header**	**EAP payload**

There are several EAP authentication options specified in the standard, being the most used:

- EAP-MD5: MD5 hashed username and password (challenge-response).
- EAP-OTP: One Time Passwords.
- EAP-TLS: Strong authentication based on SSL and PKI certificates. Of course, this is the recommended solution.
- EAP-MSCHAPv2: MSCHAPv2 username and password challenge-response.

- LEAP: username and password authentication.
  The LEAP protocol, Lightweight EAP, has been broken using a fast dictionary attack in the last DefCon conference [LEAP1].

When a device needs to connect to the network it plugs into a port and an 802.1x login request is generated from the network, such as a switch. The new device offers its credentials, through 802.1x packets, which are then transferred by the switch to an authentication server in an encrypted way, using RADIUS packets:

- If it is a valid device, then the switch applies the associated configuration to the port, like the VLAN it should belong to, and the port is enabled. Other VLAN information, as QoS features or VLAN ACLs can be applied.
- If the device credentials are not valid or it is a non 802.1x capable device, then the authentication server rejects the access or the login requests timeout, respectively, and the switch applies the default policy if any. For example, this policy could put the device in a quarantine zone, like a DMZ, where privileges and access are extremely reduced.

It is absolutely recommended to use this protocol instead of other authentication methods like IP addresses or clear text passwords.

The protocol has been implemented in Windows XP and Windows 2000 using a specific HotFix (see MS KB article 313664). Linux offers an open source solution [LX802] and other OS, like Solaris and Windows NT must use third party solutions [MEET1].

Private VLANS

There are several security solutions focused on enforcing the network security policy, like firewalls, ACLs, VPN devices… but they don't apply to devices placed on the same LAN. To be able to add another security level inside the LAN, Private VLANs, PVLANs [PVLA1], were created.

The main scenario were this solution is really effective is in the DMZ networks. Typically these are insecure networks, exposed to Internet, offering public services. A server located in a DMZ used to receive external requests, from Internet, and it processes them and reply or, it needs to contact an internal server to obtain some information and reply the end client.

DMZ servers are not supposed to talk between them or initiate external connections, but these requisites are not enforced. Therefore, if a DMZ server is compromised it is possible to attack other servers in the same network segment. To avoid this situation, PVLANs can be used, defining the traffic flows allowed in the segment and controlling the packets exchanged between servers in the same VLAN.

Extending this DMZ design to a normal network segment, it is possible to perform an ARP spoofing attack if the evil system is in the same LAN as the target system. PVLANs can restrict which systems can establish conversations,

therefore blocking the possibility of sending spoofed ARP packets between them. This is a very restrictive radical solution because inter-host communication is disabled: it is like placing every host in a different subnet.

PVLANs are a new feature that allow dividing traffic at layer 2, isolating traffic into different communities and creating different layer 2 networks in the same VLAN or IP subnet. The layer 3 network is not modified.

To be able to differentiate traffic flows inside the same VLAN, new concepts are introduced:

- Primary VLAN: it bundles multiple secondary VLANs. It corresponds to the previous VLAN concept.
- Secondary VLAN: they represent the new subnetworks inside the VLAN, and could be defined as traffic flows between specific hosts. There are three types of secondary VLANs: isolated, community and two-way community.

New port definitions associated to these VLANs types are used:

- Promiscuous port: it is able of carrying traffic for primary and secondary VLANs. Routers and firewalls are connected to these ports to be able to forward traffic to all hosts in the subnet.
- Port mapped to a secondary VLAN: traffic is only exchanged between the same community VLAN. End hosts are connected to this type of port, and they can only exchange traffic through the primary and **its** secondary VLAN.
  Host will be capable of communicating with routers and firewalls (placed in promiscuous ports) but not with other hosts in a different community or in the same isolated secondary VLAN.

When VLANs were initially designed the trunking concept was created: a trunk is an inter-switch link that can carry on multiple VLANs. The same concept is applied here: a primary PVLAN is able to encapsulate multiple secondary PVLANs.

This countermeasure can only be implemented at the switch device because all communications take place in the same IP network, where forwarding devices like routers or firewalls don't act. Only personal firewall at the host level can offer a filtering mechanism at this level, but all the existent solutions work at the layer 3 level.

On one hand, this solution has no associated performance penalty, because switches implement it at the hardware level but, on the other hand, there is a huge administrative overload to define and maintain the different primary and secondary VLANs. It is recommended to apply this concept in critical network segments.

PVLANs has a design limitation, where any host will be able to jump from one isolated secondary VLAN to another using the LAN router as a bridge, due to the fact that the router can forward traffic coming from the VLAN to the same

VLAN again. PVLANs can work at layer 2 but cannot avoid processing at layer 3, as the router does. This situation can be solved configuring VACLs for the router allowing the traffic addressed to itself, denying all the traffic were source and destination are the same subnet it belongs to, and allowing any other traffic to or from other networks.

This limitation doesn't affect the ARP spoofing attack because ARP is a layer 2 protocol, routers don't forward ARP packets, so the VACL is not needed to block this attack.

PVLANs are Cisco platform and OS version dependent. In some devices they are called "port protected".

A successful security network design must enforce two main elements:

- Who needs to talk with whom: Private VLANs.
- What traffic should be exchanged: VLAN ACLs (see next section).

VACLs

VLAN Access Control Lists, VACLs [PVLA1] provide further control over traffic from or to a specific network segment. They act as the router ACLs or firewall rules but at layer 2.

These are commonly known as wire-speed port ACLs because they are implemented at the hardware level and no performance penalty is introduced. Cisco uses the PFC, Policy Feature Card, to do so. There is a known vulnerability where the PFC is not able of managing IP fragments.

VACLs understand PVLANs, so they can be applied to primary or secondary VLANs independently.

Part 3 – The Incident Handling Process

Due to the research nature of this paper, and based on the fact that this document doesn't describe an actual security incident in which I took part, the Incident Handling Process, IHP, covered by this section will focus on providing recommendations to help incident handling teams addressing the attack described in Part 2.

This section provides tips for all the six stages of the IHP in a real life production IT environment belonging to a hypothetical big company in Spain, my country, based on my penetration testing and security auditing experience acquired in the last 4 years.

Preparation

Having an incident handling (IH) policy, defined procedures and a multi-disciplinary IH team is crucial for successfully managing security incidents.

Nowadays, IT environments are becoming conscious of the importance of IT security, but it is not one of the top IT priorities yet. Therefore, all the available countermeasures, described in "How to protect against it" section, are not applied; only some of them are effectively used.

The company IT security personnel is focused on administering the common basic security solutions, like the firewall, the mail server antivirus software, the web filtering devices and proxies and, perhaps, some IDS systems.

It is very rare to find a well-established IHP, so incident handling teams are conformed on the fly when an incident occurs, mixing system, networking and security engineers, based on the nature of the attack.

However, companies have detailed policies against the traditional incidents, such as thefts, where physical security is affected, having several countermeasures in place: security guards, control barriers in the facilities entrances, surveillance cameras, etc. Therefore management should support IT security at the same level.

Thus, security policies and procedures to deal with incidents are not well defined and there are no skills and resources available. Therefore companies present a poor incident handling preparation status.

In the same way, the relationship with the law enforcement is not established, and knowledge about their interests, the cases they work in, and interfaces with them have not been developed. The incident is typically managed in-house, unless it leaks to the media.

Let's focus in the ARP spoofing attack specific aspects.

The most important thing to be prepared is having an accurate and updated source of information about all the different pieces that conform the company's network: every device able to communicate (hosts, network devices, printers, handhelds…) should be identified and its MAC and IP addresses if they have been statically assigned, registered on a secure centralized database.

It is important to have information that will help to determine important details about the systems affected and the attacker's resources. For instance, in order to get the manufacturer associated to a given MAC address, the IEEE MAC addresses database is a must [IEEE1]. Remember that the first portion of the MAC address is known as the OUI, which uniquely identifies the card manufacturer, which would help to get the type of system the attacker is in. This information will be useful if the attacker has not modified the MAC addresses used to launch the ARP poisoning attack, which, unfortunately, is not a hard task.

Apart from the countermeasures described in Part 2, a network audit trail will be a helpful tool. It will register all traffic (in an overpopulated network the most common traffic could be discarded, such as web, mail and dns), or for the purpose we are interested in, at least all the ARP traffic. The network traces could provide facts about which packet travelled the network and poisoned the systems in case an incident occurs.

Identification

The first aspect that must be determined in ARP spoofing cases is if the events are associated to a real ARP spoofing attack.

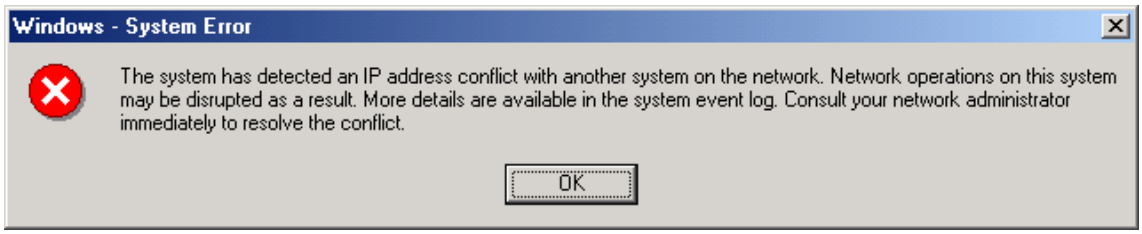
It is very common to find the same events an ARP spoofing attack introduce due to a network misconfiguration: duplicate address messages, the same IP having different MAC addresses, poor network performance, lost of network connectivity…; so it is very important to determine if an event constitutes an incident.

The last two events are frequently appreciated when ARP spoofing attacks, take place, so it is important to make users aware of this fact and train them to notify strange networking behaviours.

This fact is very negative from the detection perspective, because it is usual to think that this type of events are associated to network problems, and the ARP spoofing attacks are not considered as a possible reason. From the user point of view, any application, that use the network, running on the target system can have an unusual behaviour due to the redirection introduced by the attack.

The most common countermeasures that work effectively are:

- Physical security, so that not anyone can enter the company buildings and connect a laptop to the network. Social engineering methods can potentially bypass these controls.
- "duplicate messages" pop-up windows in Windows hosts:



- Sometimes syslog messages in Unix devices are monitored by some network management application, so the messages are effectively read by some operator, but if this solution is not used, the syslog messages slip by:

```
duplicate IP address!! sent from Ethernet address: 00:01:02:03:04:05.
```

- All acceptable LAN networks use switches and VLANs, so the scope of the attack is restricted to a specific set of devices.

Other countermeasures like static ARP entries, encryption in all the main protocols used (although it seems unbelievable), strong authentication, or ARP monitoring solutions, like IDS, are rarely used due to the management overhead they introduce.

If a very detailed procedure to deal with ARP issues is not defined, being able to detect this type of attack, takes long periods of time, until other common networking scenarios have been discarded. If no monitoring solutions are used, an ARP attack could even go totally unnoticed.

To improve the protection time, monitoring solutions should alert about the ARP anomalies as fast as possible and at least one security engineer should be in charge of the internal incidents, analyzing MAC and IP addresses changes. To do so he will need a defined interface with the network administrators, to get information about network changes and upgrades. However, it is better to detect it late than later.

As has been described, the detection occurs at the system and network levels. The security perimeter protection is useless against ARP spoofing. To be able to detect this attack, it is necessary to gather information about the systems, the network and the actions executed by the users.

Once it has been confirmed that there is an incident, the main goal of the network security administrator will be to draw the traffic flow in the network, as accurately as possible.

To do so he has several sources of information:

- End hosts provide their ARP table snapshots.
- Switches provide their CAM snapshots.
- The audit trail mentioned in the "Preparation" phase provides the network traces containing the ARP packets that influenced the tables.
- The ARP modules warnings, in the form of syslog messages or windows pop-ups.

- If available, the alert messages generated by HIDS, "arpwatch", and NIDS, "snort", are providential.

Additionally, if a physical intrusion has taken place, the building cameras will help to determine the unauthorized person/s involved in the incident.

All this information must be considered as evidence, and chain of custody procedures must be applied to keep all the data safe from contamination. It is recommended to get all this information, sign it cryptographically, using MD5 hashes, and save it in a secure system created for this purpose and owned by a specific and defined individual. As mentioned in the next section, before containing the attacker all this information must be preserved.

The main aspect to identify this type of attack is being able to have fast and accurate procedures to correlate all the different data sources, so that it is possible to point out the offending MAC and IP addresses. During this process anomalies should be searched for, like the ones described in the "Signature of the attack" section, such as:

- the same MAC address in several switch ports (CAM)
- the same MAC for several IP addresses (ARP)
- host (not uplink or cascade) switch ports with multiple MAC addresses (CAM)
- incongruences between CAM and ARP tables (CAM and ARP)

To be able to differentiate valid scenarios from forged ones it is necessary to have a great knowledge of the network topology, being able to identify hosts with virtual network interfaces (more than one IP address associated to the same MAC address), fault tolerant solutions, where IP-MAC addresses changes are possible, host and uplink ports in the switches, and any other special cases.

This correlation process will help to list all evidence elements and could be improved by other factors, such as time synchronization, NTP, between all network devices.

Having a good knowledge of the ARP spoofing attack tools available will help the IH team to determine what the attacker's goal could be, such as hijack other user sessions, and the type of attacker:

- If he is using an easy to use standard tool, that usually generates lot of network noise he will probably be a script kiddie.
- If the pattern captured is not common he could even have developed his own ARP tool, or modified an existent one.

Containment

A way of mitigating the attack at the moment it is taking place is manually clearing the ARP cache tables of the target systems involved and setting static valid entries between these hosts.

Remember to save a copy of the ARP tables before clearing them to be able to research who the offending MAC address belongs to. Use the same policy for the CAM tables and any configuration changes made in any device. The containment phase will start modifying the systems.

At the switch level it is possible to isolate the systems affected by the attack in a separate VLAN already defined for this type of incidents. This is possible if the common scenario is presented, where there are two targets involved, client and server, and a system owned by the attacker.
If the attack affects the whole LAN, using ARP broadcasted crafted packets, this has no sense. If this is the case, then it will be necessary to review the logs and status, like ARP tables, of all neighboring systems.

The isolated VLAN can be specially prepared to monitor all traffic and get as much information as possible to be able to restore the compromised systems later on.

Another way of stopping the attack is disabling the attacker's port once it is identified. This would allow analyzing the life system and understanding the relevance of the incident, seeing how far the attacker has gone and what type of information he has got. This is only possible if we are fast enough, and the attacker doesn't have enough time to react and format its system.

Sending security personnel to the physical location where the network cable associated to the port ends won't be a bad idea. It is possible to find the attacker sitting in the working place or, if the attacker owns the system from a remote location, a complete forensic analysis over the compromised system should be performed.

It is possible to use a filter device that can stop traffic based on the MAC addresses to at least avoid other higher-level attacks. See "Filtering devices" section.

Other advanced solutions have the same effect, interrupting the communication between the attacker's host and the other devices, like Private VLANs or VACLs.

None of the methods explained generate network traffic. The only alerting event from the attacker perspective is the lost of network connectivity.

The incident handler "jump bag" in ARP spoofing cases, apart for having the standard elements to take notes and capture information, should include all the tools described along this paper: almost all of them are standard available tools in all OS, like the "arp" command, or the switches configuration commands. Additional networking equipment, like a 10/100 hub or cables (straight-through and cross-over) could be needed to be able to get network traces in switched environments without SPAN ports.

Although the usage of this attack has other high-risk implications, see "Advanced attacks based on ARP Spoofing" section, once the ARP spoofing

has been stopped all the other attacks are aborted. This doesn't mean that the attacker has not obtained critical information, like passwords, or that he has not opened other back doors into the compromised target systems to access them in the future using other methods. The standard policy for security incidents must be activated, such as changing system passwords or activating extra monitoring elements.

Therefore, it is recommended to make a backup of the target systems for later analysis and store it in a safe location. An additional step will be to consult the system owners about anomalies they find inside the hosts.

Eradication

Once the attack has been contained, there is no way to completely eliminate the ARP spoofing thread from the network, because it is inherent to the ARP dynamic and insecure design.

The first step will be to analyze the attacker's system, that contains the tools used to execute the attack. Probably the faster and safer solution from a functional perspective will be to rebuild this system from scratch once all the forensic analysis information has been extracted. If this is not possible, the most recent clean backup should be used.

If the attacker was an internal employee using his system, then general company policies must be applied. Penalties and punishments will be applied or even he could be dismissed. If the incident may lead to disciplinary actions or prosecution, systems warning banners advising about the privacy, the monitoring features and the possible penalties from unauthorized people will be required.

The ARP spoofing events should never be considered a minor thread; they are typically used intentionally to get or destroy secrets or proprietary information.

Then, the target systems must be also analyzed to determine the impact level: it is very possible that the attacker reached the highest privileges in the compromised system. The "cleanup" needed must be evaluated case by case, because there are no limits in the type of control the attacker is able to obtain through this methods.

Typically the root cause of the ARP spoofing attacks are the excess of trust on the internal users and the lack of monitoring and security controls.

It is recommended to periodically run an ARP spoofing assessment, simulating what an attacker would do and analyzing if all the countermeasures are effectively applied.

With the idea of detecting and even intimidating internal users about their illegal activities, the usage of an internal honeypot would be useful, at least in very critical network segments.

Recovery

There are no special recovery procedures to return the compromised systems to a well-known state related to ARP spoofing. The general recommendations should be applied to be sure the "new" systems are safe and clean from malicious software.

To improve the defenses against this attack, the network must be designed using protection features mentioned in Part 2: the 802.1x authentication protocol, IDS detectors to alert when the incident occurs, using filtering and access list controls between systems, adding encryption mechanism to mitigate the information that can be exposed, monitoring systems event and syslog messages, etc. These actions affect both, the end hosts and the network itself.

If systems have not being rebuilt, a deep and detailed monitoring should be put in place to analyze the ARP network traffic generated to and from them to really verify that the problem has been eliminated. To do so, the following solutions are recommended:

- Systems ARP module: syslog messages.
- HIDS, "arpwatch".
- NIDS, "snort".
- A complete audit trail containing all network traffic.

Lessons Learned

By carefully studying ARP we have learned how important is to have built-in authentication features at the protocol level. Due to this lack of authentication, any system can take the place of any other for evil purposes.

ARP is one of the oldest protocols in use today, and in the eighties, when it was designed, security was not a networking concern. New protocols will be needed, but if even new protocols were defined, their implementation is not an easy task, specially because that would require to upgrade all the deployed computing environments.

The idea that should be considered to secure the ARP protocol is the same one that was used when IPv6 was designed to secure the IP protocol. A new protocol, IPSec appeared, protecting from generic inherent TCP/IP vulnerabilities, such as IP spoofing, sniffing…

Therefore, ARP should evolve to ARPSec, an authenticated and encrypted protocol. This solution could have a huge overload in performance because ARP is a lightweight very frequently used protocol. Perhaps through the usage of longer timeouts, except for defined fail-over solutions the performance penalty could be solved. The 802.1x protocol in some way solves some of the ARP problems at the design level.

Besides, from the protocol design perspective it should be clearly defined the stateless or stateful nature of every protocol. In case of lack of a definition we have seen how each different implementation applies its own rules, which won't be the same for sure.

Another generic design concept that must be clearly explained is the validity of the information obtained when new traffic is received. Should the new information supersede older one? Are there some exceptions?

Besides, things must be tested twice due to the regular publication of bugs in nowadays software. As an example, we analyzed how dynamically learned ARP packets overwrite static ARP entries in some OS.

To learn as much as possible from a specific incident it is recommended to gather all the available information such as the tools used, operating systems involved, equipment used (laptops, desktop or servers)…

At the IHP level, it is important to review all the incident handling phases to improve them. At least the review should include:

- learn what detection mechanisms failed
- what tools used can be improved
- the profile and number of members in the IH team
- analyze if it is possible to define a faster and clearer procedure to get the information
- determine what ARP network events are security incidents in the environment affected
- improve the correlation of the data to isolate the affected systems and recover them as soon as possible

The ARP spoofing recovery process is simple and all the steps involved must be clearly defined, improving the IH capabilities for these cases.

Extras

Due to the extension of this paper an all the aspects it has tried to cover about the ARP spoofing attacks there has been several topics that have not being included or analyzed in detail.
This section presents the future work that would be performed in later versions of this paper or by "The SARP project" proposed in the "Abstract" section if it finally comes up.

Along the paper the future improvements have been referenced using the word "future", so a search using this term is recommended to be able to read all them.

To sum up, the main future research proposed are:

- Being able to create "The SARP project" and make it available through a web page in Internet.

- Covering additional operating systems, analyzing their ARP behaviours and their configurable parameters.
- Analyze the ARP table management in multihome systems.
- RFC1433, "Directed ARP".
- Cover channel proof of concept code based on ARP packets.
- Check the OS ARP "trailer" packet implementation, RFC 1122.
- Include packet with a "Sender MAC Address" of zero inside the Packet Taxonomy.
- Two additional cases reflected in the Packet Taxonomy design:
  - When the Ethernet "Target MAC Address" is broadcast or multicast, the ARP "Target MAC Address" could be zero.
  - Additional multicast addresses, not only 224.0.0.0.
- Analyze the ARP packets generated by all the available ARP spoofing tools.
- Analize the OS ARP behaviours when its network interface is in promiscuous mode.
- Test the HW and Protocol spaces fields when different from Ethernet and IP. Include this functionality in the "arpplet" tool.
- Analyze the timeout behaviour with all packets defined by the Packet Taxonomy.
- A complex environment, with DNS, default router and traffic crossing the lab network.
- Test systems when they are configured with DHCP.
- Analyze the ARP behaviour of the high availability and failover solutions most commonly used.
- How do the switches ARP tables learn?, and its relationship with the CAM tables.
- Check if the ARP timeout tests are influenced by UDP and TCP traffic instead of ICMP.
- Add new ARP timeout local and remote tests, described in the appropriate APPENDIX.
- Add new ARP timeout remote tests, 2, 3 and 4, where target system will learn through an ARP reply.
- Improve the ARP.pm module implementing the AUTO feature using the SSH protocol.
- Test Unix systems with its interface not listening ARP: ifconfig –arp.

List of References

[802.1X] IEEE. "802.1x - Port Based Network Access Control". URL: http://www.ieee802.org/1/pages/802.1x.html (1 Sep 2003)

[ABAD1] Christopher Abad (aempirei@ucla.edu). "Applications of ARP Analysis". 2002.

[ADVR1] Bert Hubert. "Linux Advanced Routing & Traffic Control HOWTO". Rev. 1.12. 2002.
URL: http://www.linux.org/docs/ldp/howto/Adv-Routing-HOWTO/index.html (19 Aug. 2003)

[ARPMAN1] Linux kernel 2.4 "arp (7)" man page.

[ARPSK1] "ARP-SK: a swiss knife tool for ARP". URL: http://www.arp-sk.org/ (15 Aug. 2003)

[ARPW1] "arpwatch tool". URL: ftp://ftp.ee.lbl.gov/arpwatch.tar.gz (1 May 2003)

[ATM1] "Approved specifications: LanE". The ATM Forum. URL: http://www.atmforum.com/standards/approved.html (17 Aug. 2003)

[BH2001] Jeff Nathan & Kevin Depeugh. "Layer 2 Attacks". BlackHat USA 2001.
URL:http://www.blackhat.com/html/bh-multi-media-archives.html#USA%202001
URL: http://jeff.wwti.com/blackhat-2001/blackhat_slides_v.9.ppt (1 Aug. 2003)

[CAIO1] Cisco. "Comparing Layer 2 Operations in CatOS and IOS on the Catalyst 6000/6500".
URL: http://www.cisco.com/warp/public/473/101.pdf (29 Aug. 2003)

[CATC1] Cisco. "Cisco Catalyst switches, Cat OS, Command References".
URL:http://www.cisco.com/en/US/products/hw/switches/ps679/prod_command_reference_list.html (14 Aug. 2003)

[CERT1] CERT. URL: http://www.cert.org (1 May 2003)

[CERT2] CERT. "Vulnerability Note VU#412115: Network device drivers reuse old frame buffer data to pad packets". 2003.
URL: http://www.kb.cert.org/vuls/id/412115 (23 May 2003)

[CLEA1] Cisco. "How to Clear a Single ARP Entry in a Router Using SNMP". Document ID: 13495.
URL:http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a0080094a9c.shtml (7 Jul. 2003)

[CIPRO1] Cisco Systems. "Proxy ARP". Document ID: 13718.
URL:http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094adb.shtml (10 Aug. 2003)

[CISCO-TCL] Cisco Systems, Inc. "Cisco IOS (12.3) Scripting with Tcl". 2003. URL:http://www.cisco.com/en/US/products/sw/iosswrel/ps5207/products_feature_guide09186a00801a75a7.html (14 Aug. 2003)

[CISPO1] Cisco. "Configuring Port Security in CatOS devices". URL:www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sw_7_3/confg_gd/sec_port.htm (28 Aug. 2003)

[CISPO2] Cisco. "Cisco Catalyst 6500 Series Switches. Configuring Port Security in IOS devices." URL:http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/12_1e/swconfig/port_sec.pdf (28 Aug. 2003)

[CISVU1] Cisco. "Cisco Security Advisory: Cisco IOS ARP Table Overwrite Vulnerability". URL: http://www.cisco.com/warp/public/707/IOS-arp-overwrite-vuln-pub.shtml (26 Jul. 2003)

[CSI2002] CSI/FBI. "2002 Computer Crime and Security Survey". 2002. URL: http://www.gocsi.com/press/20020407.jhtml (17 Aug. 2003)

[CSI2003] CSI/FBI. "2003 Computer Crime and Security Survey". 2003. URL: http://www.gocsi.com/press/20030528.jhtml (17 Aug. 2003)

[DATA1] DataWizard. "Altering ARP Tables (version 1.00)". 2001. The Netherlands. URL: http://packetstorm.icx.fr/papers/general/Altering_ARP_Tables_v_1.00.htm (14 Jul. 2003)

[DSNIFF1] "Dsniff attacking tools suite". URL: http://www.monkey.org/~dugsong/dsniff/ (1 May 2003)

[ETH1] TechFest. "TechFest Ethernet Technical Summary". URL: http://www.techfest.com/networking/lan/ethernet2.htm (17 Aug. 2003)

[ETH2] "Ethernet II frames". Firewall.cx. URL:http://www.firewall.cx/ethernet-frames-ethernet-ii.php (1 Apr. 2003)

[ETHR1] Ethereal. URL: http://www.ethereal.org (1 May. 2003)

[ETHS1] Computer Networks and Internet. "Q & A on Minimum Frame Size for Ethernet". URL: http://www.netbook.cs.purdue.edu/othrpags/qanda85.htm (18 Aug. 2003)

[FIRE1] "Introduction to multicast". Firewall.cx. URL: http://www.firewall.cx/index.php?c=multicast-intro (18 Jun. 2003)

[FLAK1] Grzegorz Flak. "Hole in ARP module in Windows NT/2000". Neohapsis.

URL: http://archives.neohapsis.com/archives/vuln-dev/2001-q4/0541.html (17 Aug. 2003)

[FRAG1] Dug Song. "Fragrouter tool".
URL: http://www.securityfocus.com/tools/176 (17 Aug. 2003)

[GILLI1] RouterGod Celebrity Guest Interview. ;-). "Gillian Anderson on Lan Switching Part 1". URL: http://routergod.com/gilliananderson/ (21 Aug. 2003)

[GRAT1] "Gratuitous ARP packet decoded".
URL: http://roads.lut.ac.uk/Linux2001/talk/slide11.html (14 Aug. 2003)

[HACK1] Sean Convery. "Hacking Layer 2: Fun with Athernet switches". Cisco at Black Hat.
URL:http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-convery-switches.pdf (7 Jul. 2003)

[HPING1] "HPING tool". URL: http://www.hping.org/ (17 Aug. 2003)

[HPP1] Hewlett-Packard Company. "Tech Brief: A Primer on HP Probe". July 1993. URL: http://www.hp.com/rnd/support/manuals/pdf/hp_probe.pdf (8 May 2003)

[HSRP1] Cisco Systems. "Load sharing with HSRP". Document ID: 13781.
URL:http://www.cisco.com/en/US/tech/tk648/tk362/technologies_configuration_example09186a0080094e90.shtml (13 Aug. 2003)

[HYPE1] "ARP". HyperDictionary.
URL: http://www.hyperdictionary.com/dictionary/Address+Resolution+Protocol (17 Aug. 2003)

[IANA] IANA. "Ethernet numbers". 2001-05-01.
URL: http://www.iana.org/assignments/ethernet-numbers (18 Jun. 2003)

[IANA2] IANA. "Multicast addresses".
URL: http://www.iana.org/assignments/multicast-addresses (18 Jun. 2003)

[IEEE1] IEEE OUI and Company_id assignments.
URL: http://standards.ieee.org/regauth/oui/index.shtml (9 Jun. 2003)

[IETF1] Internet Engineering Task Force. URL: http://www.ietf.org/ (7 Jun. 2003)

[IOS123] Cisco Systems. "IOS 12.3 Command Reference".
URL:http://www.cisco.com/en/US/products/sw/iosswrel/ps5187/prod_command_reference_list.html (1 Aug. 2003)

[IOSC1] Cisco "IOS CAM Command Reference".
URL:http://www.cisco.com/en/US/products/hw/switches/ps607/products_command_reference_chapter09186a008007e90a.html (14 Aug. 2003)

[IPROU1] "iproute2" suite. URL: ftp://ftp.inr.ac.ru/ip-routing/ (1 May. 2003)

[KEVIN1] Kevin D. Mitnick & William L. Simon. "The Art of Deception. Controlling the human element of security". Wiley Publishing, Inc. 2002. ISBN: 0-471-23712-4

[LEAP1] Cisco. "WLANs: Dictionary Attack on Cisco LEAP". Document ID: 44281. URL: http://www.netstumbler.com/article.php?sid=731
URL:http://www.cisco.com/en/US/tech/tk722/tk809/technologies_tech_note09186a00801aa80f.shtml (25 Aug. 2003)

[LVS1] "ARP problem in LVS/TUN and LVS/DR". LinuxVirtualServer.
URL: http://www.linuxvirtualserver.org/docs/arp.html (10 Aug. 2003)

[LVS2] "LinuxVirtualServer project". URL: http://www.linuxvirtualserver.org/ (8 Jun. 2003)

[LX802] "Linux 802.1x Open Source implementation".
URL: http://www.open1x.org (5 Aug. 2003)

[LXPA1] Linux kernel patch to protect against ARP spoofing: arp-antidote.
URL:
http://hypermail.idiosynkrasia.net/linux-kernel/archived/2002/week50/0009.html
URL: http://securitylab.ru/_tools/antidote2.diff.gz (23 Jun. 2003)

[MART1] Martin A. Brown. "Guide to IP Layer Network Administration with Linux". Version 0.4.4. "Chapter 2.1. ARP".
URL: http://linux-ip.net/html/ether-arp.html (4 Aug. 2003)

[MEET1] Meeting House 802.1x implementations.
URL: http://www.mtghouse.com (5 Aug. 2003)

[MENTA1] Mentat. "TCP/IP protocol stack".
URL: http://www.mentat.com/tcp/tcpdata.html (14 Jun. 2003)

[MITRE1] "Common Vulnerabilities and Exposures".
URL: http://www.cve.mitre.org/ (1 May 2003)

[MS-KB1] Microsoft Knowledge Base. "Article 219374: How to Disable the Gratuitous ARP Function".
URL:http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/articles/Q219/3/74.ASP&NoWebContent=1 (14 Aug. 2003)

[MS-KB2] Microsoft Knowledge Base. "Article 199773: Behaviour of Gratuitous ARP in Windows NT 4.0".
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;199773 (17 Aug. 2003)

[MS-KB3] Microsoft Knowledge Base. "Article 244331: MAC Address Changes for Virtual Server During a Failover with Clustering".

URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;244331 (17 Aug. 2003)

[MS-KB4] Microsoft Knowledge Base. "Article 124797: ARP Static Cache Entries Switch to Dynamic".
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;124797 (17 Aug. 2003)

[MS-KB5] Microsoft Knowledge Base. "Article 280524: Disabling the Gratuitous ARP Functionality in Windows 2000".
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;280524 (17 Aug. 2003)

[MS-KB6] Microsoft Knowledge Base. "Article 166750: ARP Cache Entries May Not Time Out for 10 Minutes".
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;166750 (17 Aug. 2003)

[NDD1] Sun Microsystems. "nddconfig script v.1.9". 2001.
URL: http://www.sun.com/solutions/blueprints/tools/ (20 Jun. 2003)

[NNM1] HP & Cisco Systems. "When Running HSRP, "Duplicate IP Address" Messages Appear in HP OpenView NNM Event Browser". Document ID: 13432.
URL:http://www.cisco.com/en/US/tech/tk648/tk362/technologies_configuration_example09186a0080094b25.shtml (19 Aug. 2003)

[OBS1] "13.2 Obscure settings. Linux Advanced Routing & Traffic Control HOWTO". URL:http://ldp.kernelnotes.de/HOWTO/Adv-Routing-HOWTO/lartc.kernel.obscure.html (20 Aug. 2003)

[OFIR1] Ofir Arkin. "ICMP Usage In Scanning – The Complete Know How".(Version 3.0). June 2001. Sys-Security Group.
URL: http://www.sys-security.com/html/projects/icmp.html,
URL: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf (1 Aug. 2003)

[OFIR2] Ofir Arkin and Josh Anderson. "EtherLeak: Ethernet frame padding information leakage". @stake. 2003.
URL:http://www.atstake.com/research/advisories/2003/atstake_etherleak_report.pdf (23 May 2003)

[PATTON1] Patton, Michael A. "Vendor Codes". 1999/03/09.
URL: http://www.cavebear.com/CaveBear/Ethernet/vendor.html (18 Jun. 2003)

[PREL1] Prelude-IDS. URL: http://www.prelude-ids.org (19 Jul. 2003)

[PVLA1] Cisco. "Securing networks with Private VLANs and VLAN Access Control Lists". Document ID: 10601.
URL: http://www.cisco.com/warp/public/473/90.shtml (3 Aug. 2003)

[PWER1] "Linux Powerteak Project". URL: http://powertweak.sourceforge.net (1 Aug. 2003)

[RFC] Request For Comments. URL: http://www.rfc-editor.org/ (4 Jun. 2003)

[RFC826] Plummer, David C. "RFC 826: An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware". Network Working Group. November 1982. URL: ftp://ftp.rfc-editor.org/in-notes/rfc826.txt (11 Jun. 2003)

[RFC893] Samuel J. Leffler, Michael J. Karels. "RFC 893: Tralier Encapsulations". Network Working Group. April 1984.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc893.txt (29 Jul. 2003)

[RFC894] Charles Hornig. "RFC 894: A Standard for the Transmission of IP Datagrams over Ethernet Networks". Network Working Group. April 1984.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc894.txt (29 Jul. 2003)

[RFC1027] Smoot Carl-Mitchell. "RFC 1027: Using ARP to Implement Transparent Subnet Gateways". Network Working Group. Texas Internet Consulting. October 1987. URL: ftp://ftp.rfc-editor.org/in-notes/rfc1027.txt (21 Jun. 2003)

[RFC1042] J.Postel, J. Reynolds. "RFC 1042: A Standard for the Transmission of IP Datagrams over IEEE 802 Networks". Network Working Group. February 1988. URL: ftp://ftp.rfc-editor.org/in-notes/rfc1042.txt (29 Jul. 2003)

[RFC1112] S. Deering. "RFC 1112: Host Extensions for IP Multicasting". Network Working Group. August 1989.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1112.txt (10 Jun. 2003)

[RFC1122] R. Braden. "RFC 1122: Requirements for Internet Hosts -- Communication Layers". Network Working Group. October 1989.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1122.txt (29 Jul. 2003)

[RFC1433] J. Garrett. "RFC 1433: Directed ARP". Network Working Group. AT&T Bell Laboratories. March 1993.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1433.txt (10 Jun. 2003)

[RFC1700] Reynolds, J., Postel, J. "RFC 1700: Assigned numbers". Network Working Group. October 1994. URL: ftp://ftp.rfc-editor.org/in-notes/rfc1700.txt (18 Jun. 2003). [See also RFC 3232]

[RFC1812] F. Baker, Editor. "RFC 1812: Requirements for IP Version 4 Routers". Network Working Group. Cisco Systems. June 1995.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1812.txt (29 Jul. 2003)

[RFC1868] G. Malkin. "RFC 1868: ARP Extension – UNARP". Network Working Group. Xylogics, Inc. November 1995.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1868.txt (20 Jun. 2003)

[RFC2225] M. Laubach. "RFC 2225: Classical IP and ARP over ATM". Network Working Group. Com21, Inc. April 1998.
URL: ftp://ftp.rfc-editor.org/in-notes/rfc2225.txt (25 Jul. 2003)

[ROB1] Robert Wagner. "Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks". Practical Assignment GSEC Version 1.2f (amended August 13, 2001). URL: http://www.sans.org/rr/threats/address.php (10 May 2003)

[SANS1] Search by the "arp" term in Sans Reading Room.
URL: http://www.sans.org/rr/ (17 May. 2003)

[SEAN1] Sean Whalen (arpspoof@gmx.net). "An Introduction to Arp Spoofing". Revision 1.8. April, 2001. URL: http://chocobospore.org/arpspoof (1 Aug. 2003)
URL:http://www.rootsecure.net/content/downloads/pdf_downloads/arp_spoofing_intro.pdf (6 Jun. 2003)

[SEC2005] Ian Foo. "Deploying 802.1x Identity Services for LAN Security".
URL:http://www.cisco.com/networkers/nw03/presos/general_abstracts.html#SEC2005
URL: http://www.cisco.com/networkers/nw03/presos/docs/SEC-2005.pdf (27 Aug. 2003)

[SIL1] Siles, Raúl. "Análisis de seguridad de la familia de protocolos TCP/IP y sus servicios asociados". Edición I. June 2002.
URL: http://rain.prohosting.com/rsiles/ (17 May 2003)

[SMART1] Althes. "The IP Smart spoofing". 2002.
URL: http://www.althes.fr/ressources/avis/smartspoofing.htm (17 Jul. 2003)

[SMART2] Althes. "Arp-fillup tool". 2002.
URL: http://www.althes.fr/ressources/tools/arp-fillup/README (17 Jul. 2003)

[SNIFF1] Robert Graham. "Sniffing (network wiretap, sniffer) FAQ".
URL: http://www.robertgraham.com/pubs/sniffing-faq.html (14 Apr. 2003)

[SNO1] Snort NIDS: http://www.snort.org (1 May. 2003)

[SOLA1] Ido Dubrawsky. "Solaris Kernel Tuning for Security". December, 2000.
URL: http://www.securityfocus.com/infocus/1385 (19 May 2003)

[SSHA1] stealth <stealth@segfault.net> "It cuts like a knife. SSHarp.". Phrack.
URL: http://www.phrack.org/show.php?p=59&a=11
URL: http://stealth.7350.org/7350ssharp.tgz
URL: http://www.shellcode.com.ar/docz/asm/ssharp.pdf (17 Jun. 2003)

[SSP1] "Sniffable Switch Project". URL: http://www.alaricsecurity.com/ssp.html (1 Jun 2003)

[STEV1] Stevens, W. Richard. "TCP/IP Illustrated, Volume 1. The Protocols". Addison Wesley Longman, Inc, 1994. ISBN: 0201633469.

[STEV2] Stevens, W. Richard, Wright, Gary R.. "TCP/IP Illustrated, Volume 2. The Implementation". Addison Wesley Longman, Inc, 1995. ISBN: 020163354X.

[SUNS1] Alex Noordergraaf and KeithWatson. "Solaris Operating Environment Network Settings for Security". Global Enterprise Security Service. Sun BluePrints OnLine - December 1999.
URL: http://www.sun.com/solutions/blueprints/1299/network.pdf (20 Jun. 2003)

[TARA1] Jonathan Wilkins. "Taranis: invisible traffic redirection on Ethernet switches". Phrack.
URL: http://www.bitland.net/taranis/
URL: http://www.phrack.org/phrack/57/p57-0x06 (30 Jul. 2003)

[VISV1] Mahesh Visvanathan and Ramya Ramakrishnan. "Seminar on ARP Spoofing". Albert-Ludwigs- University of Freiburg, Department of Computer Science, Internetworking.
URL: http://www.ks.uni-freiburg.de/inetwork/papers/ARP-spoofing-handout.pdf (14 Jul. 2003)

[W2000] Dave MacDonald and Warren Barkley. "MS Windows 2000 TCP/IP Implementation Details".
URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/network/deploy/depovg/tcpip2k.asp (5 Jun. 2003)

[WIKI1] "ARP protocol". Wikipedia.
URL: http://www.wikipedia.org/wiki/Address_resolution_protocol (17 Aug. 2003)

[WINARP1] Winarpwatch: Windows porting of the arpwatch tool available for Unix environments. URL: http://www.arp-sk.org/files/related/warpwatch.zip (8 Aug. 2003)

[WINNT1] Dave MCDonald. "MS Windows NT Server. TCP/IP Implementation Details, Version 2.0". MS Enterprise Technical Support and the Personal and Business Systems Division. 1996.
URL:
https://www.microsoft.com/ntserver/techresources/commnet/TCPIP/tcpip.asp
URL: https://www.microsoft.com/ntserver/docs/TCPIP.DOC (5 Jun. 2003)

[WIRE1] Bob Fleck (rfleck@cigital.com) and Jordan Dimov (jdimov@cigital.com). "Wireless Access Points and ARP Poisoning: Wireless vulnerabilities that expose the wired network". Cigital, Inc.
URL: http://www.ljudmila.org/matej/arppoison.pdf (1 Jun. 2003)

[YURI1] Yuri Volobuev. "Redir games with ARP and ICMP".
URL: http://lists.insecure.org/lists/bugtraq/1997/Sep/0059.html (17 May. 2003)

[YURI2] Response from Neil J Long to [YURI1].
URL: http://lists.insecure.org/lists/bugtraq/1997/Sep/0070.html (17 May. 2003)

APPENDIX I: Operating Systems researched

This section contains the details about the operating systems, versions and patch levels used in this paper's research:

Target Systems
Cisco switch WS-C2924-XL: IOS 11.2(8)SA5
Cisco router 2612: IOS 12.0(2)XC2
HP-UX 10.20 9000/712 – "Workstation ACE for HP-UX 10.20 (April 1998)"
HP-UX 11.00 - B.11.00.47.08 General Release Patches, November 1999 (ACE)
HP-UX 11i - B.11.11.0102.2 (February 2001)
Linux: kernel 2.4 – Red Hat 8.0 (kernel version 2.4.18-14)
Windows 2000 Professional SP 3 (5.00.2195)
Windows 2000 Professional (without SP) (5.00.2195)
Windows NT Server 4.0 SP6 or SP6a (Build 4.00.1381)
Solaris 8, "SunOS 5.8 Generic_108528-13": "February 2000" and "Patch October 2001"
Network Devices
Hewlett Packard AdvancedStack Hub-8E (J3128A)
Hewlett Packard Procurve 10BT Hub 24M (J3303A): ROM A.01.00, EEPROM A.02.02, HW A.01.00.
Cisco switch WS-C2924-XL: IOS 11.2(8)SA5
Attacker's system
Linux: kernel 2.4 – Red Hat 7.3 (kernel version 2.4.18-19.7)

APPENDIX II: Research lab description

To develop all the test sets defined under the different "ARP protocol security research" sections, the following lab environment is proposed:

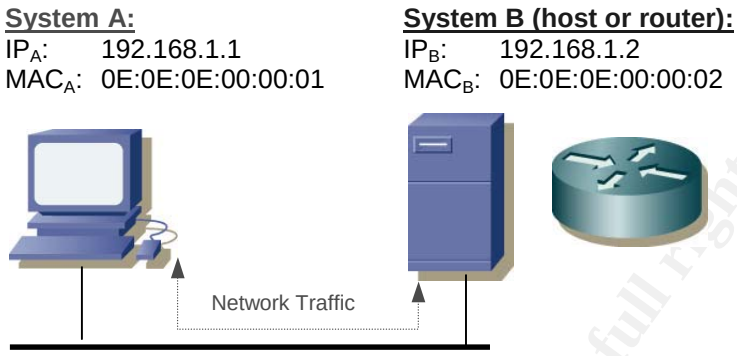


Figure II.1.Research lab network diagram

This network environment is based in the simplest LAN network possible (except when just using a crossover cable, ;-)). It is based on a simple Ethernet hub to interchange network traffic between systems.
The idea behind this simple design is disturbing the tests as less as possible through the existence of additional spurious traffic. For example, switches generate lots of frames belonging to different layer 2 protocols: STP (Spanning Tree Protocol), CDP (Cisco Discovery Protocol; used not only by Cisco equipment but by other vendors, like HP), VTP (VLAN Trunking protocol) and other protocols.

"System A" is the "analyzer" system, also referred as the attacker's system in some tests, used to generate traffic (and capture it at the same time) to stimulate "System B", the system to be analyzed or "target" system. Network traces will be taken in both systems to complement the different ARP analysis and to check the "Unicast Poll" method implemented.

This environment has been used for the following test sets:

- "ARP timeouts tests" section.
- "ARP packet taxonomy tests" section.
- "Bootstrap and shutdown times research" section.
- "ARP big anomalies tests" section.

The complexity of the lab could be incremented using other network devices, as switches, and adding additional equipment that will host some of the monitoring and complementary tools described bellow.

The following lab options are recommended to get as much benefit as possible from the different research tests proposed along this paper:

1. Install ARP network detection software to be able to test how every test is seen from a defensive point of view. The software recommended for this purpose are:
    - Snort ARP preproccessor [SNO1]
    - Arpwatch [ARPW1]
2. Configure the attacking system, a Linux box is recommended due to the freely available security tools, to allow the usage of ARP spoofing attacks. An example "pre_arp_spoofing.sh" script has been provided in section "ARP spoofing preparation script".
3. Network traces must be taken through sniffing tools in every operating system involved [SNIFF1]. This will allow analyzing all the network traffic crossing the LAN and understand how ARP works.
   If a switch is used as the LAN network device, an SPAN port must be configured to get all the traffic associated to all the switch ports. If VLAN are used this port should allow monitoring all defined VLANs.
4. Monitor the main logging subsystem of every operating system analyzed, that is, the syslog in the Unix variants, the "Event viewer" in the Windows ones, and the console, internal buffer or syslog in the Cisco systems.
   The logging capabilities will be in charge of generating messages about ARP anomalies, as the typical "duplicate address" notification.
5. Time synchronization (through the NTP protocol) will be needed to correlate all the different information sources collected: network traces, ARP table contents (output is timestamped through the ARP table status scripts, see "ARP table status scripts" section) and the timestamped output generated by the scripts used to create traffic (see "APPENDIX III: ARP timeouts research" section).
   As a recommendation it will be better to synchronize the systems clock initially and then stop the NTP client processes, to avoid traffic disruption and anomalies in the tests due to the UDP packets used by the NTP protocol. The initial synchronization will maintain clocks synchronized for some hours under normal circumstances.
6. All the tests run for this analysis should consider that all Internet communications occurs using IP addresses directly, not needing the translation between hostnames and IP addresses associated to the DNS service. If not, the DNS resolution traffic, ARP and UDP packets, to and from the DNS servers could influence the results.

APPENDIX III: ARP timeouts research

This section describes the ARP timeout tests in detail.

Local tests: [TestTLn]

By local we mean the "target" system, "System B".

The actions that originate the traffic are mainly OS commands, such as "ping", which are platform dependent. That's the reason why they are not integrated in the automatized scripts.

TestTL1: Timeout associated to an ARP reply solicited packet. [T1b]
`System_B$ ping –c 1 "System_A" (1) (2)`
It should generate an ARP request broadcast packet, because "System B" shouldn't have a previous entry in the ARP table associated to "System A", a reply should be generated by "System A" that should populate B's ARP table.

(1) All IP traffic will be generated by ICMP requests through the "ping" command, widely available in all OS. Although the IP traffic influence in ARP timeouts should be the same independently of the upper layer protocol, in the future these tests could be rerun by using UDP and TCP traffic, in order to observe if there are differences. Simple TCP and UDP traffic can be generated using "hping2" [HPING1] (only available for the Unix platform or a similar tool in Windows). In case of Cisco IOS and CatOS other TCP client tools (telnet or ftp) and UDP tools (traceroute, tftp or name resolution) will be required.

(2) Local ping command implementation is platform dependent, so the goal of this command execution is to send only one ICMP packet: HP-UX uses option "-n 1", as Windows does, while Linux uses the "-c 1" option (the one included in the example) and Cisco IOS should use the extended ping command to indicate how many packets are going to be send.

It is recommended to delete the ARP tables of both systems before running a new test, not being influenced by previous results.

TestTL2 and TTL3: ARP entry reusing influence in timeout.

TestTL2 - Learned by ARP request: `System_A$ ping –c 1 "System_B"` [T1ra]
TestTL3 - Learned by ARP reply: `System_B$ ping –c 1 "System_A"` [T1rb]
 	
`System_B$ sleep 60`
`System_B$ ping –c 1 "System_A"`

This is a similar test to TestTL1, but which reuses the learned ARP entry. The second "ping" command will get the MAC address from the ARP table. It is there because the first "ping" command helped to learn it. Check if the TestTL1 timeout is increased in "n" seconds, being "n" the "sleep" command parameter value, such as, 60 (could be modified based on OS timeout values, but should be lower than the ARP entry expiration timeout).

If timeout varies, the timeout algorithm is influenced by the reusing of the entry when generating outbound traffic, if not, it is using a fixed timeout [T1-fixed].

Additional local test could be added in the future. It could be interesting to analyze:

- Timeout for incomplete ARP entries: generating traffic (ping) from the local system to a non existent IP address.
- ARP entry reusing influence in timeout for incomplete ARP entries: same as TestTL2 and 3 but for incomplete entries.
- ARP entry reusing influence in timeout for continuous traffic: if a continuous traffic flow is generated (ping), check if the associated entry never disappears from the ARP table.
- Same test for non existent destination system, to check if the incomplete entry never disappears.

Remote tests: [TestTRn]

The term "remote" refers to the "target" system, "System B", from the point of view of the "analyzer's" system, "System A". We will analyze the inbound traffic coming to "target" remotely generated by "analyzer".

See "ARP timeouts script" section for this tests related scripts.

TestTR1: Timeout associated to a received ARP request packet. [T1a]
`System_A$ ping -c 1 "System_B"`
"Analyzer" system, "System A", should generate an ARP request broadcast packet when running this action if its local table didn't have a previous entry in the ARP table associated to "System B". The goal is letting this request populate "System B" ARP table. Also a reply should be generated by "System B" that should populate A's ARP table.

TestTR2: Generic traffic reception influence in timeout. [T3b]

`System_A$ ping -c 1 "System_B"`
`System_A$ sleep 60`
`System_A$ ping -c 1 "System_B"`

This is a similar test to TestTR1, but by resending traffic to test if destination system learns and resets the ARP entry timeout when generic IP traffic (ICMP, TCP, UDP…) is received for the same IP address. If so, information is learned from the Ethernet header, because the new packets are not ARP packets ("System A" will reuse the previous learned ARP entry (from first "ping" command), and won't generate additional ARP traffic). Check if the TestTR1 timeout is increased in "n" seconds, being "n" the "sleep" command parameter value, such as, 60 (could be modified based on OS timeout values, but should be lower than the entry expiration timeout). If so, the timeout algorithm is influenced by the reception of inbound traffic. *NOTE:* Due to the fact that a Linux box is used as the "analyzer" system "n" must be less than 30 seconds for an ARP request not to be generated, when an entry leaves the "reachable" state.

TestTR3: ARP request traffic reception influence in timeout. [T3aa]

`System_A$ ping –c 1 "System_B"`
`System_A$ arp –d "System_B"`
`System_A$ sleep 60`
`System_A$ ping –c 1 "System_B"`

This is a test similar to TestTR2, but which evaluates how a new ARP request influences the destination timeout. The second "ping" command will need an additional ARP resolution because the associated entry has been deleted. If timeout is not influenced it will probably use the "Unicast Poll" method [T3-poll] to renew ARP entries or a method where entries just directly expire. *NOTE:* Due to the fact that a Linux box is used as the "analyzer" system if "n" is greater than 30 seconds an ARP request will be generated, so the entry deletion step it's not necessary.

TestTR4: ARP reply traffic reception influence in timeout. [T3ab]

`System_A$ ping –c 1 "System_B"`
`System_A$ sleep 60`
`System_A$ arpplet … (ARP standard reply)`

The goal of this test is to check if receiving an additional ARP reply packet influences the timeout. Due to the fact that the "System B" ARP entry has not yet expired, it has not asked for this reply, so it is an ARP unsolicited packet. It must be crafted (using "arpplet" tools), looking similar to the ARP reply in TestTL1. If the timeout is not influenced it will probably use the "Unicast Poll" method [T3-poll] to renew ARP entries or a method where entries just directly expire. *NOTE:* For Cisco tests "n" must be greater than 2 to be able to visualize if the timer is reset through the "show arp" command. It shows time in minutes.

This situation cannot occur in "normal" network stack behaviour because an ARP reply solicited message should arrive only when "target" system asks for it. Due to the fact that "target" system ARP entry has not expired, entry is still present in the ARP table, so it will never send an ARP request packet. The only case in which this could be seen is when the OS uses the "Unicast Poll" method and asks for an still valid ARP entry (then a request/reply interchange will be seen). In this test, the polling doesn't exist so the reply has been crafted.

Last 3 tests, TestTR2 to TestTR4, force target system to learn the ARP entry through an ARP request packet associated to the "ping" command. In a future research these tests should be also developed to force target system to learn through an ARP reply packet.

Next set of 4 tests uses crafted packets because the main goal is to study the timeout behaviour when two flows of traffic are crossing the network with a different "MAC address" each.

TestTR5: ARP blocked entry timeout based on learning through ARP request and changing through ARP request [T2aa]
`System_A$ ping –c 1 "System_B"` (same as TestTR1) `System_A$ arpplet … (ARP standard request with a different MAC)`
This test sends an ARP request packet, forcing "target" system to learn a new MAC

address, and then it sends a crafted ARP request packet with a different MAC address to check if this second packet changes the ARP table entry previously created. If not, it continues sending this packet until it changes, to be able to know the blocking timeout value.

TestTR6: ARP blocked entry timeout based on learning through ARP request and changing through ARP reply [T2ab]
`System_A$ ping -c 1 "System_B"` (same as TestTR1) `System_A$ arpplet … (ARP standard reply with a different MAC)`
This test sends an ARP request packet, forcing "target" system to learn a new MAC address and then it sends a crafted ARP reply packet with a different MAC address to see if this second packet changes the ARP table entry previously created. If not, it continues sending this packet until it changes, to be able to know the blocking timeout value.

TestTR7: ARP blocked entry timeout based on learning through ARP reply and changing through ARP request [T2ba]
`System_B$ ping -c 1 "System_A"` (same as TestTL1) `System_A$ arpplet … (ARP standard request with a different MAC)`
This test sends an ARP reply packet, forcing "target" system to learn a new MAC address and then it sends a crafted ARP request packet with a different MAC address to see if this second packet changes the ARP table entry previously created. If not, it continues sending this packet until it changes, to be able to know the blocking timeout value.

TestTR8: ARP blocked entry timeout based on learning through ARP reply and changing through ARP reply [T2bb]
`System_B$ ping -c 1 "System_A"` (same as TestTL1) `System_A$ arpplet … (ARP standard reply with a different MAC)`
This test sends an ARP reply packet, forcing "target" system to learn a new MAC address and then it sends a crafted ARP reply packet with a different MAC address to see if this second packet changes the ARP table entry previously created. If not, it continues sending this packet until it changes, to be able to know the blocking timeout value.

Additional remote test could be added in the future. It could be interesting to analyze:

- Same tests as TestTR1 and 3 but using an ARP gratuitous packet.

APPENDIX IV: ARP spoofing research scripts

This section contains all the different scripts referenced along this paper.

ARP spoofing preparation script

This Linux shell script allows preparing a Linux box to carry on ARP spoofing attacks without disrupting network traffic, allowing packet forwarding, and avoiding other network noise.

Script name: "pre_arp_spoofing.sh"

```
#!/bin/sh
echo
echo "Preparation for ARP spoofing script"
echo "Linux kernel 2.4. Version: 0.9.0"
echo
echo "BEGIN"
echo

# Activating forwarding in the kernel:
echo "* Activating IP forwarding at the kernel level..."
echo 1 > /proc/sys/net/ipv4/ip_forward
echo

# Deactivating IP ICMP redirects for ALL interfaces
echo "* Deactivating ICMP redirects at the kernel level for: "

echo "  all"
echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
echo 0 > /proc/sys/net/ipv4/conf/all/secure_redirects

echo "  default"
echo 0 > /proc/sys/net/ipv4/conf/default/secure_redirects
echo 0 > /proc/sys/net/ipv4/conf/default/send_redirects

echo "  eth0"
echo 0 > /proc/sys/net/ipv4/conf/eth0/secure_redirects
echo 0 > /proc/sys/net/ipv4/conf/eth0/send_redirects
echo

# Checking IPCHAINS or IPTABLES
echo "* Checking ipchains..."
iptables --list | grep FORWARD
echo

echo "* Checking iptables..."
ipchains --list | grep FORWARD
echo

# Suggest setting interface not to manage ARP traffic
echo "* Checking network interface ARP management..."
# All interfaces: ifconfig -a
```

```
ifconfig eth0

# "ifconfig eth0 -arp":
# eth0      Link encap:Ethernet  HWaddr 00:10:A4:ED:97:97
#           UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
#                        ^^^^^^^

echo "END"
```

ARP table status scripts

These scripts will allow monitoring the OS ARP table along the time. They show the ARP table contents and a timestamp to be able to measure the ARP timeouts. They are platform dependent so different implementations were needed to cover different operating systems.

The scripts allow setting a timer value to define when an ARP table snapshot is taken, argument 1, and through argument 2 it is possible to configure a filtering mechanism to select which IP address entry will be displayed.

Unix usage example, taking table status snapshots every 5 seconds for the IP address 192.168.1.200:

```
$ arp_table.sh 5 192.168.1.200 | tee "/tmp/arp_status.log"
```

In the Windows version the "seconds" parameter specifies seconds - 1.

Both, Solaris and Windows OS don't support the "arp –n" option, while other Unixes do. So, three different scripts were created to continuously poll the ARP table.

Cisco IOS

Cisco devices don't have a scripting language until IOS version 12.3 [CISCO-TCL], the latest one, to periodically get the ARP table entries. This is not very important in order to measure its timeouts because the IOS default timeout value is very long, 4 hours.

Unix: HP-UX and Linux

Script name: "arp_table.sh"

```
#!/bin/sh
echo
echo \"Unix ARP table \& timer utility v.0.1\"
echo \"by Raul Siles \(2003\)\"
echo

# Argument $1 is the number of seconds to sleep.
# If it is not defined it sleeps 1 second.
```

```
if [ "$1" = "" ]
then
TIMER=1
else
TIMER=$1
fi

while true
do
echo
echo Unix ARP table:
echo ----------------
echo

date

# Argument $2 is the IP address we are interested in. If not defined
the whole table is displayed.
echo
arp -an | grep "$2)"

echo
echo Sleeping about $TIMER seconds...

sleep $TIMER

done
```

Windows

Script name: "arp_table.bat"

```
@echo off
REM Windows batch file
REM Name and args: %0, %1, %2 ...

echo.
echo "Windows ARP table & timer utility v.0.1"
echo "by Raul Siles (2003)"
echo.

REM ARGUMENT %1 is the number of seconds - 1 to sleep.
REM If not defined it sleeps 1 second.
if "%1"=="" (set TIMER=2) else (set TIMER=%1)

:SHOW
echo.
echo Windows ARP table:
echo -------------------
echo.


REM Argument %2 MAY allow hostname selection if set:
echo. | time
arp -a %2

echo.
echo Sleeping about %TIMER% seconds...
```

```
REM http://www.jsiinc.com/SUBJ/tip4600/rh4630.htm
REM Example.- sleep 5 seconds ---> "-n 6" = 5+1
@ping -n %TIMER% 127.0.0.1>nul

goto SHOW
```

Solaris

Script name: "sol_arp_table.sh"

```
#!/bin/sh
echo
echo \"Unix ARP table \& timer utility v.0.1\"
echo \"by Raul Siles \(2003\)\"
echo

# Argument $1 is the number of seconds to sleep.
# If it is not defined it sleeps 1 second.
if [ "$1" = "" ]
then
TIMER=1
else
TIMER=$1
fi

while true
do
echo
echo Unix ARP table:
echo ----------------
echo

date

# Argument $2 is the IP address we are interested in. If not defined
the whole table is displayed.
echo
arp -a | grep "$2 "

echo
echo Sleeping about $TIMER seconds...

sleep $TIMER

done
```

ARP timeouts scripts

A script called "arp_timeouts.pl" has been developed based on the generic "ARP.pm" module. See the "ARP.pm" perl module described in next section.

This script implements the ARP timeout remote tests, from TestTR1 to TestTR8, described in the "APPENDIX III: ARP timeouts research" section. It

mainly sets the variables and parameters used by the timeout functions included in the "ARP.pm" module, and invokes the desired functions.

ARP packet taxonomy scripts

A perl module, called "ARP.pm", was created to contain all the functions and variables needed to develop the researches described along this document. It integrates functions needed for tests of different nature: timeouts, packet taxonomy…

It is recommended to inspect its source code to visualize all the features included, but from a very general perspective, this module contains:

- all the variables and constants that can affect ARP packets and timeouts
- logging capabilities to register every step and result obtained, by default in files "/tmp/arp.log" and "/tmp/arp_detail.log".
- and implements all the needed functions to carry on the research tests.

The functions can be classified in the following groups:

- ARP table manipulation and visualization:
  - check_local_arp_table_entry()
  - remove_local_arp_table_entry()
  - remove_target_arp_table_entry()
  - create_target_arp_table_entry()
  - set_ifconfig_arp()
  - show_local_arp_table()
  - show_target_arp_table()
- Traffic generators: force_arp_solicited(), ping_arp_req(), target_ping_arp_req() and send_packet().
- "arpplet" wrappers: arpplet() and arpplet_eth().
- ARP timeout tests functions: testTR1() to testTR8().
- All the functions that implement the packet taxonomy designed in section "ARP packet taxonomy: analyzing all ARP packet variations":
  - test_ifconfig_arp
  - test_src_mac
  - test_dst_mac
  - test_src_ip
  - test_dst_ip
  - test_type
  - test_table_status

  This taxonomy functions require a set of variables that reflect all the different possible variations inside an ARP packet and system ARP table and interface status.

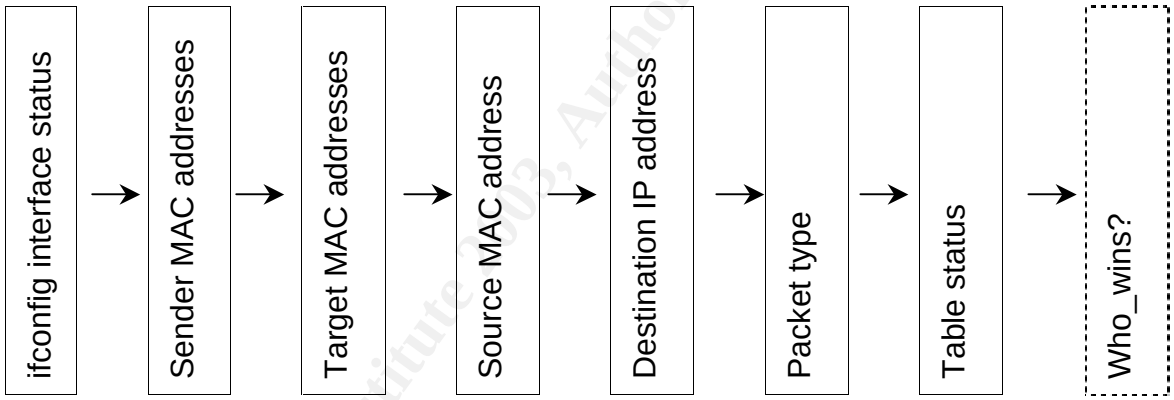
- Functions to select the packet taxonomy tests to run: print_packet_taxonomy_tests(), all_packet_taxonomy_tests() and select_*() functions.

The perl module has been designed to run on Linux Red Hat. With some minor changes it would be available for other Linux distributions because some command arguments are different. For example, Red Hat "ping" command set a waiting timeout value using the "-w" option while Debian uses the "-i" option.

The implementation code uses the following variables to map the scenario elements from section "ARP packet taxonomy tests":

Variables	System
$TARGET_IP, $TARGET_MAC	X
$DESTINATION_IP, $DESTINATION_MAC	Y
$OTHER_IP, $OTHER_MAC	Z
$ATTACKER_IP, $ATTACKER_MAC	H

The implementation has modified the design flow that a packet must cross to get all its values. One of the steps designed in the "ARP table and interface variables" section has been added as the first step. The change increases execution performance not being needed to change the network interface state frequently:



The following sections describes the specific group of tests run to map and extend the previous existent ARP classifications [BH2001] and [ARPSK1], called "Test BH" and "Test SK" respectively.

A skeleton has been created for automatically auditing an operating system ARP module, although the implementation has not been completed. In a future version in should be finished being the proposed method to execute commands remotely the SSH protocol. Its complexity resides in the operating system differences about how to manipulate and display the ARP table. The tool will need to:

- automatically search for ARP entries and add, modify or delete them from the target table.
- Configure the remote system network interface (ifconfig). Only for Unix systems.

- Generate simple traffic from the target system in order to run the timeout tests.

The user will only need to set the $AUTO variable equal to TRUE to run the tests without manual assistance.

Tests BH

These tests check if target favors Ethernet "Source MAC Address" or ARP "Source MAC Address" when learning new entries. This runs a total of 6 individual tests per OS.

TESTs IDs:
00 and 03 and 01 and 00 and 00 and 00 01 13 04

They check situations when sender MAC addresses have a unicast value and are different in the Ethernet and ARP headers. It ensures that the destination MAC addresses are valid, setting them to X, the target system, or a broadcast value. Finally, the IP addresses are filled with usual values, that is, coming from another host and addressed to the target IP. The packet generated must be an ARP request and there was no entry in the target ARP table previously.

The script that implements the tests is called "arp_packet_taxonomy_BH.pl".

Test SK

These tests check valid packets from the Ethernet and ARP perspective, ensuring the "Sender MAC Addresses" is equal at both layers. It also ensured valid "Target MAC Addresses" addressed to the target system or to the broadcast address. At the IP level all possible unicast combinations are checked, using two packet types, ARP requests and unsolicited replies. Finally, all the 3 possible ARP table states are analyzed.

TESTs IDs:
02 and 03 and 00 and 00 and 00 and 00 03 11 01 01 02 01 13 02 02 03

This runs a total of 180 individual tests per OS.

Due to the fact that some OS doesn't allow to change the default behaviour in which the network interface listen for ARP traffic, all the tests were run using this configuration, that is, setting the "ifconfig –arp" option to false. Only Unix OS can stop listening ARP through the "ifconfig –arp" command, therefore future tests should analyze how Unix systems behave when non-listening for the ARP protocol.

The script that implements these tests is called "arp_packet_taxonomy_SK.pl".

Results

A new graphical table was designed to extract the results obtained for these sets of tests. Given the fact they only use IP unicast addresses and two packets types, ARP request and unsolicited ARP reply packets, the whole taxonomy table of Figure 2.16 can be slightly reduced.

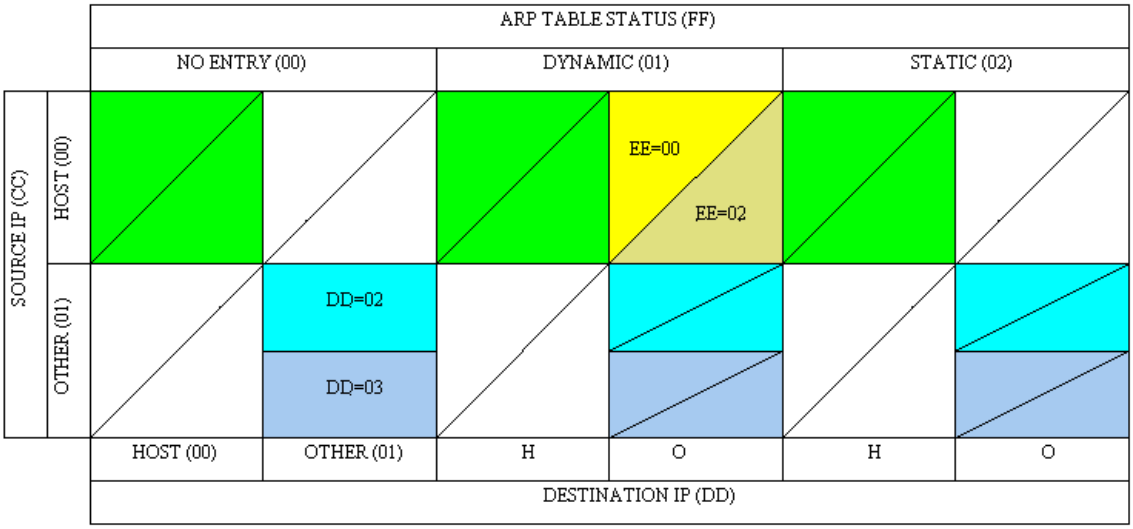


Figure IV.1. Packet Taxonomy Tests - working table

Once a specific "Sender MAC Addresses" and "Target MAC Addresses" values have been selected, that is, one specific cell in the main Packet Taxonomy matrix (Figure 2.16), all the possible combinations for unicast IP addresses can be written down using the figure above:

- Green boxes specify the target host gratuitous combinations, and light blue ones other host gratuitous ARP packets.
- EE=00 refers to ARP requests, while EE=02 refers to ARP unsolicited replies.
- DD=02 refers to same source and destination IP addresses for another host, while DD=03 refers to different IP addresses belonging to two distinct hosts.

For the dynamic tests, displayed in the column in the middle, the focus must be centred in seeing if the second packet overwrites the previously learnt dynamic entry.

Not to extend this paper's length too much the detailed result for every test, documented through the table above, have not been included. The conclusion in section "ARP packet taxonomy tests" summarizes all the results that have been obtained.

APPENDIX V: the "arp" command

General arguments comparison

As a very general overview, the main peculiarities about the "arp" command and its arguments in different OS will be compared. This information is useful from the script generation point of view because the tools need to take into account the variations every operating system introduces.

The first point that must be taken into account is how every different OS references the MAC addresses when using the ARP administrative commands:

OS	Symbol	Example
Unix variants	:	00:0C:EE:01:02:03
Windows	-	00-0C-EE-01-02-03
Cisco	.	000C.EE01.0203

Cisco IOS

The Cisco IOS OS doesn't have an "arp" command as its Unix and Windows counterparts, but it has equivalent commands. Both IOS devices, routers and switches use the same command set. The latest IOS (August 2003) version is 12.3.

To be able to display the ARP cache table the following command can be used:

```
Router>show arp
OR
Router>show ip arp

Router>show ip arp ethernet 0/0
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1              -   0003.6b88.8800  ARPA   Ethernet0/0
Router>show ip arp 0003.6b88.8800
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1              -   0003.6b88.8800  ARPA   Ethernet0/0
Router>show ip arp 192.168.1.1
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1              -   0003.6b88.8800  ARPA   Ethernet0/0

Router#show ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1              -   0003.6b88.8800  ARPA   Ethernet0/0
Internet  192.168.1.2              -   0e0e.0e02.0202  ARPA
Internet  192.168.1.254            4   0011.aa97.9797  ARPA   Ethernet0/0

Switch#sh ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.2              -   00d0.5888.8800  ARPA   VLAN1
Internet  192.168.1.254            6   0011.aa97.9797  ARPA   VLAN1

Age (min):
Age in minutes of the cache entry. A hyphen (-) means the address is local or
has been statically set using the "arp IP_address MAC_address arpa" command.
```

Setting a static ARP entry:

```
Router>en
Password:
Router#
Router#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#
Router(config)#arp 192.168.1.2 0e0e.0e02.0202 arpa
Router(config)#^Z

In configuration:
!
arp 192.168.1.2 0e0e.0e02.0202 ARPA
!
```

See ARP statistics. Really helpful to see ARP is not stateful:

```
Router>show ip traffic
…
ARP statistics:
  Rcvd: 0 requests, 3 replies, 0 reverse, 0 other
  Sent: 5 requests, 53 replies (0 proxy), 0 reverse
```

Deleting entries from the ARP table.

```
Router#clear arp-cache
Router#show ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1            -   0003.6b88.8800  ARPA   Ethernet0/0
Internet  192.168.1.2            -   0e0e.0e02.0202  ARPA
Router#

Only static entries are kept.
```

Use the "clear arp interface ethernet 0/0" command to clean up ARP entries (the entire ARP table) associated with an interface, for example, "eth 0/0".

An ARP table cannot be cleared on a single entry basis using a Cisco IOS software command. The only relevant command in the Cisco IOS software is the "clear arp-cache" command, but this clears the entire ARP table; not just a single entry within the table. To clear a single entry the SNMP protocol must be used [CLEA1].

Get entry to delete through "snmpwalk" command, and using the "snmpset" command change its value to 2, that set it to invalid:

```
# snmpset 192.168.1.100 private ipNetToMediaType.1.192.168.1.200 i 2
```

Delete a static ARP entry:

```
Router#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#no arp 192.168.1.2 0e0e.0e02.0202 ARPA
Router(config)#^Z
00:23:41: %SYS-5-CONFIG_I: Configured from console by console
Router#show ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1            -   0003.6b88.8888  ARPA   Ethernet0/0
Router#
```

The IOS interface configuration mode allows setting the ARP expiration timeout for entries, using the "arp timeout" command.

```
Router(config)#int e 0/0
Router(config-if)#arp ?
  arpa         Standard arp protocol
  frame-relay  Enable ARP for a frame relay interface
  probe        HP style arp protocol (see comment)
  snap         IEEE 802.3 style arp
  timeout      Set ARP cache timeout
```

Although Cisco on-line help seems to denote the HP probe [HPP1] protocol as ARP related, it is not related to it at all. They are only similar because both are layer 2 protocols working with and mapping IP and MAC addresses.

Just to avoid confusion with the generic ARP gratuitous packets, the following Cisco IOS command applies to gratuitous ARPs for PPP/SLIP peer addresses:

```
Router(config)#ip gratuitous-arps
```

A Cisco router will send out a gratuitous ARP message when a client connects and negotiates an address over a PPP connection. This transmission occurs even when the client receives the address from a local address pool.

Cisco CatOS

To complement the Cisco IOS OS analyzed along this document, we will cover some commands of the CatOS OS. The CatOS is the operating system used in some Cisco Catalyst switches, as the 5000 family. Other newest switches, as the 6000/65000 families can also use CatOS although Cisco goal is to unify all their network devices under a common OS, the IOS.

The CatOS [CATC1] defines a unique timeout parameter to control the period of time after which an ARP entry is removed from the ARP table.

Parameter:	set arp agingtime
Description:	Used to set the period of time after which an ARP entry is removed from the ARP table. Setting this value to 0 disables aging.
Actions:	GET: Console> show arp ARP Aging time = 1200 sec 198.162.1.209 at 00-40-01-10-ec-31 198.162.1.40 at 08-00-f0-ff-f1-ac Console> … SET: (seconds) Console> (enable) set arp agingtime 1800 ARP aging time set to 1800 seconds.

Min.	**Default**	**Max.**
1 (never)	1200 (20 min.)	1 million (~ 277 h.)

It also allows creating new static ARP entries:

```
Console> (enable) set arp 198.168.1.2 00-0c-0c-44-44-44
ARP entry added.
```

Use the "clear arp" command to delete a specific entry or all entries from the ARP table:

```
Console> (enable) clear arp 198.168.1.2
ARP entry deleted.
Console> (enable) clear arp all
ARP table cleared. (9)
```

The (9) indicates the number of entries cleared.

Use the "show arp" command to display the ARP table.

```
Console> show arp
ARP Aging time = 1800 sec
hostname2 at 00-44-0b-44-44-88
198.168.1.2 at 00-44-0b-44-cc-44
198.168.1.4 at 08-40-20-44-44-44
Console>
```

HP-UX 11

Manual pages: (1m) arp and (7p) arp.

To use the ARP protocol, the "ether" encapsulation method for the network interface must be used.

HP-UX doesn't show static entries through the "arp -a" command. They must be seen using the "ndd" command. The reason is that "arp" only shows MIB2 information, not the actual ARP cache.

The "arp –D" command, not available in HP-UX 10.20, deletes a local interface permanent entry. In this situation system won't reply to ARP requests, so communication is only possible when initiated by local system.

Types of table entries:

- permanent (-s)
- temporary (temp)
- published (pub): which means that this system will act as an ARP proxy server responding to requests for hostname even though the host address is not its own.
- Trailer encapsulations allowed (trail), only in HP-UX 10.20.

It has a mapping for the multicast entry of 224.0.0.0 that can be visualized with the "ndd" command.

Linux: kernel 2.4

Manual pages: (8) arp and (7) arp.

As HP-UX, "ether" is the default encapsulation method for the network interfaces and required for ARP.

Types of table entries:

- permanent (M flag)
- temp
- pub (P flag)
- complete (C flag)

As of kernel 2.2.0 it is no longer possible to set an ARP entry for an entire subnet. Linux instead does automagic Proxy ARP when a route exists and it is forwarding.

Windows 2000 SP3

Windows doesn't implements trailers, published or temp entries. Its entries can be "static" or "dynamic":

```
I:\>arp -s 192.168.1.200 01-01-01-01-01-01
I:\>arp -a
Interface: 192.168.1.254 on Interface 0x1000005
  Internet Address      Physical Address      Type
  192.168.1.200         01-01-01-01-01-01     static
  192.168.1.6           00-60-bb-ff-44-88     dynamic
  192.168.1.15          00-a0-bb-eb-bf-00     dynamic
```

Solaris 8

Solaris 8, also known as SunOS 5.8, document the ARP module in the following man pages: (1m) arp and (7p) arp.

Types of table entries:

- pub
- temp
- trail
- unresolved: waiting for an ARP response.

It has a mapping for the multicast entry of 224.0.0.0 that can be visualized with the "arp" and "ndd" commands.

Execution privileges

There are three main actions that can be executed through the "arp" command:

- Static entry creation: arp –s.
- Entry removal: arp –d.
- ARP table visualization: arp –a.

All operating systems allow non-privileged users to visualize the table, but require root (Unix), Administrator (Windows) or enable (Cisco) privileges to be able to add or delete an ARP entry.

Output format per Operating System

This section shows an example of the different output the "arp" command generates in every different operating system analyzed. The discrepancies between OS must be considered when programming scripts to extract the desired information from the ARP table:

```
- Cisco IOS:
--------------

Router#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1              -   0003.6bff.0c00  ARPA   Ethernet0/0
Internet  192.168.1.254           61   000e.0e0e.0e0e  ARPA   Ethernet0/0
Router#

- HP-UX 10.20, 11 y 11i:
--------------------------

system:/>arp -a
192.168.1.1 (192.168.1.1) at 0:0:c:7:ac:7 ether
systemA.domain.com (192.168.1.96) at 8:0:9:12:05:da ether
systemB.domain.com (192.168.1.67) at 8:0:20:07:97:a1 ether
systemC.domain.com (192.168.1.62) at 0:0:c8:75:7c:29 ether
systemD.domain.com (192.168.1.60) at 0:7:57:95:07:59 ether
systemE.domain.com (192.168.1.58) at 0:7:57:9a:d0:81 ether
systemF.domain.com (192.168.1.57) at 0:90:27:d1:d2:9d ether
10.0.0.1 (10.0.0.1) -- no entry
? (192.168.1.1) at 0:0:c:7:ac:7 ether
192.168.1.61 (192.168.1.61) -- no entry

system:/>arp -an
 (192.168.1.1) at 0:0:c:7:ac:7 ether
 (192.168.1.96) at 8:0:9:12:05:da ether
 (192.168.1.67) at 8:0:20:07:97:a1 ether
 (192.168.1.62) at 0:0:c8:75:7c:29 ether
 (192.168.1.60) at 0:7:57:95:07:59 ether
 (192.168.1.61) at 0:7:57:9a:9e:c0 ether
 (192.168.1.58) at 0:7:57:9a:d0:81 ether
 (192.168.1.57) at 0:90:27:d1:d2:9d ether
 (192.168.1.52) at 0:70:6e:1c:71:6e ether
10.0.0.1 (10.0.0.1) -- no entry
192.168.1.61 (192.168.1.61) -- no entry

"arp -A": not documented in the "man" pages.

system:/# arp -An
 (192.168.1.1) at 0:0:f:7:af:7 ether; arp flags: 0x7
 (192.168.1.96) at 8:0:9:12:05:da ether; arp flags: 0x7
 (192.168.1.100) at 0:70:6e:7:d9:8a ether; arp flags: 0x7
 (192.168.1.66) at 0:70:6e:58:85:d6 ether; arp flags: 0x7
 (192.168.1.277) at 8:0:9:f0:8a:21 ether; arp flags: 0x7
192.168.1.67 (192.168.1.67) -- no entry

system:/# arp -A
192.168.1.1 (192.168.1.1) at 0:0:f:7:af:7 ether; arp flags: 0x7
sys1.domain.com (192.168.1.96) at 8:0:9:12:05:da ether; arp flags: 0x7
```

```
sys2.domain.com (192.168.1.21) at 0:0:6e:7:d:8a ether; arp flags: 0x7
sys7.domain.com (192.168.1.6) at 0:0:e:58:85:d6 ether; arp flags: 0x7
sys5.domain.com (192.168.1.2) at 8:0:9:f0:8a:21 ether; arp flags: 0x7
192.168.1.67 (192.168.1.67) -- no entry

- Linux 2.4:
-------------

# arp -a
d.domain.com (192.168.1.11) at 08:00:09:C6:55:87 [ether] on eth0
a.domain.com (192.168.1.57) at 08:00:09:EE:02:51 [ether] on eth0
o.domain.com (192.168.1.96) at 08:00:09:12:05:DA [ether] on eth0
? (192.168.1.1) at 00:00:0C:07:AC:07 [ether] on eth0

# arp -ae
Address         HWtype  HWaddress           flags Mask            Iface
d.domain.com     ether   08:00:09:C6:55:87   C                     eth0
a.domain.com     ether   08:00:09:EE:02:51   C                     eth0
o.domain.com     ether   08:00:09:12:05:DA   C                     eth0
192.168.1.1      ether   00:00:0C:07:AC:07   C                     eth0

# arp -an
? (192.168.1.57) at 08:00:09:EE:02:51 [ether] on eth0
? (192.168.1.96) at 08:00:09:12:05:DA [ether] on eth0
? (192.168.1.1) at 00:00:0C:07:AC:07 [ether] on eth0

- Windows 2000 SP3:
--------------------

I:\>arp -a

Interface: 192.168.6.255 on Interface 0x1000005
  Internet Address      Physical Address      Type
  192.168.1.200         01-01-01-01-01-01     static
  192.168.5.6           00-00-00-cd-75-98     dynamic
  192.168.5.15          00-a0-c9-e9-8c-0d     dynamic
  192.168.5.255         00-00-6e-21-00-95     dynamic
  192.168.6.121         00-90-07-07-6a-a5     dynamic
  192.168.7.165         00-01-06-85-cc-95     dynamic
  192.168.7.202         00-a0-c9-00-ea-d5     dynamic

- Solaris 8:
-------------

# arp -a

Net to Media Table: IPv4
Device   IP Address               Mask      flags   Phys Addr
------ -------------------- --------------- ----- ---------------
hme0   h.domain.com        255.255.255.255       08:00:09:c6:55:87
hme0   192.168.1.1         255.255.255.255       00:00:0c:07:ac:07
hme0   o.domain.com        255.255.255.255       08:00:09:12:05:da
hme0   p51.domain.com      255.255.255.255       08:00:20:07:97:a1
hme0   sp2.domain.com      255.255.255.255       00:07:57:95:07:59
hme0   sp7.domain.com      255.255.255.255       00:07:57:9a:9e:c0
hme0   tr5.domain.com      255.255.255.255       00:07:57:9a:d0:81
hme0   spp2.domain.com     255.255.255.255       00:90:27:d1:d2:9d
hme0   tr2.domain.com      255.255.255.255       00:70:6e:1c:92:00
hme0   10.0.0.1            255.255.255.255       00:d0:07:91:1c:88
#
```

APPENDIX VI: First traffic seen in the network

From the network side, and showing the importance of the ARP protocol, it is necessary to analyze, that commonly, the ARP traffic generated by a system constitute the first packet (or set of packets) seen by the network, and therefore, by any other systems in the same LAN coming from that system.

This first traffic is really important in switching environments where the first packet seen from a host lets the switch learn a new CAM association, joining MAC address with a physical switch port. This first packet will be typically an ARP request or a gratuitous ARP packet.

Other monitoring solutions, like "arpwatch", could also learn information based on the first packets seen in the network. This increases the importance of this initial traffic too.

If the first traffic seen is ARP, very frequently except in some cases defined bellow, two situations can occur: the system can be using name resolution, DNS, or not.

If it is not using DNS, the first traffic can be addressed to:

- the local network, so a broadcast ARP request packet asking for the end system constitutes the first packet thrown into the network.
- to a remote network, so the first traffic will be the same request but addressed to the local router.

If it is using DNS, the first ARP query will be addressed to the DNS server if it is located in the same LAN or to the router again if the DNS servers are placed in a remote subnet.

There are some exceptions where the ARP packets are not the first ones seen in the network coming from a specific host:

- BOOTP (Bootstrap Protocol) is used for bootstrapping diskless systems to find its IP address. It uses broadcast addresses and ARP is not involved at all.
- DHCP (Dynamic Host Configuration Protocol) provides a framework for passing information to hosts on a TCP/IP network. DHCP is based on BOOTP.
- Windows boxes generate tones of IP broadcasted traffic, associated to NetBIOS, that arrive to the whole subnet and doesn't require a previous ARP mapping because it uses the MAC broadcast address.

APPENDIX VII: ARP flux

Linux ARP filter configuration parameter is configured through "/proc/sys/net/ipv4/conf/*/arp_filter".

This functionality allows filtering ARP replies based on the routing table, restricting ARP according to the routes. It is useful for various things, one of them being automatic load balancing for incoming connections using multipath routes, while another is fault tolerance solutions.

When a Linux box has two interfaces configured, not necessarily in the same subnet, when a system sends an ARP request for one of this two IP addresses, both interfaces will respond each with its MAC address. The requester system will receive two ARP replies, so based on its OS, the first or the last reply will win and be kept into the ARP table.

This behaviour is known as the "ARP flux" problem [MART1]. This feature can cause problems, for example when one network interface in a multihomed system fails and you don't know right away.

There are some solutions to prevent this packet duplication:

- Although this is the default Linux behaviour it can be turned off in 2.4 kernels by setting the "arp_filter" parameter:

  ```
  # echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter
  ```

  Linux 2.2 kernels must use an old equivalent parameter: "hidden":

  ```
  # echo 1 > /proc/sys/net/ipv4/conf/all/hidden
  # echo 1 > /proc/sys/net/ipv4/conf/<ethX>/hidden
  ```

  This ensures that interfaces will only respond to ARP requests for its IP address and not for all the IP addresses configured in the system.
- Use "ifconfig -arp " command to force an interface not to respond to ARP requests. Hosts which want to send information to that interface IP address may need to manually add the proper MAC address static entry to their ARP tables.
- Use the standard Linux packet filtering tool, iptables, and filter the ARP packets so that only the proper traffic gets through. When an ARP request for the MAC address of an interface arrives, filter out ARP replies from all the other interfaces. See "Filtering devices" section. Example:

  ```
  # iptables -A INPUT -m mac --mac-source 0E:0E:0E:F1:F2:F3\
  -s server -j ACCEPT
  ```

There are some other solutions [MART1] that were designed during the development of the Linux Virtual Server project [LVS2]: the "ip arp" tool and the "noarp" route flag.

APPENDIX VIII: ARP table snapshots

This section includes different special OS behaviours related to the ARP table management.

ARP static entries for its IP address

Every OS behaves differently when setting a new ARP static entry for its IP address.

HP-UX 10.20 doesn't allow the creation of an ARP static entry for the IP address of the system, while Cisco IOS allows it and overwrites its own entry. Cisco IOS always shows an entry associating its MAC address to its IP address. When the new static entry is accepted, the MAC address is overwritten. Then if the static entry is manually removed, the real entry takes its place again.

Solaris is even worse, because in the same way as IOS does, it always maintains an entry reflecting its MAC and IP addresses relationship, but when a static entry for the same IP is created, not only it overwrites the MAC information but it uses it. When the entry is manually deleted, the entry disappears and it is not possible to communicate using the ARP protocol. Warning messages are generated in the system syslog with the following text:

```
"ar_entry_query: Could not find the ace for source address
192.168.1.100"
```

Windows 2000 allows setting an entry for its IP address but it doesn't affect the communications. The entry can be deleted as any other entry.

Linux kernel 2.4 doesn't allow the addition of an entry for its IP address. It generates an error:

```
# arp -s 192.168.1.2 08:00:09:EE:EE:EE
SIOCSARP: Invalid argument
```

Some of the proposed tests force this configuration, that is, setting a different MAC address for the local IP address using a static ARP entry.

There is another situation that can severely affect a system ARP module that is tested in this research too: when a system receives ARP packets claiming to be its IP address.

During the past there was a Cisco DoS vulnerability [CISVU1], where it was possible to send an ARP packet on a local broadcast interface, like Ethernet, which could cause a router or switch running specific versions of Cisco IOS to stop sending and receiving ARP packets on this interface. Reason is that ARP packets received by the router for the router's own interface IP address but a

different MAC address will overwrite the router's MAC address in the ARP table with the one from the received ARP packet.

ARP static entries for another IP network

Setting an ARP static entry for another IP network is sometimes not possible depending on the OS used.

Cisco IOS router or switch

```
Switch(config)#arp 1.1.1.1 0e0e.0e0e.0e0e arpa
Switch(config)#^Z
Switch#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  1.1.1.1                -   0e0e.0e0e.0e0e  ARPA
```

HP-UX 10.20

```
# arp -s 192.168.150.200 01:01:01:01:01:01
192.168.150.200: Network is unreachable
```

HP-UX 11 and 11i

```
# arp -s 192.168.150.200 01:01:01:01:01:01
SIOCSARP: Invalid argument
```

Linux kernel 2.4

```
# arp -s 192.168.150.200 01:01:01:01:01:01
SIOCSARP: Network is unreachable
```

Windows 2000 SP3

It accepts ARP entries for other networks.

```
I:\>arp -s 192.168.150.200 01-01-01-01-01-01
I:\>arp -a
Interface: 192.168.1.254 on Interface 0x1000005
  Internet Address      Physical Address      Type
  192.168.150.200       01-01-01-01-01-01     static
  192.168.1.6           00-60-60-fd-64-98     dynamic
  192.168.1.15          00-a0-60-e6-86-06     dynamic
```

Solaris 8

```
# arp -s 192.168.150.200 01:01:01:01:01:01
192.168.150.200: No such device or address
```

ARP entries without response

This section pretends to evaluate which state is used by every OS analyzed when a response, ARP reply, is not received for a given ARP request, as when the destination system is not available. The entry is created when the ARP request is sent.

Cisco IOS

Cisco IOS as others OS show these entries in the "Incomplete" state:

```
Router#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.200           0   Incomplete      ARPA
```

HP-UX 10.20

HP-UX 10.20 reflects this type of entries as "incomplete" too:

```
# arp -an
 (192.168.1.200) at (incomplete)
```

Linux kernel 2.4

Linux also keeps the entries in the "incomplete" state. When an entry is manually deleted in Linux it is always kept in the same state:

```
# arp -d 192.168.1.200
# arp -an:
? (192.168.1.200) at <incomplete> on eth0
```

Windows 2000

When an ARP request is made but there is no response entry is kept in the "invalid" state:

```
C:\>arp -a
Interface: 192.168.1.254 on Interface 0x1000005
  Internet Address      Physical Address      Type
  192.168.1.10          00-00-00-00-00-00     invalid
```

When the entry is removed manually, using the "arp –d" command, the entry disappears and a message is shown: "No ARP Entries Found".

Solaris 8

Solaris 8 uses the "U" flag to mark entries that have not received an ARP reply:

```
# arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
...
hme0   192.168.1.200        255.255.255.255 U
```

APPENDIX IX: "arpplet" source code

This is the main ARP packet generation tool used during this research. All other scripts and tests use it when they need to generate an ARP crafted packet.

To compile it, use the default "cc" options:

```
# cc -o arpplet arpplet.c
```

It has been tested in different Red Hat Linux kernel 2.4 versions: 7.1, 7.3 and 8.

"arpplet" help message

```
# ./arpplet -h

Usage:      arpplet     [-hve] [-i interface] [-t timeout] [-c count]
            [-S src_mac_eth] [-D dst_mac_eth]
            [-H h_length] [-P p_length] [-O op_code]
            sender_mac sender_ip target_mac target_ip

 * Running options:
 -------------------
 -c count: number of packets to be sent. (default is 1)
           if count is 0, it will never stop (infinite loop).
 -t timeout: timeout (in seconds) between sent packets. (default is 1
second)
           if timeout is 0 it goes as fast as possible.
 -i interface: specify the network interface to be used. (default is
eth0)
 -h: shows this usage/help information message.
 -e: shows a usage example message (including this help message).
 -v: shows verbose information during execution.

 * Ethernet header fields:
 --------------------------
 -S: source MAC address in Ethernet header. (default is "sender_mac")
 -D: destination MAC address in Ethernet header. (default is
"target_mac")

 * ARP packet fields:
 ---------------------
 -H: hardware length. (default is 6)
 -P: protocol length. (default is 4)
 -O: operation code: request (1) or reply (2, default).

 sender_mac: ethernet sender hardware (MAC) address.
 sender_ip: sender IP address.
 target_mac: ethernet target hardware (MAC) address.
 target_ip: target IP address.
```

"arpplet" source code

```
/*
 * arplet.c
 *
 * Version:           1.01 (11 June 2003)
```

```
 * First Creation Date:       17 May 2003
 * Author:            Raul Siles Pelaez
 * e-mail:            raul_siles@hp.com
 *
 * Description:
 * [to be created]
 *
 */

/* Acknowledgments:
 *
 * The free source code provided by some people through Internet was
 * really helpful for learning ARP packet programming:
 * -"Alexey Kuznetsov" - tool: "arping.c".
 *  (it belongs to the "iputils" Linux package)
 * -"Yuri Volobuev" - tool: "send_arp.c".
 *
 * Hey, what about the Russian's guys and the ARP protocol.
 * It seems they are really involved in playing with it ;-)
 */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netdb.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <sys/ioctl.h>

/* Information and error messages size */
#define MESSAGE_SIZE 1024

/*
 * CONSTANT DEFINITIONS:
 * ---------------------
 */

/* /usr/include/linux/if_ether.h */
#define FRAME_TYPE ETH_P_ARP  /* arp = 0x0806 */
#define PROT_TYPE ETH_P_IP    /* ip = 0x0800 */
#define HARD_LENGTH ETH_ALEN  /* 6 octects */

/* Myself */
#define PROT_LENGTH 4         /* 4 octects */
#define DEFAULT_INTERFACE "eth0"

/* /usr/include/linux/if_arp.h */
#define HARD_TYPE ARPHRD_ETHER        /* 1: Ethernet 10Mbps */

/* /usr/include/net/if_arp.h */
#define ARP_REQUEST ARPOP_REQUEST     /* 1 */
#define ARP_REPLY ARPOP_REPLY         /* 2 */


/*
 * RFC 826: ARP protocol:
 * ----------------------
 */

/*
 * ETHERNET ARP PACKET (packet):
 * Ethernet Header (ethernet) + ARP payload (ether_arp)
 * (42 bytes on wire)
 *
 * Ethernet Header: 14 bytes
 * - Destination address: 6 bytes
 * - Source address: 6 bytes
 * - Frame type: 2 bytes
 *
 * ARP payload: 28 bytes
 * - Hardware type: 2 bytes
 * - Protocol type: 2 bytes
 * - Hardware size: 1 byte
```

```
 * - Protocol size: 1 byte
 * - Op code: 2 bytes
 * - Sender MAC: 6 bytes
 * - Sender IP: 4 bytes
 * - Target MAC: 6 bytes
 * - Target IP: 6 bytes
 *
 */

/* - General summary information about Ethernet frames:
 *
 * Ethernet header length = 14
 * Total data field length = 1500
 * Maximum frame length = 1514
 * ARP data field length = 28 of 1500
 */

/* RFC 826: ARP packet (header + addresses) */
/*
 * - Defined in Linux:
 * /usr/include/net/if_arp.h --> struct      arphdr (ARP header)
 * /usr/include/netinet/if_ether.h --> struct ether_arp (ARP packet)
*/

/*
 * Ethernet header structure:
 */
struct ethernet {
        unsigned char dst_addr[HARD_LENGTH];
        unsigned char src_addr[HARD_LENGTH];
        unsigned short frame_type;
};

/*
 * Complete Ethernet/ARP packet
 */
struct packet {
        /* Ethernet header */
        struct ethernet eth_hdr;
        /* ARP payload: header + addresses (4) */
        struct ether_arp arp_packet;

#if 0
        /* Padding is not needed */
        /* Padding data: 18 bytes, to reach minimum: 64 */
        unsigned char padding[18];
#endif
};


/*
 * GLOBAL VARIABLES:
 * -----------------
 */

/* Program name */
char * program = "arpplet";
/* By default, no vebose information is displayed */
int verbose = 0;
/* Information and error messages buffer */
char * m;


/*
 * AUXILIARY FUNCTIONS:
 * --------------------
 */

/*
 * Print error messages and terminate program execution.
 */
void error(char* message) {
        fprintf(stderr,"%s: error -->\n",program);
        fprintf(stderr,"%s\n\n",message);
        exit(-1);
}
```

```
/*
 * Print a verbose message ONLY if verbose is set to 1.
 * Does not terminate program execution.
 */
void message(char* message) {
       if (verbose) {
          fprintf(stderr, "%s\n", message);
       }
}

/*
 * Check if the user that is running this program has root privs.
 */
int ami_root() {
       return (!getuid());
}


/*
 * Ascii to HEX conversion function:
 *
 * Examples.-
 * 48 --> 0, 49 --> 1 ... 57 --> 9, and
 * 97 --> a, 98 --> b ... 102 --> f
 */
char ascii_hex(char a) {

       char c = tolower(a);
       char result;

       if (isdigit(c)) { result = c-'0'; }
       else if (isxdigit(c)) { result = c-'a'+10; }
       else { result = -1; }

       return result;
}


/*
 * Set the Hardware/Ethernet/MAC address.
 *
 * This function understand two MAC addresses formats:
 * 01:02:03:1A:1B:1C and 0102031A1B1C
 */
void set_hw_address(char* dest, char* src) {

       int i;
       char c;

       /* MAC address comes in ASCII format from user ("src" variable) */
       for (i=0;i<HARD_LENGTH;i++) {
              /* First element */
              if ((c=ascii_hex(*src++)) < 0) {
                     error("Wrong hardware address.");
              } else {
                     *dest = c << 4;
              }
              /* Second element */
              if ((c=ascii_hex(*src++)) < 0) {
                     error("Wrong hardware address.");
              } else {
                     *dest++ |= c;
              }
              if (*src == ':') src++;
       }
}


/*
 * Set the IP address.
 */
void set_ip_address(char* dest, char* src) {

       struct in_addr ip_addr;
       struct hostent *host;

       /* Resolve IP if necesary and ... */
```

```
        /* convert number-dot notation into binary (network byte order) */
        if (inet_aton(src, &ip_addr) == 0) {

                if (host=gethostbyname(src)) {
                        bcopy(host->h_addr_list[0], &ip_addr, host->h_length);
                } else {
                        sprintf(m, "Unknown host %s (CODE=%i).",src ,h_errno);
                        error(m);
                }
        }
        memcpy(dest,&ip_addr,PROT_LENGTH);
}


/*
 * Get the network interface index to be used to send the packets.
 */
int get_interface(int socket, char* int_name) {

        int i;
        struct ifreq if_request;

        /* Setting network interface to be used to send ARP packets */
        memset(&if_request, 0, sizeof(if_request));
        strncpy(if_request.ifr_name, int_name, IFNAMSIZ-1);
        if (ioctl(socket, SIOCGIFINDEX, &if_request) < 0) {
                sprintf(m, "Unknown network interface %s.",int_name);
                error(m);
        }
        i = if_request.ifr_ifindex;

        sprintf(m, "Using interface: %s (id:%i)", int_name, i);
        message(m);
        bzero(m,sizeof(m));

        /* It also checks if the net interface proposed by the user is valid.*/
        /* Checking interface status and features */
        if (ioctl(socket, SIOCGIFFLAGS, (char*)&if_request)) {
                error("Interface features: ioctl(SIOCGIFFLAGS).");
        }
        if (!(if_request.ifr_flags&IFF_UP)) {
                error("The specified network interface is down.");
        }
        if (if_request.ifr_flags&(IFF_NOARP|IFF_LOOPBACK)) {
                error("The specified network interface is not ARPable.");
        }

        return i;
}


/*
 * Usage or help function.
 */
void usage() {
        fprintf(stderr,
        "\nUsage: \tarpplet\t[-hve] [-i interface] [-t timeout] [-c count]\n"
        "\t\t[-S src_mac_eth] [-D dst_mac_eth]\n"
         "\t\t[-H h_length] [-P p_length] [-O op_code]\n"
        "\t\tsender_mac sender_ip target_mac target_ip\n\n");

        fprintf(stderr," * Running options:\n");
        fprintf(stderr," -------------------\n");
        fprintf(stderr,
        " -c count: number of packets to be sent. (default is 1)\n"
        "           if count is 0, it will never stop (infinite loop).\n"
        " -t timeout: timeout (in seconds) between sent packets. (default is 1 second)\n"
        "            if timeout is 0 it goes as fast as possible.\n"
        " -i interface: specify the network interface to be used. (default is eth0)\n"
        " -h: shows this usage/help information message.\n"
        " -e: shows a usage example message (including this help message).\n"
        " -v: shows verbose information during execution.\n\n");

        fprintf(stderr," * Ethernet header fields:\n");
        fprintf(stderr," --------------------------\n");
        fprintf(stderr,
        " -S: source MAC address in Ethernet header. (default is \"sender_mac\")\n"
```

```
        " -D: destination MAC address in Ethernet header. (default is
\"target_mac\")\n\n");

        fprintf(stderr," * ARP packet fields:\n");
        fprintf(stderr," ---------------------\n");
        fprintf(stderr,
        " -H: hardware length. (default is 6)\n"
        " -P: protocol length. (default is 4)\n"
        " -O: operation code: request (1) or reply (2, default).\n\n"

        " sender_mac: ethernet sender hardware (MAC) address.\n"
        " sender_ip: sender IP address.\n"
        " target_mac: ethernet target hardware (MAC) address.\n"
        " target_ip: target IP address.\n\n");

}

/*
 * Example function.
 */
void example() {

        fprintf(stderr," -----------\n");
        fprintf(stderr," * Example:\n");
        fprintf(stderr," -----------\n");
        fprintf(stderr,
        " # arpplet -c 10 01:02:03:04:05:06 1.2.3.4 0708090A0B0C 5.6.7.8\n\n"
        " It sends an ethernet frame from 01:02:03:04:05:06 to 0708090A0B0C\n"
        " (both MAC address notations are valid) containing an ARP reply packet\n"
        " saying to 5.6.7.8 (0708090A0B0C) that 1.2.3.4 is at 01:02:03:04:05:06\n\n"
        " * ETHERNET/ARP PACKET: (42 bytes)\n\n"
        " * Ethernet Header: (14 bytes)\n"
        "   - Destination address: 0708090A0B0C\n"
        "   - Source address: 01:02:03:04:05:06\n"
        "   - Frame type: 0x0806 (ARP)\n\n"
        " * ARP payload: (28 bytes)\n"
        "   - Hardware type: 0x0001 (Ethernet)\n"
        "   - Protocol type: 0x0800 (IP)\n"
        "   - Hardware size: 6\n"
        "   - Protocol size: 4\n"
        "   - Op code: 2 (reply)\n"
        "   - Sender MAC: 01:02:03:04:05:06\n"
        "   - Sender IP: 1.2.3.4\n"
        "   - Target MAC: 0708090A0B0C\n"
        "   - Target IP: 5.6.7.8\n\n");

}


/*
 * MAIN PROGRAM:
 * --------------
 */
int main (int argc, char** argv) {

int arg;
int i = 1;

/* By default, it only sends 1 packet */
int number_packets = 1;
/* By default, it doesn't run forever, remember it only sends 1 packet */
int infinite = 0;
/* By default, it sends ONLY 1 packet.
 * To make it run as fast as possible, without time interval, set timeout to "0" */
int timeout = 1;

/* Socket and packet structures */
int the_socket;
struct packet p;
struct sockaddr_ll sending_socket;

/* Local interface */
int interface;
char *interface_name = DEFAULT_INTERFACE;

/* Ethernet: MAC addresses */
unsigned char * source_addr = NULL;
```

```
unsigned char * destination_addr = NULL;

/* Hardware and Protocol sizes */
unsigned char hw_length = HARD_LENGTH;
unsigned char protocol_length = PROT_LENGTH;

/* ARP option: request or reply */
unsigned short int option = htons(ARP_REPLY);

/* ARP: MAC and IP addresses */
unsigned char * sender_hw_addr = NULL;
unsigned char * sender_ip_addr = NULL;
unsigned char * target_hw_addr = NULL;
unsigned char * target_ip_addr = NULL;



/* Initializing global variables */
program = argv[0];
if ((m = (char *) malloc(MESSAGE_SIZE)) == NULL) {
        error("Problem allocating memory for information and error messages.");
}
bzero(m,sizeof(m));


/* Checking root privileges */
if (!ami_root()) {
        error("Sorry, but you need to be root to run this program.\nJust having the
effective UID equal to zero is not enough :-)");
}


/* Through the program arguments it is possible to change all the relevant
 * fields of an Ethernet/ARP packet, that is:
 * - Source and destination MAC addresses in Ethernet header.
 * - Hardware and protocol sizes in ARP header.
 * - ARP operation code.
 * - ARP sender and target MAC and IP addresses.
 */
/* From a hacking point of view it has no sense manipulating the folowing fields:
 * - Ethernet Frame type (ARP): 0x0806
 * - Hardware and Protocol types (Ethernet:0x0001 and IP:0x0800)
 */

/* Get program arguments */

while ((arg = getopt(argc,argv, "hvec:t:i:S:D:H:P:O:")) != EOF ) {
        switch(arg) {
                case 'v':
                        verbose=1;
                        message("Parsing program arguments...");
                        break;
                case 'c':
                        number_packets=atoi(optarg);
                        if (number_packets == 0)
                                infinite=1;
                        break;
                case 't':
                        timeout=atoi(optarg);
                        break;
                case 'i':
                        interface_name=optarg;
                        break;
                case 'S':
                        /* Set the Ethernet source address */
                        source_addr=optarg;
                        break;
                case 'D':
                        /* Set the Ethernet destination address */
                        destination_addr=optarg;
                        break;
                case 'H':
                        /* Set the hardware length */
                        hw_length=atoi(optarg);
                        break;
                case 'P':
                        /* Set the protocol length */
```

```
                        protocol_length=atoi(optarg);
                        break;
                case 'O':
                        /* Set ARP option: request (1) or reply (2)*/
                        option=htons(atoi(optarg));
                        break;
                case 'e':
                        usage();
                        example();
                        exit(-1);
                case 'h':
                default:
                        usage();
                        exit(-1);
        }
}
argc -= optind;
argv += optind;

if (argc != 4) {
        message("All addresses, 2 MAC and  2 IP, MUST allways be specified");
        usage();
        exit(-1);
}

sender_hw_addr = *argv++;
sender_ip_addr = *argv++;
target_hw_addr = *argv++;
target_ip_addr = *argv;


/* If values for Ethernet header addresses have not been defined the default
 * values are extracted from the MAC adresses in the ARP packet */
if (source_addr == NULL) { source_addr = sender_hw_addr; }
if (destination_addr == NULL) { destination_addr = target_hw_addr; }


/* Building the Ethernet header */
/* ---------------------------- */
sprintf(m, "Building the Ethernet header: [dst:%s,src:%s,type:%x]",destination_addr,
source_addr, FRAME_TYPE);
message(m);
bzero(m,sizeof(m));

set_hw_address(p.eth_hdr.dst_addr, destination_addr);
set_hw_address(p.eth_hdr.src_addr, source_addr);
p.eth_hdr.frame_type = htons(FRAME_TYPE);


/* Building the ARP packet */
/* ----------------------- */
message("Building the ARP packet...");

/* Hard Type */
p.arp_packet.ea_hdr.ar_hrd = htons(HARD_TYPE);
/* Prot type */
p.arp_packet.ea_hdr.ar_pro = htons(PROT_TYPE);
/* Hard size */
p.arp_packet.ea_hdr.ar_hln = hw_length;
/* Prot size */
p.arp_packet.ea_hdr.ar_pln = protocol_length;
/* Op */
p.arp_packet.ea_hdr.ar_op = option;

/* Addresses:
 * ----------- */

/* Sender HW address */
set_hw_address(p.arp_packet.arp_sha, sender_hw_addr);
/* Sender IP address */
set_ip_address(p.arp_packet.arp_spa, sender_ip_addr);
/* Target HW address */
set_hw_address(p.arp_packet.arp_tha, target_hw_addr);
/* Target IP address */
set_ip_address(p.arp_packet.arp_tpa, target_ip_addr);

/* bzero(p.padding,18); */
```

```
sprintf(m, "--> ARP PACKET=
[hw:%x,prot:%x,hw_l:%i,prot_l:%i,opcode:%x,mac_src:%s,ip_src:%s,mac_dst:%s,ip_dst:%s]",H
ARD_TYPE, PROT_TYPE, hw_length, protocol_length, htons(option),sender_hw_addr,
sender_ip_addr, target_hw_addr, target_ip_addr);
message(m);
bzero(m,sizeof(m));


/* Linux 2.0:
the_socket = socket(PF_INET, SOCK_PACKET, htons(FRAME_TYPE));
*/

/* Linux 2.2 or higher */
the_socket = socket(PF_PACKET, SOCK_RAW, htons(FRAME_TYPE));

sprintf(m, "Creating socket (fd:%i)...", the_socket);
message(m);
bzero(m,sizeof(m));


/* Getting interface id: verbose message inside the function */
interface = get_interface(the_socket, interface_name);


/* sockaddr_ll: man 7 packet AND man 7 netdevice */
sending_socket.sll_family = AF_PACKET;
sending_socket.sll_ifindex = interface;
sending_socket.sll_protocol = htons(FRAME_TYPE);


/* LOOP */
i = 1;
while ((number_packets>0) | (infinite)) {

        /* Send packet */
        sprintf(m, "Sending ARP packet (n=%i)...", i++);
        message(m);
        bzero(m,sizeof(m));

        if (sendto(the_socket, &p, sizeof(p), 0, (struct sockaddr *)&sending_socket,
sizeof(sending_socket)) < 0) {
                error("Unable to send ARP packet.");
        }

        number_packets--;

        /* Wait ... */
        if (number_packets != 0) { sleep(timeout); }

} /* end LOOP */

exit(0);

} /* main */
```

APPENDIX XI: Google state of the art

Just as a curiosity trying to show the Internet/Google (www.google.com) state of the art about the ARP spoofing topic, these were the results obtained searching The Net the 12th of August of 2003:

- Searching Cisco by "arp": 5200 results.
- Searching Microsoft KB (http://support.microsoft.com) by "arp": 25 results.
- Searching Google by "arp": 993000 results (some of them not related with the networking protocol).
- Searching Google by "arp" "spoofing": 29400 results.
- Searching Google by "arp spoofing": 13800 results.
- Searching Google by "arp" "poisoning": 5380 results.
- Searching Google by "arp poisoning": 2540 results.

Last Updated: January 17th, 2018

Upcoming SANS Penetration Testing

Mentor Session - SEC542	Louisville, KY	Jan 24, 2018 - Mar 28, 2018	Mentor
SANS Dubai 2018	Dubai, United Arab Emirates	Jan 27, 2018 - Feb 01, 2018	Live Event
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
Las Vegas 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	vLive
Community SANS Charlotte SEC504	Charlotte, NC	Jan 29, 2018 - Feb 03, 2018	Community SANS
SANS Miami 2018	Miami, FL	Jan 29, 2018 - Feb 03, 2018	Live Event
Cyber Threat Intelligence Summit & Training 2018	Bethesda, MD	Jan 29, 2018 - Feb 05, 2018	Live Event
Community SANS Columbia SEC542	Columbia, MD	Feb 05, 2018 - Feb 10, 2018	Community SANS
SANS London February 2018	London, United Kingdom	Feb 05, 2018 - Feb 10, 2018	Live Event
Community SANS Kansas City SEC504	Kansas City, MO	Feb 05, 2018 - Feb 10, 2018	Community SANS
SANS Scottsdale 2018	Scottsdale, AZ	Feb 05, 2018 - Feb 10, 2018	Live Event
Mentor Session - SEC504	Detroit, MI	Feb 06, 2018 - Mar 20, 2018	Mentor
SANS Southern California- Anaheim 2018	Anaheim, CA	Feb 12, 2018 - Feb 17, 2018	Live Event
SANS Secure India 2018	Bangalore, India	Feb 12, 2018 - Feb 17, 2018	Live Event
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201802, Germany	Feb 13, 2018 - Mar 22, 2018	vLive
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Brussels February 2018	Brussels, Belgium	Feb 19, 2018 - Feb 24, 2018	Live Event
Cloud Security Summit & Training 2018	San Diego, CA	Feb 19, 2018 - Feb 26, 2018	Live Event
SANS Secure Japan 2018	Tokyo, Japan	Feb 19, 2018 - Mar 03, 2018	Live Event
SANS New York City Winter 2018	New York, NY	Feb 26, 2018 - Mar 03, 2018	Live Event
SANS vLive - SEC542: Web App Penetration Testing and Ethical Hacking	SEC542 - 201802,	Feb 27, 2018 - Apr 12, 2018	vLive
Mentor Session - SEC504	Seattle, WA	Mar 01, 2018 - Apr 12, 2018	Mentor
SANS London March 2018	London, United Kingdom	Mar 05, 2018 - Mar 10, 2018	Live Event
Community SANS Virginia Beach SEC504	Virginia Beach, VA	Mar 05, 2018 - Mar 10, 2018	Community SANS
Community SANS Portland SEC542	Portland, OR	Mar 05, 2018 - Mar 10, 2018	Community SANS
Mentor Session - SEC504	Stroudsburg, PA	Mar 06, 2018 - Apr 03, 2018	Mentor
Mentor Session - SEC504	Long Beach, CA	Mar 12, 2018 - May 21, 2018	Mentor
SANS Paris March 2018	Paris, France	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Secure Singapore 2018	Singapore, Singapore	Mar 12, 2018 - Mar 24, 2018	Live Event
Mentor Session - SEC560	Baltimore, MD	Mar 12, 2018 - Apr 12, 2018	Mentor
SANS San Francisco Spring 2018	San Francisco, CA	Mar 12, 2018 - Mar 17, 2018	Live Event