Handling ARP Based Spoofing Attacks

A dissertation submitted towards the partial fulfillment of the requirement for the Award of the Degree of

Master of Engineering
in
Computer Technology & Application

Submitted by
Vineet Garhewal
College Roll No: 15/CTA/04
University Roll No. 8519

Under the guidance of
Dr. D. Roy Choudhury

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI

2004 – 2006

CERTIFICATE

This is to certify that the work that is being presented in this dissertation entitled ***"Handling ARP Based Spoofing Attacks"*** submitted by **Vineet Garhewal** in the partial fulfillment of the requirement for the award of the degree of **Master of Engineering** in Computer Technology and Application, Delhi College of Engineering is an account of his work carried out under my guidance and supervision.

The work embodies in this dissertation has not been submitted for the award of any other degree to the best of my knowledge.

Dr. Goldie Gabrani
Head of Department
Department of Computer Engineering
Delhi College of Engineering
Delhi

Dr. D. Roy Choudhury
Professor
Department of Computer Engineering
Delhi College of Engineering
Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extent my heartiest felt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisor Professor. D. Roy Choudhury, Professor in the Department of Computer Engineering, Delhi College of Engineering, for her invaluable guidance, encouragement and patient review. His continuous inspiration only has made me complete this major project.

I would also like to take this opportunity to present my sincere regards to my teachers viz. Professor Goldie Gabrani, Dr S. K. Saxena, Mr. Rajeev Kumar and Mrs. Rajni Jindal for their support and encouragement.

I would like to express my heartiest felt regards to Faculty of Computer Engineering Department for providing facility and for their co-operation and patience.

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

Vineet Garhewal
M.E. (Computer Technology & Applications)
College Roll No. 15/CTA/04
University Roll No. 8519
Delhi College of Engg. Delhi

ABSTRACT

The classic Man-in-the-Middle(ARP Spoofing) attack relies upon convincing two hosts that the computer in the middle is the other host. This can be accomplished with a domain name spoof if the system is using DNS to identify the other host or address resolution protocol (ARP) spoofing on the LAN. This thesis is designed to introduce and explain ARP spoofing and DNS spoofing and their prevention measures.

Throughout the Internet, IP is used for communications. Once an IP packet comes into a Local Area Network (LAN) it must be converted into a packet that the LAN can understand. An IP packet is encapsulated into an Ethernet frame for local handling. Ethernet is the most common type of network and other types of networks handle communications differently. Ethernet does not communicate via IP addresses, but uses the hardware address instead. Ethernet uses the Address Resolution Protocol (ARP) to resolve IP addresses into hardware addresses. This hardware address is also called the Media Access Controllers (MAC) addresses. Once the destination's MAC address is determined, the encapsulated IP packet can be transmitted to the host. Every network device must have an unique MAC address for the Ethernet LAN to function correctly.

All Ethernet hosts and switches keep a list of MAC and IP address in a table and will request this information from the network when a new IP address is requested or a table entry expires. The TCP/IP protocol based host trusts on its IP/MAC table.

ARP spoofing attacks involve tricking a system into modifying its MAC table and sending the data to the attacker instead. **The purpose of this thesis is to look at several ways and methods for modifying ARP cache information and how to defend the network against ARP spoofing attacks.**

TABLE OF CONTENTS

1. Introduction……..……1

1.1. Introduction…….....1

1.2. No Authentication…….1

1.3. Problem Definition……...2

1.3.1. Address Resolution Protocol…….…….3

1.3.2. ARP Poisoning…….…....4

1.4 The Threat of ARP Attacks…….…....5

1.5 The symptoms of ARP spoofing attacks……5

1.6. Objective of the Thesis…….…5

1.7. Major Step of Our Approach…..…….…….7

1.8 Organization of the thesis……..……...8

2. Address Resolution Protocol (ARP)…….…….9

2.1. Introduction…….…...…...9

2.2. ARP Frame Format…….……10

2.3. Overview of Address Resolution Protocol Request-Reply..…12

2.4. The ARP cache……..…...16

3. ARP Spoofing Attacks…….……21

3.1. How do ARP Spoofing-Based Attacks Work?.....21

3.2. ARP Vulnerabilities…….……23

3.3. ARP Attacks…….…….24

3.3.1. Man in the Middle…….…….24

3.3.2. Denial of services…….……..24

3.3.3. Hijacking…….……25

3.3.4. Cloning……25

3.3.5. Sniffing……25
3.4. Detecting ARP spoofing attacks……26
3.5. Protecting Against ARP Spoofing Attacks……..…….26

4. DNS Cache Poisoning…….……28
4.1. Introduction……..…...28
4.2. What is DNS?.....28
4.3. What is DNS Cache Poisoning?.....33
4.4. Potential Consequences of Cache Poisoning……...35
4.4.1. Identity Theft …….....35
4.4.2. Distribution of Malware……36
4.4.3. Dissemination of False Information……..…36
4.4.4. Man-in-the-middle Attack ……...36
4.5. Protecting Your Assets……....36

5. Digital Signatures……..…….38
5.1. Introduction…….…. …..38
5.2. Signatures and the Law…….…….39
5.3. How Digital Signature Technology Works…….42
5.4. Public Key Certificates…….…..47
5.5. Authenticated Transacrion Illustration……...49
5.6. Digital Siganture Standard(DSS)……..…….55
5.6.1. Use if the DSA Algorithm……..……..55
5.6.2. DSA Parameters…….……...56
5.6.3. Signature Generation……..……..57
5.6.4. Signature Verification……..…….58
5.7. Benefits of Digital Signatures…….……..58
5.8. Challenges and Opportunities…….……60

6. Digital Signature Based ARP Protocol ……62

6.1. 6.1. LAN Architecture ……..……..62

6.2. DS based ARP Fame Format..……..……..63

6.3. DS based ARP Protocol……...…….……..64

6.3.1. DS based ARP Request……..……..64

6.3.1. DS based ARP Reply……..…….64

6.4. Implementation……..…….65

6.5. Proof of Concept…….65

7. Results……..….67

7. Related Work……..….67

8. Conclusions and Future Work……..…..71

9. References……....…..72

11. Bibliography……74

Chapter 1
Introduction

1.1. Introduction

LAN technology has never been secure. As with so many of our aging network technologies, the original developers of Ethernet local area networking system never imagined the incredible future their brainchild has experienced. Thirty years ago, when Network Interface Controllers (NICs) cost $5,000 each — as they originally did — and only the richest and stuffiest corporations could afford to network their $50,000 computers together, network security wasn't a consideration. The whole SYSTEM would be secure since the hardware would all be in a glass enclosed temple surrounded by serious men wearing white lab coats.

So the very idea that someone's unemployed neighbor, three apartments down on the floor below, might use an old wireless laptop they bought off eBay for $346, along with some freely downloadable software they got from the Internet, to take over your own wireless Ethernet-based network and capture your credit card information as you transfer funds into your online bank account ... was not something old Bob Metcalfe, the principal designer of Ethernet, ever considered or, frankly, could have possibly imagined.

1.2. No Authentication

The problem is that Ethernet, upon which virtually all modern LANs are based, was designed without any sort of authentication technology whatsoever. So it is horrifyingly trivial for any computer with access to an Ethernet LAN, to re-route any other computer's traffic through itself simply by impersonating one or more other computers on the LAN. In fact, any computer can re-route ALL of the LAN's traffic through itself, allowing it to not only monitor but also to edit and alter anything sent to or received from any other machine on the local network. Attackers take the

advantages of flaw in the design of the networking protocol rather than designing their own virus program or any other complex constructs.

1.3. Problem Definition

1.3.1. Address Resolution Protocol

When an Ethernet frame is sent from one host to another on the same LAN, the 48 bit Ethernet address determines the interface to which the frame is destined. The IP address in the packet is ignored. ARP provides the mapping between the 32 bit IPv4 address and the 48 bit Ethernet address. In the next chapter it has been explained how ARP works.

When a host needs to send an IP datagram as an Ethernet frame to another host whose MAC address it ignores, it broadcasts a request for the MAC address associated with the IP address of the destination. Every host on the subnet receives the request and checks if the IP address in the request is bound to one of its network interfaces. If this is the case, the host with the matching IP address sends a unicast reply to the sender of the request with the <IP address, MAC address> pair. Every host maintains a table of <IP, MAC> pairs, called *ARP cache*, based on the replies it received, in order to minimize the number of requests sent on the network. No request is made if the <IP, MAC> pair of interest is already present in the cache. ARP cache entries have a typical lifetime of 20 minutes, but some operating systems may reset the expiration time every time they use an entry, thus possibly delaying forever entry refresh.

ARP is a stateless protocol, i.e., a reply may be processed even though the corresponding request was never received. When a host receives a reply, it updates the corresponding entry in the cache with the <IP, MAC> pair in the reply. While a cache entry should be updated only if the mapping is already present, some operating systems, e.g., Linux and Windows, cache a reply in any case to optimize

performance. Another stateless feature of ARP is the so called *gratuitous ARP*. A gratuitous ARP is a message sent by a host requesting the MAC address for its own IP address. It is sent either by a host that wishes to determine if there is another host on the LAN with the same IP address or by a host announcing that it has changed its MAC address, thus allowing the other hosts to update their caches.

1.3.2. ARP Poisoning

By forging an ARP reply, an attacker may easily change the <IP,MAC> association contained in a host ARP cache. Since each host presumes its local cache to be trustworthy, the poisoned host will send IP packets encapsulated into Ethernet frames with a bogus MAC address as destination. This way the attacker may receive all the frames originally directed to some other host. If also the cache of the real destination host is poisoned, both communication flows are under the attacker's control. The attacker realizes a two-way man in the middle, where she can forward the received packets to the correct destination after inspecting and possibly modifying them. The two end points of the connection will not notice the extra hop added by the attacker if the packet TTL is not decremented.

Some operating systems, e.g., Solaris, will not update an entry in the cache if such an entry is not already present when an unsolicited ARP reply is received. Although this might seem an effective precaution against cache poisoning, the attack is still possible. The attacker needs to trick the victim into adding a new entry in the cache first, so that a future (unsolicited) ARP reply can update it. By sending a forged ICMP echo request as if it was from one of the two victims, the attacker has the other victim create a new entry in the cache. When the first victim receives the spoofed ICMP echo request, it replies with an ICMP echo reply, which requires resolving first the IP address of the original ICMP request into an Ethernet address, thus creating an entry in the cache. The attacker can now update it with an unsolicited ARP reply.

ARP poisoning is possible also in switched networks. A layer 2 switch accepts the traffic that comes into each port and directs it only to the port to which the destination host is connected, except for broadcast messages which are sent to all ports. Therefore sniffing is no longer possible by simply configuring the network interface in promiscuous mode. However, it is possible to poison a host cache by sending an unsolicited ARP reply to the host containing the attacker's MAC address. The same can be done against two hosts at the same time, thus allowing an attacker to intercept all the traffic between those two hosts, without the switch realizing it. Once the attacker has hijacked the packets of a communication, she can modify the payload or even inject new packets in the communication as long as the TCP sequence numbers are adjusted so as to maintain the communication synchronized. The following figure 1.1 shows an example of ARP spoofing.

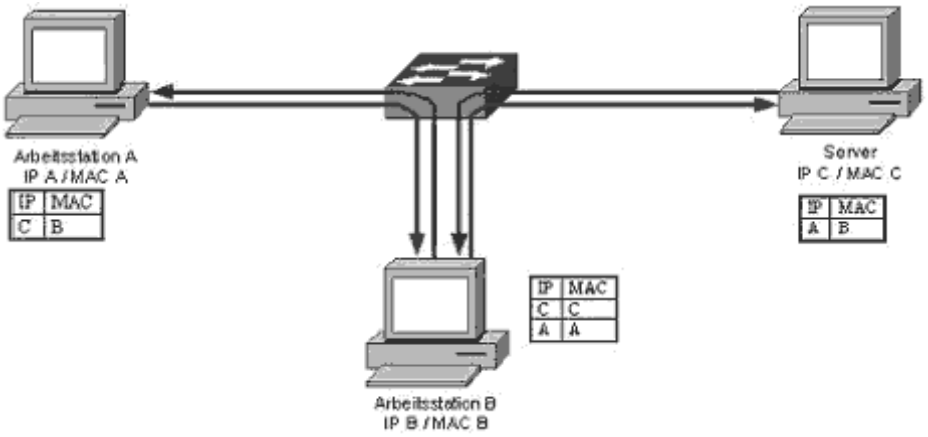


Figure 1.1 ARP spoofing(cache poisoning) attacks. These ARP attacks are usually successful even with encrypted connections like SSL, SSH or PPTP. ARP belongs to the OSI data link layer (layer 2).

1.4. The Threat of ARP Attacks

External attacks by hackers, viruses, worms and Trojans are permanent threats to any progressive company. What is not widely known, though, is that the major portion of attacks comes from within the network.

It has been reported that up to 80 % of all intrusions were initiated internally, from inside a company network. Technical ignorance, curiosity and intentional manipulation of data often lead to serious damages for organizations.

Internal network attacks are typically operated via so called ARP Spoofing or ARP Poisoning attacks. Malicious software to run ARP Spoofing attacks can be downloaded on the Internet by everyone. Using fake ARP messages an attacker can divert all communication between two machines with the result that all traffic is exchanged via his PC. By means of such as man-in-the-middle attack the attacker can in particular

- Run Denial of Service (DoS) attacks
- Intercept data
- Collect passwords
- Manipulate data
- Tap VoIP phone calls.

1.5. The Symptoms of ARP Spoofing Attacks

Confidential company information has leaked out and nobody knows how it could have happened.

- Your employees report about intrusion into their online bank account or into their email account.
- Inexplicable incidents have compromised the data of applications that can only be accessed via "secure" web interfaces.
- Strange occurrences in the ARP tables of your network.

1.6. Objective of the Thesis

The Address Resolution Protocol (ARP) is the glue that holds together the network and link layers of the IP protocol stack. The primary function of ARP is to

map IP addresses onto hosts hardware addresses within a local area network. As such, its correctness is essential to proper functioning of the network. However, like other protocols within IP, ARP is subject to a range of serious and continuing security vulnerabilities. Adversaries can exploit ARP to impersonate hosts, perform man-in-the-middle attacks, or simply DoS victims. Moreover, such attacks are trivial to perform, and few countermeasures have been widely deployed.

Current network environments present two central design challenges for ARP security. Firstly, the solution must not require ARP be discarded. The deployed base of IP is large and diverse enough that replacing any major component of the IP protocol stack is technically and cost prohibitive. Secondly, the costs of implementing ARP security must be minimal. Resource constrained devices and already computationally loaded hosts can not afford to budget large amounts of resources for ARP security. Any solution that would demonstrably change the performance profile of ARP will not be adopted.

In this thesis, we introduce the *Digital Signature based Address Resolution Protocol*. This protocol implements security by distributing centrally generated MAC/IP address mapping attestations. These attestations, called *certificates*, are given to clients as they join the network and are subsequently distributed through existing ARP messages. Unlike other popular ARP-based solutions, the costs per resolution are reduced to one public key validation per request/reply pair in the worst case. As such, this secure protocol is a feasible approach for the diverse assortment of existing network capable devices. This thesis provides a detailed description of the protocol design and its implementation within the **Linux** operating system.

This thesis is designed in such a way a novice LAN user can understand easily. This thesis starts with explaining the terminologies or LAN technologies in very simple manner with the descriptive examples.

1.7. Major Step of Our Approach

The Digital Signature based ARP protocol explained in this thesis has been implemented in Linux platform. Basically the existing ARP protocol is extended with the concept of authentication by using the Digital Signatures. Each ARP reply is authenticated by the sender host by appending its own signature that ensures the ARP reply is from the correct host, and it is not an ARP spoof message.

The approach explained in this thesis simply consists of the following major steps.

1.7.1 Setting up secure LAN.

Assign a host which is out of LAN ; responsibility of certificate generator for each systems on LAN. The certificates are nothing but just the public key digital signature based on their IP/MAC address pair. The private key is kept safe and is know to the CAS (Certificate Authority System). This scenario seems to be little bit confusing in short, but is explained in detailed in later chapters.

1.7.2. Setting up the system environment.

The 2 major components of our executable programs are:

A. A loadable kernel module – The kernel module is responsible for filtering of the incoming packets. The existing kernel daemon is disabled to let new loadable kernel module to work in place of that. The new loadable kernel daemon filters out the ARP request or reply that is coming in or going out of the system, and passess those ARP request-reply messages to the user space daemon to process them according to Digital signature based ARP protocol.

B. User space daemon – The user space daemon process the incoming or outgoing ARP messages. Basically that deamon is responsible for appending the host's certificate to the outgoing ARP reply so that the receiver system can authenticate its reply, and verifying the incoming ARP reply messages by using the certificate appended in it and and applying the DSA (Digital Signature Algorithm).

1.8. Organization of the Thesis

The thesis is organized and written in very simple way that a novice user of LANcan easily understand the problem of ARP spoofing and its remedial measures. This thesis starts with the simple introduction of the ARP Protocol in chapter 2. In chapter 3, how the ARP spoofing messages are generated, what the effects of such messages on system or LAN, and what are the other proposed method for protection system against ARP spoofing are described. Same kind of problem in networks, DNS spoofing is described in chapter 4. Chapter 5 introduces the concept of Digital Signatures with an example of signature generation and verification. Approach for handling ARP based spoofing attacks proposed by this thesis is described in chapter 6. The latter chapters are Results, Conclusion and future work, Bibliography and References.

Chapter 2
Address Resolution Protocol (ARP)

2.1 Introduction

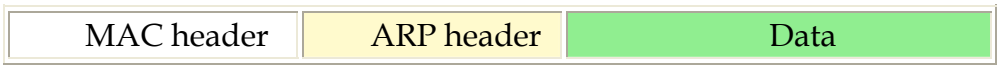
ARP is the Address Resolution Protocol as described in RFC 826. ARP is used by a networked machine to resolve the hardware location/address of another machine on the same local network. Machines on the Internet are generally known by their names which resolve to IP addresses. This is how a machine on the foo.com network is able to communicate with another machine which is on the bar.net network. An IP address, though, cannot tell you the physical location of a machine.

Address Resolution Protocol (ARP) hovers in the shadows of most networks. Because of its simplicity, by comparison to higher layer protocols, ARP rarely intrudes upon the network administrator's routine. All modern IP-capable operating systems provide support for ARP. The uncommon alternative to ARP is static link-layer-to-IP mappings.

ARP defines the exchanges between network interfaces connected to an Ethernet media segment in order to map an IP address to a link layer address on demand (ARP request). Link layer addresses are hardware addresses (although they are not immutable) on Ethernet cards and IP addresses are logical addresses assigned to machines attached to the Ethernet. Subsequently in this chapter, link layer addresses may be known by many different names: Ethernet addresses, Media Access Control (MAC) addresses, and even hardware addresses. Disputably, the correct term from the kernel's perspective is "link layer address" because this address can be changed (on many Ethernet cards) via command line tools. Nevertheless, these terms are not realistically distinct and can be used interchangeably.

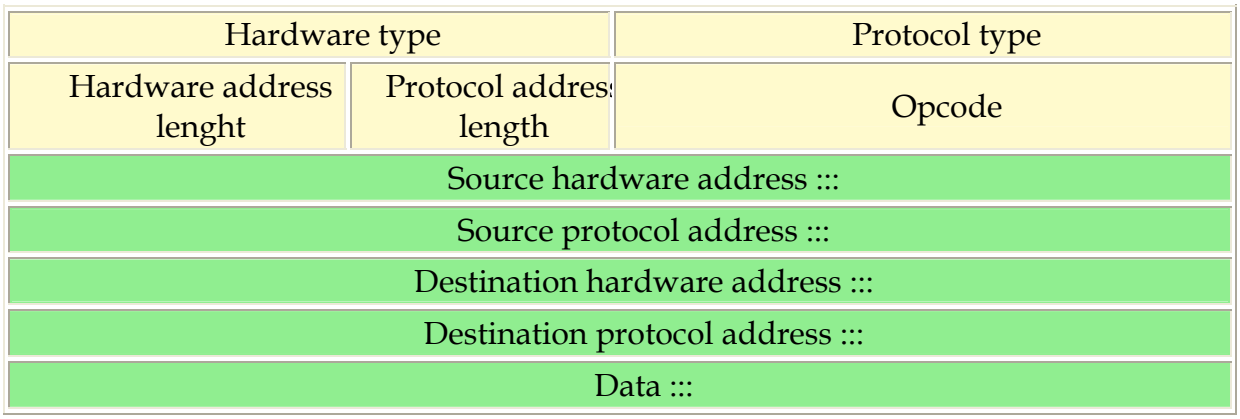
2.2 ARP Frame Format

The encapsation of ARP packet in Ethernet frame is shown below.



ARP header:

0 **15** **31**



Hardware type. 16 bits.

Table 2.1. Hardware type field values with description.

Value	Description
1	Ethernet.
2	Experimental Ethernet.
3	Amateur Radio AX.25.
4	Proteon ProNET Token Ring.
5	Chaos.
6	IEEE 802.
7	ARCNET.
8	Hyperchannel.
9	Lanstar.
10	Autonet Short Address.
11	LocalTalk.
12	LocalNet (IBM PCNet or SYTEK LocalNET).
13	Ultra link.

14	SMDS.
15	Frame Relay.
16	ATM, Asynchronous Transmission Mode.
17	HDLC.
18	Fibre Channel.
19	ATM, Asynchronous Transmission Mode.
20	Serial Line.
21	ATM, Asynchronous Transmission Mode.
22	MIL-STD-188-220.
23	Metricom.
24	IEEE 1394.1995.
25	MAPOS.
26	Twinaxial.
27	EUI-64.
28	HIPARP.
29	IP and ARP over ISO 7816-3.
30	ARPSec.
31	IPsec tunnel.
32	Infiniband.
33	CAI, TIA-102 Project 25 Common Air Interface.

Protocol type. 16 bits.

Value	Description
0x800	IP.

Hardware address length. 8 bits.
Length of the hardware address in bytes.

Protocol address length. 8 bits.
Length of the protocol address in bytes.

Opcode. 16 bits.

Table 2.2. Opcode field value references

Value	Description	References
1	Request.	RFC 826

2	Reply.	RFC 826, RFC 1868
3	Request Reverse.	RFC 903
4	Reply Reverse.	RFC 903
5	DRARP Request.	RFC 1931
6	DRARP Reply.	RFC 1931
7	DRARP Error.	RFC 1931
8	InARP Request.	RFC 1293
9	InARP Reply.	RFC 1293
10	ARP NAK.	RFC 1577
11	MARS Request.	
12	MARS Multi.	
13	MARS MServ.	
14	MARS Join.	
15	MARS Leave.	
16	MARS NAK.	
17	MARS Unserv.	
18	MARS SJoin.	
19	MARS SLeave.	
20	MARS Grouplist Request	
21	MARS Grouplist Reply.	
22	MARS Redirect Map.	
23	MAPOS UNARP.	

Source hardware address. Variable length.

Source protocol address. Variable length.

2.3 Overview of Address Resolution Protocol Reqest-Reply

Address Resolution Protocol (ARP) exists solely to glue together the IP and Ethernet networking layers. Since networking hardware such as switches, hubs, and bridges operate on Ethernet frames, they are unaware of the higher layer data carried by these frames. Similarly, IP layer devices, operating on IP packets need to be able to transmit their IP data on Ethernets. ARP defines the conversation by which IP capable hosts can exchange mappings of their Ethernet and IP addressing.

ARP is used to locate the Ethernet address associated with a desired IP address. When a machine has a packet bound for another IP on a locally connected Ethernet network, it will send a broadcast Ethernet frame containing an ARP request onto the Ethernet. All machines with the same Ethernet broadcast address will receive this packet. If a machine receives the ARP request and it hosts the IP requested, it will respond with the link layer address on which it will receive packets for that IP address

Once the requestor receives the response packet, it associates the MAC address and the IP address. This information is stored in the arp cache. The arp cache can be manipulated with the **ip neighbor** and **arp** commands. In Example 2.1, we used **ping** to test reachability of host `masq-gw .` Using a packet sniffer to capture the sequence of packets on the Ethernet as a result of host `tristan's` attempt to ping, provides an example of ARP request-reply conversation snapshot.

This is an archetypal conversation between two computers exchanging relevant hardware addressing in order that they can pass IP packets, and is comprised of two Ethernet frames.

Example 2.1. ARP conversation captured with tcpdump

```
[root@masq-gw]# tcpdump –ennqti eth0 \( arp or icmp \)
tcpdump: listening on eth0
0:80:c8:f8:4a:51 ff:ff:ff:ff:ff:ff 42: arp who-has 192.168.99.254 tell
192.168.99.35            ❶
0:80:c8:f8:5c:73 0:80:c8:f8:4a:51 60: arp reply 192.168.99.254 is-at
0:80:c8:f8:5c:73           ❷
0:80:c8:f8:4a:51 0:80:c8:f8:5c:73 98: 192.168.99.35 > 192.168.99.254:
icmp: echo request (DF)    ❸
0:80:c8:f8:5c:73 0:80:c8:f8:4a:51 98: 192.168.99.254 > 192.168.99.35:
icmp: echo reply           ❹
```

❶ This broadcast Ethernet frame, identifiable by the destination Ethernet

address with all bits set (ff:ff:ff:ff:ff:ff) contains an ARP request from `tristan` for IP address 192.168.99.254. The request includes the source link layer address and the IP address of the requestor, which provides enough information for the owner of the IP address to reply with its link layer address.

❷ The ARP reply from `masq-gw` includes its link layer address and declaration of ownership of the requested IP address. Note that the ARP reply is a unicast response to a broadcast request. The payload of the ARP reply contains the link layer address mapping.

The machine which initiated the ARP request (`tristan`) now has enough information to encapsulate an IP packet in an Ethernet frame and forward it to the link layer address of the recipient (00:80:c8:f8:5c:73).

❸❹ The final two packets in Example 2.1 display the link layer header and the encapsulated ICMP packets exchanged between these two hosts. Examining the ARP cache on each of these hosts would reveal entries on each host for the other host's link layer address.

This example is the commonest example of ARP traffic on an Ethernet. In summary, an ARP request is transmitted in a broadcast Ethernet frame. The ARP reply is a unicast response, containing the desired information, sent to the requestor's link layer address.

An even rarer usage of ARP is gratuitous ARP, where a machine announces its ownership of an IP address on a media segment. The **arping** utility can generate these gratuitous ARP frames. Linux kernels will respect gratuitous ARP frames. The following example shows the working of gratuitous ARP.

Example 2.2. Gratuitous ARP reply frames

```
[root@tristan]# arping -q -c 3 -A -I eth0 192.168.99.35
[root@masq-gw]# tcpdump -c 3 -nni eth2 arp
tcpdump: listening on eth2
06:02:50.626330   arp   reply   192.168.99.35   is-at   0:80:c8:f8:4a:51
(0:80:c8:f8:4a:51)
06:02:51.622727   arp   reply   192.168.99.35   is-at   0:80:c8:f8:4a:51
(0:80:c8:f8:4a:51)
06:02:52.620954   arp   reply   192.168.99.35   is-at   0:80:c8:f8:4a:51
(0:80:c8:f8:4a:51)
```

The frames generated in Example 2.2 are ARP replies to a question never asked. This sort of ARP is common in failover solutions and also for nefarious sorts of purposes, such as **ettercap**.

Unsolicited ARP request frames, on the other hand, are broadcast ARP requests initiated by a host owning an IP address.

Example 2.3. Unsolicited ARP request frames

```
[root@tristan]# arping -q -c 3 -U -I eth0 192.168.99.35
[root@masq-gw]# tcpdump -c 3 -nni eth2 arp
tcpdump: listening on eth2
06:28:23.172068  arp  who-has  192.168.99.35  (ff:ff:ff:ff:ff:ff)  tell
192.168.99.35
06:28:24.167290  arp  who-has  192.168.99.35  (ff:ff:ff:ff:ff:ff)  tell
192.168.99.35
06:28:25.167250  arp  who-has  192.168.99.35  (ff:ff:ff:ff:ff:ff)  tell
192.168.99.35
[root@masq-gw]# ip neigh show
```

These two uses of **arping** can help diagnose Ethernet and ARP problems--particularly hosts replying for addresses which do not belong to them.

To avoid IP address collisions on dynamic networks (where hosts are turning on and off, connecting and disconnecting and otherwise changing IP addresses) duplicate address detection becomes important. Fortunately, **arping** provides this functionality as well. A startup script could include the **arping** utility in duplicate address detection mode to select between IP addresses or methods of acquiring an IP address.

Example 2.4. Duplicate Address Detection with ARP

```
[root@tristan]# arping -D -I eth0 192.168.99.147; echo $?
ARPING 192.168.99.47 from 0.0.0.0 eth0
Unicast reply from 192.168.99.47 [00:80:C8:E8:1E:FC] for 192.168.99.47
[00:80:C8:E8:1E:FC] 0.702ms
Sent 1 probes (1 broadcast(s))
Received 1 response(s)
1
[root@tristan]# tcpdump -eqtnni eth2 arp
tcpdump: listening on eth2
0:80:c8:f8:4a:51 ff:ff:ff:ff:ff:ff 60: arp who-has 192.168.99.147
(ff:ff:ff:ff:ff:ff) tell 0.0.0.0
0:80:c8:e8:1e:fc 0:80:c8:f8:4a:51 42: arp reply 192.168.99.147 is-at
0:80:c8:e8:1e:fc (0:80:c8:e8:1e:fc)
[root@masq-gw]# ip neigh show
```

Address Resolution Protocol, which provides a method to connect physical network addresses with logical network addresses is a key element to the deployment of IP on Ethernet networks.

2.4 The ARP cache

In simplest terms, an ARP cache is a stored mapping of IP addresses with link layer addresses. An ARP cache obviates the need for an ARP request/reply conversation for each IP packet exchanged. Naturally, this efficiency comes with a price. Each host maintains its own ARP cache, which can become outdated when a host is replaced, or an IP address moves from one host to another. The ARP cache is also known as the neighbor table.

To display the ARP cache, the venerable and cross-platform **arp** admirably dispatches its duty. As with many of the **iproute2** tools, more information is available via **ip neighbor** than with **arp**. Example 2.5 below illustrates the differences in the output between the output of these two different tools.

Example 2.5. ARP cache listings with arp and ip neighbor

```
[root@tristan]# arp –na
? (192.168.99.7) at 00:80:C8:E8:1E:FC [ether] on eth0
? (192.168.99.254) at 00:80:C8:F8:5C:73 [ether] on eth0
[root@tristan]# ip neighbor show
192.168.99.7 dev eth0 lladdr 00:80:c8:e8:1e:fc nud reachable
192.168.99.254 dev eth0 lladdr 00:80:c8:f8:5c:73 nud reachable
```

A major difference between the information reported by **ip neighbor** and **arp** is the state of the proxy ARP table. The only way to list permanently advertised entries in the neighbor table (proxy ARP entries) is with the **arp**.

Entries in the ARP cache are periodically and automatically verified unless continually used. In Unix systems along with net/ipv4/neigh/$DEV/gc_stale_time, there are a number of other parameters in net/ipv4/neigh/$DEV which control the expiration of entries in the ARP cache.

When a host is down or disconnected from the Ethernet, there is a period of time during which other hosts may have an ARP cache entry for the disconnected host. Any other machine may display a neighbor table with the link layer address of the recently disconnected host. Because there is a recently known-good link layer address on which the IP was reachable, the entry will abide. At `gc_stale_time` the state of the entry will change, reflecting the need to verify the reachability of the link layer address. When the disconnected host fails to respond ARP requests, the neighbor table entry will be marked as `incomplete`

Here are a the possible states for entries in the neighbor table.

Table 2.3. Active ARP cache entry states

ARP cache entry state	Meaning	action if used
permanent	never expires; never verified	Reset use counter
noarp	normal expiration; never verified	Reset use counter
reachable	normal expiration	Reset use counter
stale	still usable; needs verification	reset use counter; change state to delay
delay	schedule ARP request; needs verification	Reset use counter
probe	sending ARP request	Reset use counter
incomplete	first ARP request sent	Send ARP request
failed	no response received	Send ARP request

To resume, a host (192.168.99.7) in `tristan`'s ARP cache on the above example has just been disconnected. There are a series of events which will occur as `tristan's` ARP cache entry for 192.168.99.7 expires and gets scheduled for verification. Imagine that the following commands are run to capture each of these states immediately before state change.

Example 2.6. ARP cache timeout

```
[root@tristan]# ip neighbor show 192.168.99.7
192.168.99.7 dev eth0 lladdr 00:80:c8:e8:1e:fc nud reachable     ❶
[root@tristan]# ip neighbor show 192.168.99.7
192.168.99.7 dev eth0 lladdr 00:80:c8:e8:1e:fc nud stale         ❷
```

```
[root@tristan]# ip neighbor show 192.168.99.7
192.168.99.7 dev eth0 lladdr 00:80:c8:e8:1e:fc nud delay          ❸
[root@tristan]# ip neighbor show 192.168.99.7
192.168.99.7 dev eth0 lladdr 00:80:c8:e8:1e:fc nud probe          ❹
[root@tristan]# ip neighbor show 192.168.99.7
192.168.99.7 dev eth0  nud incomplete                             ❺
```

❶ Before the entry has expired for 192.168.99.7, but after the host has been disconnected from the network. During this time, `tristan` will continue to send out Ethernet frames with the destination frame address set to the link layer address according to this entry.

❷ It has been `gc_stale_time` seconds since the entry has been verified, so the state has changed to stale.

❸ This entry in the neighbor table has been requested. Because the entry was in a stale state, the link layer address was used, but now the kernel needs to verify the accuracy of the address. The kernel will soon send an ARP request for the destination IP address.

❹ The kernel is actively performing address resolution for the entry. It will send a total of `ucast_solicit` frames to the last known link layer address to attempt to verify reachability of the address. Failing this, it will send `mcast_solicit` broadcast frames before altering the ARP cache state and returning an error to any higher layer services.

❺ After all attempts to reach the destination address have failed, the entry will appear in the neighbor table in this state.

The remaining neighbor table flags are visible when initial ARP requests are made. If no ARP cache entry exists for a requested destination IP, the kernel will generate `mcast_solicit` ARP requests until receiving an answer. During this discovery period, the ARP cache entry will be listed in an *incomplete* state. If the lookup does not succeed after the specified number of ARP requests, the ARP cache entry will be

listed in a *failed* state. If the lookup does succeed, the kernel enters the response into the ARP cache and resets the confirmation and update timers.

After receipt of a corresponding ARP reply, the kernel enters the response into the ARP cache and resets the confirmation and update timers.

For machines not using a static mapping for link layer and IP addresses, ARP provides on demand mappings.

Chapter 3
ARP Spoofing Attacks

3.1. How do ARP Spoofing-Based Attacks Work?

The key to ARP spoofing attacks lies in modifying the cached MAC and IP address pair information maintained by each system. The technique utilized to perform an ARP spoofing attack is sending false ARP broadcast notifications to devices on the locol network. These false ARP spoofing messages trick network devices into delivering network data to incorrect switch ports, allowing the attacker to have information destined for a victim system on the LAN sent to the attacker's port on the network device.

There are a variety of ways ARP spoofing can attack a network. One of the most effective and dangerous is the Man in the Middle, or MITM, attack. A MITM attack places the attacking system between the victim's system and the local gateway for egress traffic, allowing the attacking system to "sniff" everything the victim sends and receives.

first the attacker tricks the victim's system into incorrectly addressing Ethernet frames of its packets, and tricks the switch into sending the victim's data to the attacker's switch port. The attacker does this with a series of spoofed *ARP* messages, after which it is possible for the attacker to monitor the victim's egress connections. However, because the victim's system will not be receiving any data in response to its connection attempts all outbound network communications made by the victim system will fail.

In order to allow the victim to continue to send and receive data to remote systems, the attacker sends another set of false ARP broadcast frames. This set of forged messages incorrectly tells the victim's system that

information destined for remote systems should be sent to the attacker's MAC address rather than to the gateway. The attacker's system is now receiving the victim's network data in duplex; that is, both our bound and inbound traffic.

Now for the final trick of the **MITM:** The attacking system forward, the traffic to the machines for which it was originally intended, in effect becoming an temporary gateway between the victim's system and the true LAN gateway. The victim is still able to send and receive data to and from remote systems, bur all of this data is transported through the attacker's system.

The implications to security of being interposed between the victim and the gateway are severe. *Any* plain-text authentication information or communications, such as network passwords or e-mail, can be monitored by the attacker. Not only can attackers monitor network traffic, they can also modify it. Having the ability to edit the victim's network data on the fly allows the attacker to alter communications from the victim's system, perform MITM-based anacks during cryptographic key exchange, and carry out many other security compromises. For instance, by modifying traffic during the initial key exchanges of an encrypted Web session, the attacker can intercept and decrypt the victim's traffic, even though the session still appears to be encrypted.

The following diagram shows a typical **ARP** Spoofing scenario along with an example of the **ARP** spoofing messages that the attacker sends out to redirect traffic.

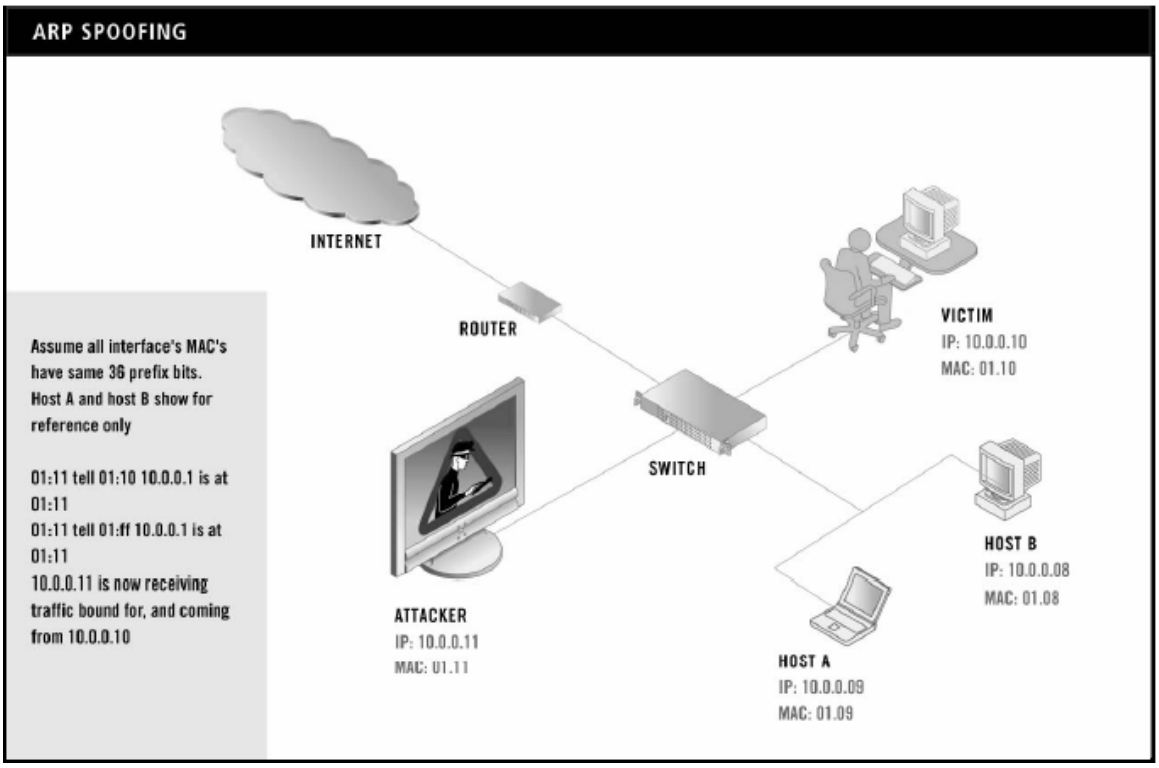


Figure 3.1. ARP Spoofing Scenario

There are many tools that will automatically perform most of the complex process, making ARP spoofing trivial for attacker. (See Appendix A for a list of softwares)

3.2. ARP Vulnerabilities.

Vulnerabilities are said to exist in ARP, when any system can spoof a reply to an ARP request and the system that will cache the reply would overwrite the existing entry and add the entry if one does not exist.

The OS that are vulnerable to ARP Spoofing are as follows

Windows 95/98/2000

Windows NT / XP

AIX 4.3

Linux

Netgear

Cisco IOS 11.1

The OS to protect against ARP spoofing is:

Sun Solaris systems

This appears to restrict cache poisoning; hence it makes the vulnerability of the OS to be much restricted.

3.3. ARP Attacks

The attacks are classified in to different types and they are:

1 Man in the Middle (MIM)

2 Denial of Service (DoS)

3 Hijacking

4 Cloning

5 Sniffing

3.3.1. Man in the Middle: A "man in the middle" attack is one of the type of attacks which is said to be performed when a malicious user inserts his computer between the communication path of two target computers. The malicious computer will forward frames between the two computers; so communications are not interrupted. The attack is performed as follows (where X is the attacking computer, and T1 and T2 are targets)

-host A poisons ARP cache of host B and host C.

- host B associates host c's IP with host A's MAC.

- host C associates host B's IP with host A's MAC.

-All of host B and host C's IP traffic will then go to host A first, instead of directly to each other.

This is extremely potent when we consider that not only can computer be poisoned, but routers/gateways as well. All Internet traffic for a host could be intercepted with this method by performing a MiM on a target computer and the LAN's router.

3.3.2. Denial of services: Updating ARP cache with non-existent MAC addresses will cause frames to be dropped i.e. because of limited size of ARP cache. These could be

sent out in a sweeping fashion to all clients on the network in order to cause a denial of service attack. This is also side effect of Post MiM attack, since targeted computers will continue to send frames to the attacker's MAC address even after the attacker had removed themselves from the communication path. To perform a clean MiM attack, the target computers would have to have the original ARP entries restored by the attacking computer.

3.3.3. Hijacking: Connection hijacking allows an attacker to take control of a connection between two computers using methods similar to the MiM attack. This transfer of control can result in any type of session being transferred. For example an attacker could take control of a telnet session after a target computer has logged in to a remote computer as an administrator.

3.3.4. Cloning: MAC addresses were intended to be a globally-unique identifier for each network interface produced. They were to be burned into the ROM of each interface and not to be changed but we are able to change the MAC address through software programs available, if we are changing it through means of hardware resources then it takes a lot more of time and work load. Linux users can even change their MAC without spoofing software, using a single parameter to "ifconfig", the interface configuration program for the OS. An attacker could *DoS* as a target computer, and then assigns them self the IP and MAC of the target computer, receiving all frames intended for the target computer.

3.3.5. Sniffing: Switches determine which frames go to which ports by comparing the destination MAC on the frame against a table. This table contains a list of ports and the attached MAC address. The table is built when the switch is powered on, by examining the source MAC from the first frame transmitted on each port. Network cards can enter state called "promiscuous mode" where they are allowed to examine frames that are destined for MAC address other than their own.

3.4. Detecting ARP spoofing attacks.

So now that we undersrand how ARP spoofing-based attacks work, how can we detect them? Due to the dynamic nature of the modern LAN and the automatic configuration of the ARP and **IP** pair information, detecting an ARP spoofing-based attack is difficult. The best approach is to enable software that monitors the ARP/IP pair combinations for machines on a given LAN. Such software can be configured to notify network or security administrators if any suspicious changes occur on the network, such as a broadcast ARP packet advertising a new MAC address for the LAN's gateway. Some operating systems will also log messages when changes occur in the gateway MAC address. Unfortunately many operating system, provide no built-in method to automatically monitor and log ARP modifications.

3.5. Protecting Against ARP Spoofing Attacks

The ARP spoofing attack is highly effective because it takes advantage of an inherent weakness in the design of a core network protocol. The only real solution an enhancement of the method of interface discovery for Ethernet networks. There is work being done in this area, bur implementation is years away.

There are some methods that network and security administrators can take to limit their exposure. The first is to implement software to monitor LAN segments for suspicious ARP traffic. This ensures that in the event of an attack the proper mitigation measures can take place. The second is, to segment your network. This can be accomplished by separating out subnets, using virtual LANs (VLANs) and router-based broadcast access control to limit the exposure of hosts to *ARP* spoofing-based attacks. A host that is on a separate subnet or isolated VLAN can not use an *ARP* spoofing-based attack against a host not on the same subnet or VLAN because the routing devices will drop the broadcast packets, thwarting the an attempted spoof. The third method is to hard-code the important MAC/IP pairs into mission critical machines, so that an attacker cannot modify them. Some network switch

devices allow static configuration of MAC/IP pair information for each port on the device; this prevents an attacker from tricking the switch into redirecting traffic to the wrong switch port.

Last and most important is to educate users about what ARP spoofing is, the telltale signs of being under an *ARP* attack, and to whom to respond. For instance, if any messages about mismatched keys during an encrypted communication such as HITPS or SSH occur, the communication must be terminated immediately and a network administrator notified.

Practically all these approaches are not appropriate for the protection against ARP spoofing due to the implementation implication in existing LAN technology. Monitoring ARP replies just monitor the ARP replies, it doesn't guarantee that the a particular massage is spoofing message. Hardwiring the IP address with MAC address in a mission critical problem would solve the problem but would create a problem while moving the system configuration or changing LAN. Changing IP address would not be that much easy. One need a solution that will match up with the architecture of existing LAN technologies while concerning all these spoofing issues.

Chapter 4
DNS Cache Poisoning

4.1. Introduction

The Internet would grind to a halt – would not be possible – without a Domain Name System (DNS). As you'll see in this chapter, the proper operation of DNS is fundamental to the maintenance and distribution of the addresses for the vast number of nodes around the globe. So it would be too much to hope for crackers (malicious hackers) to ignore DNS as they continuously look for new ways to circumvent your security.

There are several facets to DNS security. In this chapter we focus on one of the most dangerous types of attack – DNS cache poisoning. To provide a complete picture of this threat, we'll explore how DNS works, two ways crackers facilitate cache poisoning, what impact this type of attack can have on your organization, and steps you can take to protect your information assets.

4.2. What is DNS?

In the world of the Internet and TCP/IP, IP addresses are used to route packets from source to destination. A single IP address, for example 203.192.135.234, is not difficult to remember. But trying to learn or track thousands of these addresses, including which server/node is associated with each address, is a daunting task. So instead, we use domain names to refer to systems with which we want to communicate.

A real-world Internet domain name example is Google.com. When you enter the Google domain name into the address bar of your browser, the Google page appears. This is because your PC executed a process to resolve Google.com to an IP address. Only by having the IP address is a system able to initiate a session with

another system across the Internet. Let's look at two ways IP address resolution can occur.

Figure 4.1 depicts the domain name/IP address resolution process when the target system and DNS server are internal. In this example, a workstation must establish a session with a server with a domain name of *Farpoint.company.com.* In order for a workstation to implement DNS, it must be running a DNS Client or Client *Resolver*. The resolver initiates the following process, resulting in the conversion of the domain name to an IP address (Microsoft TechNet, 2005).

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.company.com. The entry would be present if the workstation had resolved the name to an IP address since the last time it was powered on, and the Time to Live of the entry had not been exceeded. In this example, no entry is found.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution query to the internal DNS server.

Step 3: When the DNS server receives the query, it first checks to see if it's authoritative for the company.com domain. In other words, is it responsible for managing the zone in which the company.com domain resides? If it is, the server performs a lookup in its internal zone table. In this case, it finds a host Resource Record (RR) that includes the IP address for Farpoint.company.com.

Step 4: The IP address of Farpoint.company.com is returned to the resolver.

Step 5: The resolved domain name and IP address are placed into the resolver cache. Figure 4.2 is an actual listing of the contents of a workstation's resolver cache. The IP address is used to contact Farpoint.

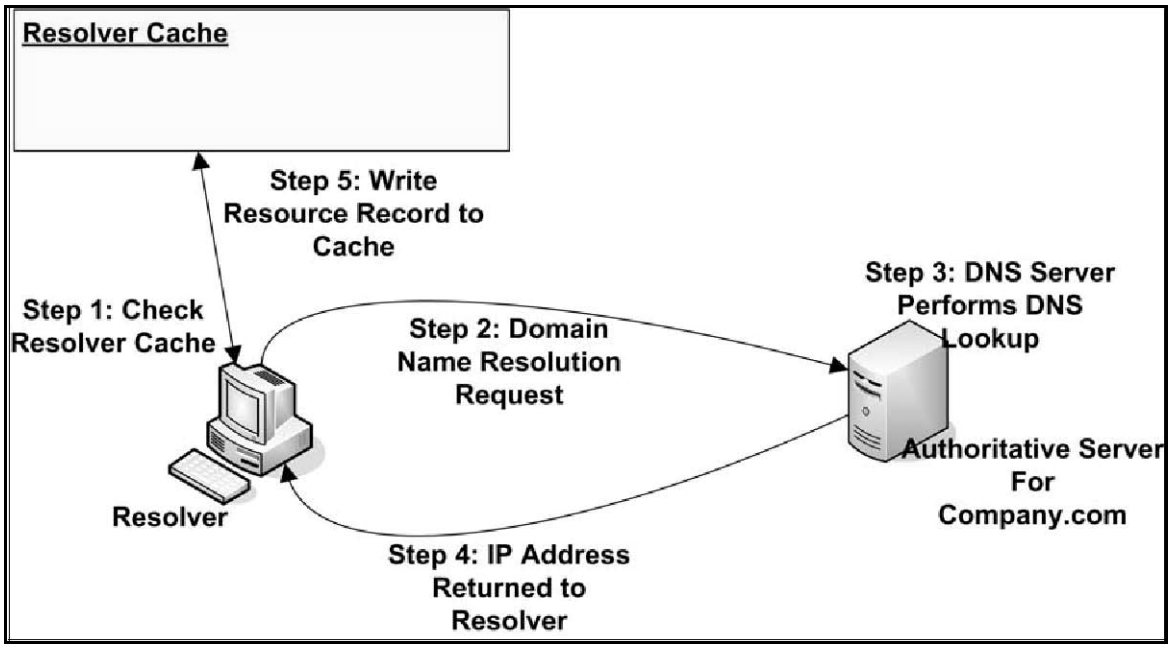


Figure 4.1: Internal DNS Server Lookup

In the previous example, the target server was located within the requestor's network. But there are many instances in which the target device is located somewhere on the Internet. In these cases, the process is somewhat different. Please refer to Figure 4.3 as we step through this second DNS resolution process.

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.companyA.com.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution request to the internal DNS server.

Step 3: When the DNS server receives the request, it first checks to see if it's authoritative. In this case, it isn't authoritative for companyA.com. The next action it takes is to check its local cache to see if an entry for Farpoint.companyA.com exists. It doesn't. So in Step 4 the internal DNS server begins the process of iteratively querying external DNS servers until it

either resolves the domain name or it reaches a point at which it's clear that the domain name entry doesn't exist.

Figure 4.2: Actual Resolver Cache Listing

Step 4: A request is sent to one of the Internet "root" servers . The root server returns the address of a server authoritative for the .COM Internet

space.

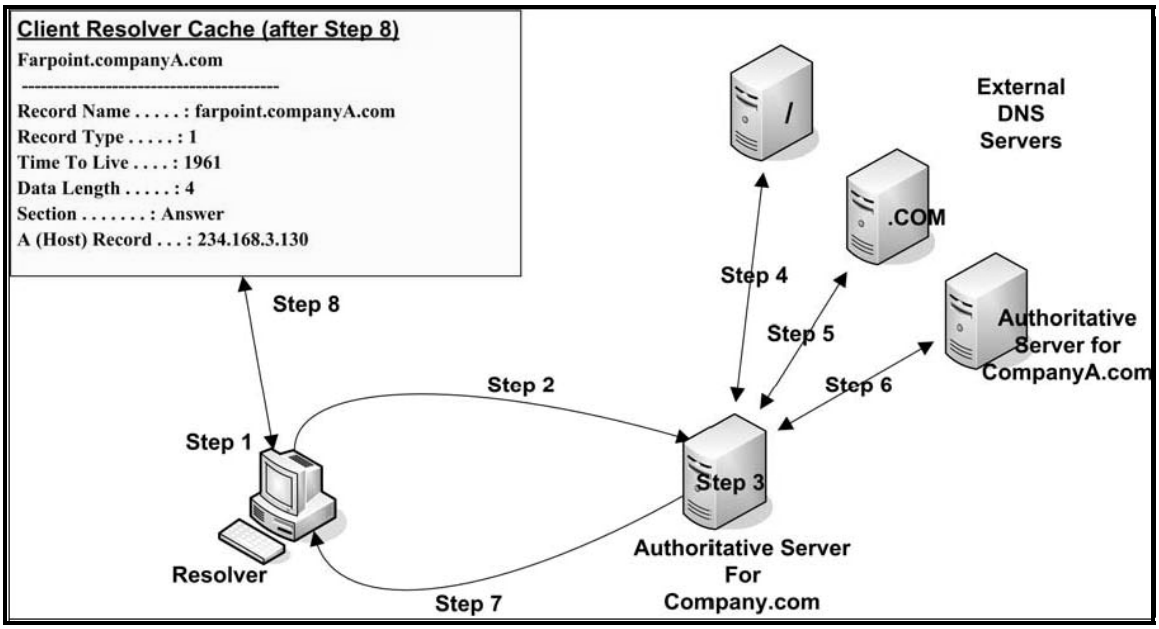


Figure 4.3: Recursive DNS Query

Step 5: A request is sent to the authoritative server for .COM. The address of a DNS server authoritative for the companyA.com domain is returned.

Step 6: A request is sent to the authoritative server for companyA.com. The IP address of Farpoint.companyA.com is returned.

Step 7: The IP address for Farpoint is returned to the client resolver.

Step 8: An entry is made in the resolver cache, and a session is initiated with Farpoint.companyA.com.

This process, from the client resolver perspective, is known as a recursive query. With the proper software, the client resolver could perform the iterative query illustrated by Steps 4, 5, and 6.

Now that we have a good idea how DNS is supposed to work, it's time to look at how this process can be used to co-opt one or more DNS caches.

4.3. What is DNS Cache Poisoning?

DNS cache poisoning consists of changing or adding records in the resolver caches, either on the client or the server, so that a DNS query for a domain returns an IP address for an attacker's domain instead of the intended domain. To demonstrate how this might work, let's step through Figure 4.4.

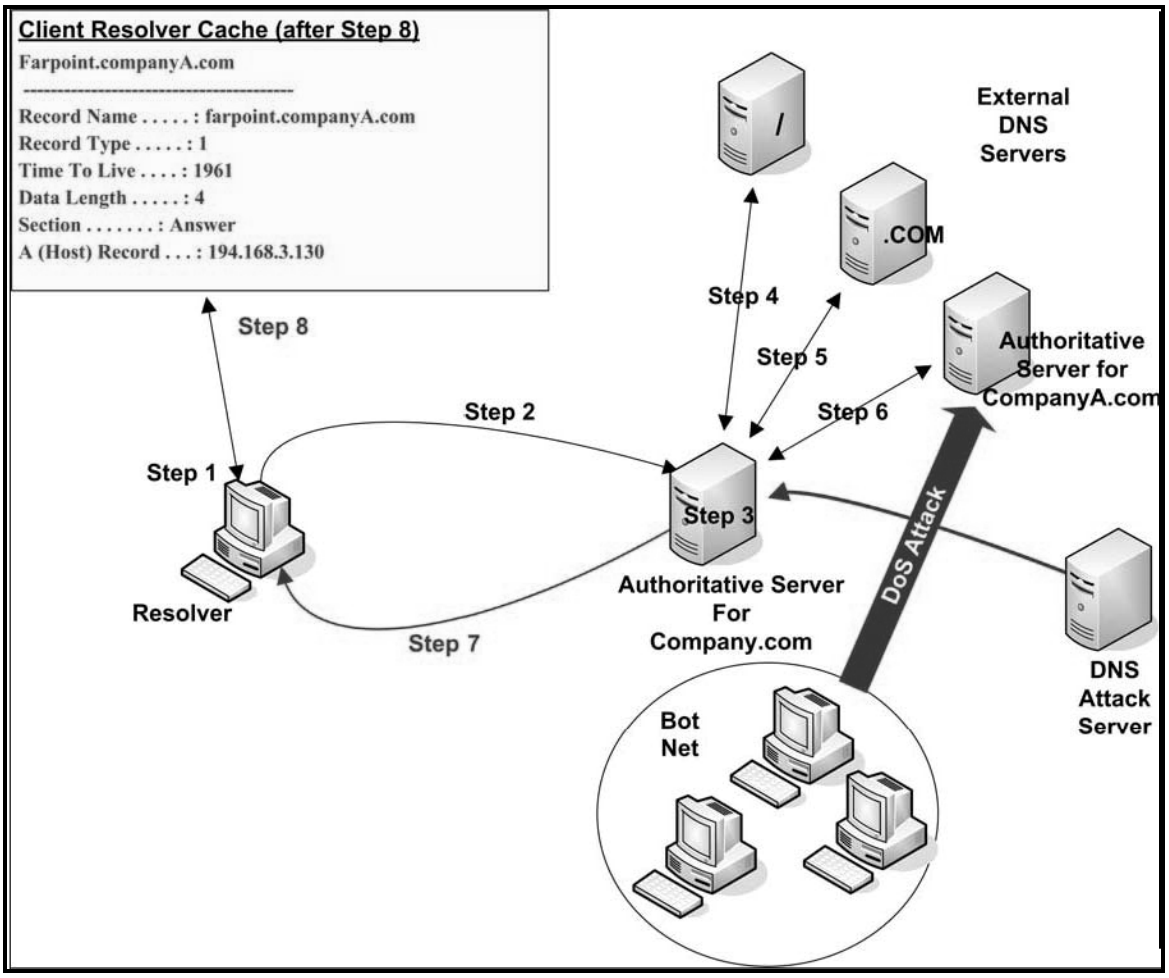


Figure 4.4: DNS Cache Poisoning

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.companyA.com.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution request to the internal DNS server.

Step 3: When the DNS server receives the request, it first checks to see if it's authoritative. In this case, it isn't authoritative for companyA.com. The next action it takes is to check its local cache to see if an entry for Farpoint.companyA.com exists. It doesn't. So in Step 4 the internal DNS server begins the process of iteratively querying external DNS servers until it either resolves the domain name or it reaches a point at which it's clear that the domain name entry doesn't exist.

Step 4: A request is sent to one of the Internet root servers. The root server returns the address of a server authoritative for the .COM Internet space.

Step 5: A request is sent to the authoritative server for .COM. The address of a DNS server authoritative for the companyA.com domain is returned.

Step 6: A request is sent to the authoritative server for companyA.com. This is identical to the standard process for an iterative query – with one exception. A cracker has decided to poison the internal DNS server's cache. In order to intercept a query and return malicious information, the cracker must know the transaction ID. Once the transaction ID is known, the attacker's DNS server can respond as the authoritative server for companyA.com.

Although this would be a simple matter with older DNS software (e.g. BIND 4 and earlier), newer DNS systems have build-in safeguards. In our example, the transaction ID used to identify each query instance is randomized. But figuring out the transaction ID is not impossible. All that's required is time. To slow the response of the real authoritative server, our cracker uses a botnet _to initiate a Denial of Service (DoS) attack. While the authoritative server struggles to deal with the attack, the attacker's DNS server has time to determine the transaction ID.

Once the ID is determined, a query response is sent to the internal

DNS server. But the IP address for Farpoint.companyA.com in the response is actually the IP address of the attacker's site. The response is placed into the server's cache.

Step 7: The rogue IP address for Farpoint is returned to the client resolver.

Step 8: An entry is made in the resolver cache, and a session is initiated with the attacker's site. At this point, both the workstation's cache and the internal DNS server's cache are poisoned. Any workstation on the internal network requesting resolution of Farpoint.companyA.com will receive the rogue address listed in the internal DNS server's cache. This continues until the entry is deleted.

Another method used to poison a DNS cache is the use of a recursive query sent by the attacker. The query can force the target server to connect to the authoritative source of the domain in the query. Once connected, rogue information about one or more domains might be sent to the querying server and posted to the server's cache. There are other methods attackers use to poison DNS caches, but the objective is the same. Now we'll explore the consequences of using a poisoned DNS cache.

4.4. Potential Consequences of Cache Poisoning

Pharming is the primary risk associated with cache poisoning. Crackers employ pharming for four primary reasons (Hyatt, 2006): identity theft, distribution of malware, dissemination of false information, and man-in-the-middle attacks.

4.4.1. Identity Theft

Once an attacker gets you to his site, he'll try to trick you into leaving behind information he can use to impersonate you. One way to do this in our first example is to create a site identical to the real Farpoint.companyA.com. When the user connects using the poisoned cache information, she might be fooled into entering

information about herself through apparently legitimate requests for her name, social security number, address, etc.

4.4.2. Distribution of Malware

Another of objective of attackers using cache poisoning is the automatic distribution of malware. Instead of releasing malicious code into the Internet and realizing random results, the use of rogue IP addresses to redirect unsuspecting users to the attacker's site can be a more focused attack vector. Once a workstation initiates a session with the malicious site, malware is uploaded to the workstation without intervention by or the knowledge of the user.

4.4.3. Dissemination of False Information

This aspect of pharming is useful to attackers who want to spread self-serving information about an organization. It's also been used to manipulate stock prices in an attempt to realize a large profit.

4.4.4. Man-in-the-middle Attack

In this attack type, the workstation initiates a session with the attacker's server. The attacker's server initiates a session with the actual target site. All information flowing between the workstation and the genuine site passes through and is intercepted by the cracker's server.

There can be serious consequences when security is an afterthought during the configuration and deployment of DNS servers. The next section provides guidelines that can help prevent cache poisoning.

4.5. Protecting Your Assets

The first layer of defense against cache poisoning is the use of the latest version of DNS. DNS based on BIND 9.3.x or Microsoft Windows Server 2003 is far more secure than DNS implemented with earlier versions. Successful completion of our

first poisoning example would have been more difficult, because these systems also randomize the port used for the DNS query in addition to the transaction ID.

Recursive queries should be limited to internal DNS servers. If Internet facing recursive queries are required, only queries from internal addresses should be accepted. This will help prevent outside systems from sending queries with malicious intent. The following list provides additional guidance (Hyatt, 2005):

- ▲ Physically separate external and internal DNS servers
- ▲ Restrict zone transfers to authorized (secondary servers) devices
- ▲ Restrict dynamic DNS updates when possible
- ▲ Hide the version of BIND being used on the DNS servers
- ▲ Remove unnecessary services running on the DNS servers
- ▲ Where possible, use dedicated appliances instead of multi-purpose servers

Chapter 5
Digital Signatures

5.1 Introduction

The authentication of computer-based business information interrelates both technology and the law, and calls for cooperation between people of different professional backgrounds and areas of expertise. Each field of expertise brings to the topic of authentication a different repertoire of concepts. Often the concepts from the information security field correspond only loosely to concepts from the legal field, even though both fields apply the same term to their differing concepts.

This interdisciplinary contrast exists even for basic, central concepts such as "authentication" or "digital signature". From a technical point of view, "digital signature" means the result of applying to specific information the technical processes described below. From a legal point of view, handwriting one's name on paper has been the principal means of signature for centuries. In addition, the legal concept of signature recognizes, in many cases, not only a handwritten name but any mark made with the intention of authenticating the marked document. In an electronic setting, today's broad legal concept of "signature" may well include markings such as digitized images of paper signatures, typed notations such as "John Smith", or even addressing notations such as letterheads, electronic mail origination headers, and the like. From an information security viewpoint, these simple electronic signatures are entirely different from the "digital signatures" described in this chapter and in technical documents, although "digital signature" is sometimes used colloquially or in some legal writing to mean another or any form of computer-based signature. To avoid confusion, this publication uses "digital signature" only in the sense in which the term is used in information security terminology, as meaning the result of applying the technical processes described in this chapter.

The differences between digital signatures and other electronic signatures are significant, not only in terms of process and result, but also because those differences make digital signatures more serviceable for legal purposes. However, some electronic signatures, though perhaps legally recognizable as signatures, may not be as secure as digital signatures, and may lead to uncertainty and disputes.

To understand why digital signatures serve well in legal applications, this chapter begins with an overview of the significance of signatures in legal transactions. It then explains digital signature technology in simple terms, and examines how, with some legal and institutional infrastructure, digital signature technology can be applied as a computer-based alternative to traditional signatures for authentication. **The concept of authentication by using the digital signatures can be extended to authenticate the ARP request reply messeges.**

5.2 Signatures and the Law

A signature is not part of the substance of a transaction, but rather of its representation or form. Parties often represent their transactions in signed writings. Signing writings and other formalistic legal processes or customs serve the following general purposes:

- **Evidence**: A signature identifies the signer with the signed document; by signing, the signer marks the text in her own unique way and makes it attributable to her.
- **Ceremony**: Signing calls to the signer's attention the legal significance of his act, and thereby helps prevent "inconsiderate engagements". The act of signing may satisfy a human desire to mark an event.
- **Approval**: In certain contexts defined by law or custom, a signature expresses the signer's approval or authorization of the writing, or the signer's intention that it have legal effect.

- **Efficiency and logistics**: A signature on a written memorandum often imparts a sense of clarity and finality to the transaction, especially if the signature is used to indicate approval or authorization. Because of this apparent clarity and finality, signatures may lessen the need to inquire beyond the face of a document,and at face value, a document may be processed more efficiently and with less risk than a document beneath which traps for the unwary may lie. Negotiable instruments, for example, attain their ability to change hands with ease, rapidity, and minimal interruption through legal rules triggered by compliance with certain formal requirements including a signature. Furthermore, the finality of signing makes it useful as a decisive point in staging how a transaction takes effect.

Although achieving these purposes is salutary, legal systems vary, both among themselves and over time, in the degree to which a particular form, including one or more signatures, is required for a legal transaction. If a particular form is required, legal systems also vary in prescribing consequences for failure to cast the transaction in the required form. The statute of frauds of the common law tradition, for example, requires a signature, but does not render a transaction invalid for lack of one. Rather, it makes it unenforceable in court, and the persistent notion that the underlying transaction remained valid led case law to greatly limit the practical application of the statute.

In general, the trend in most legal systems for at least this century has been toward reducing formal requirements in law, or toward minimizing the consequences of failure to satisfy formal requirements. Nevertheless, sound practice remains to formalize a transaction in a manner that best assures the parties of its validity and enforceability. In current practice, that formalization usually entails documenting the transaction and signing or authenticating the documentation.

However, the centuries-old means of documenting transactions and creating signatures are changing fundamentally. Documents continue to be written on paper, but sometimes merely to satisfy the need for a legally recognized form. In many instances, the information exchanged to effect a transaction never takes paper form. It also no longer moves as paper does; it is not physically carried from place to place but rather streams along digital conduits at a speed impossible for paper. The computer-based information is also utilized differently than its paper counterpart. Paper documents can be read efficiently only by human eyes, but computers can also read digital information and take programmable actions based on the information.

The law has only begun to adapt to the new technological forms. The basic nature of the transaction has not changed; however, the transaction's form, the means by which it is represented and effected, is changing. Formal requirements in law need to be updated accordingly. The legal and business communities need to develop and adopt rules and practices which recognize in the new, computer-based technology the effects achieved or desired from the paper forms.

To achieve the basic purposes of signatures outlined above, the following effects are needed:

- **Signer authentication**: To provide good evidence of who participated in a transaction, a signature should indicate by whom a document or message is signed and be difficult for any other person to produce without authorization.
- **Document authentication**: To provide good evidence of the substance of the transaction, a signature should identify what is signed, and make it impracticable to falsify or alter, without detection, either the signed matter or the signature.

- **Affirmative act**: To serve the ceremonial and approval functions of a signature, a person should be able to create a signature to mark an event, indicate approval and authorization, and establish the sense of having legally consummated a transaction.
- **Efficiency**: Optimally, a signature and its creation and verification processes should provide the greatest possible assurance of authenticity and validity with the least possible expenditure of resources.

The concepts of signer authentication and document authentication comprise what is often called "nonrepudiation service" in technical documents. The nonrepudiation service of information security "provides proof of the origin or delivery of data in order to protect the sender against false denial by the recipient that the data has been received, or to protect the recipient against false denial by the sender that the data has been sent." In other words, a nonrepudiation service provides evidence to prevent a person from unilaterally modifying or terminating her legal obligations arising out of a transaction effected by computer-based means.

Digital signature technology generally surpasses paper technology in yielding these desired effects. To understand why, one must first understand how digital signature technology works.

5.3 How Digital Signature Technology Works

Digital signatures are created and verified by means of cryptography, the branch of applied mathematics that concerns itself with transforming messages into seemingly unintelligible forms and back again. For digital signatures, two different keys are generally used, one for creating a digital signature or transforming data into a seemingly unintelligible form, and another key for verifying a digital signature or returning the message to its original form. Computer equipment and software utilizing two such keys is often termed an "**asymmetric cryptosystem**".

The keys of an asymmetric cryptosystem for digital signatures are termed the **private key**, which is known only to the signer and used to create the digital signature, and the **public key**, which is ordinarily more widely known and is used to verify the digital signature. A recipient must have the corresponding public key in order to verify that a digital signature is the signer's. If many people need to verify the signer's digital signatures, the public key must be distributed to all of them, perhaps by publication in an on-line repository or directory where they can easily obtain it.

Although the keys of the pair are mathematically related, it is "Computational infeasibility:deriving private key from public" computationally infeasible to derive one key from the other, if the asymmetric cryptosystem has been designed and implemented securely for digital signatures. Although many people will know the public key of a given signer and use it to verify that signer's signatures, they cannot discover that signer's private key and use it to forge digital signatures.

Use of digital signatures is comprised of two processes, one performed by the signer and the other by the receiver of the digital signature:

- **Digital signature creation** is the process of the computing a code derived from and unique to both the signed message and a given private key. For that code or digital signature to be secure, there must be at most only a negligible chance that the same digital signature could be created by any other message or private key.
- **Digital signature verification** is the process of checking the digital signature by reference to the original message and a public key, and thereby determining whether the digital signature was created for that same message using the private key that corresponds to the referenced public key.

A more fundamental process, termed a "**hash function**" in computer jargon, is used in both creating and verifying a digital signature. A hash function creates in effect a digital freeze frame of the message, a code usually much smaller than the message but nevertheless unique to it. If the message changes, the hash result of the message will invariably be different. Hash functions enable the software for creating digital signatures to operate on smaller and predictable amounts of data, while still providing a strong evidentiary correlation to the original message content.

As illustrated in Figure 5.1, to sign a document or any other item of information, the signer first delimits precisely what is to be signed. The delimited information to be signed is termed the "**message**". Then a hash function (Secure Hash Algorithm-1) in the signer's software computes a hash result (Message Digest), a code unique to the message. The signer's software then transforms the hash result (Message Digest) into a digital signature by reference to the signer's private key. This transformation is sometimes described as "encryption". The resulting digital signature is thus unique to both the message and the private key used to create it.

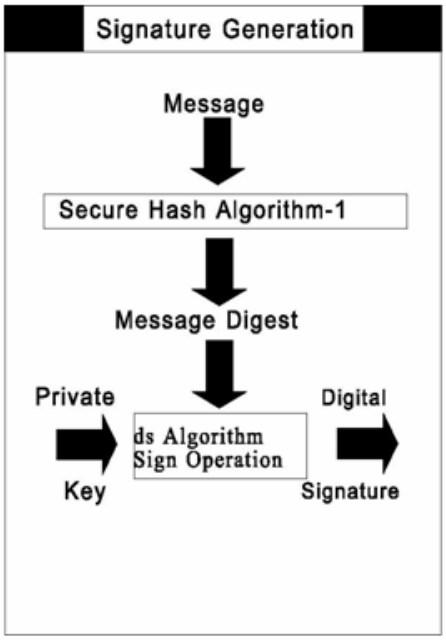


Figure 5.1. Signature Generation Alogorithm.

Typically, a digital signature is attached to its message and stored or transmitted with its message. However, it may also be sent or stored as a separate data element, so long as it maintains a reliable association with its message. Since a digital signature is unique to its message, it is useless if wholly dissociated from its message.

Verification of a digital signature, as illustrated in Figure 5.2, is accomplished by computing a new hash result of the original message by means of the same hash function used in creating the digital signature. Then, using the public key, the verifier checks whether the digital signature was created using the corresponding private key, and whether the newly computed hash result matches the hash result derived from the digital signature. If the signer's private key was used and the hash results are identical, then the digital signature is verified.

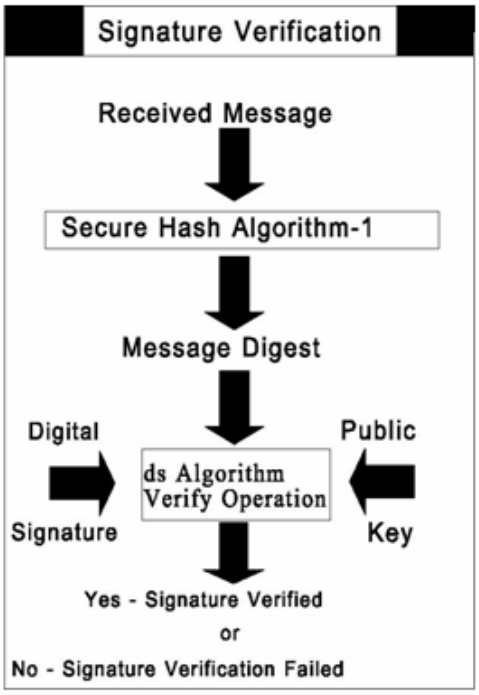


Figure 5.2. Signature verification algorithm

Verification thus indicates (1) that the digital signature was created using the signer's private key, because only the signer's public key will verify a digital signature created with the signer's private key, and (2) that the message was not altered since it was signed, because the hash result computed in verification matches the hash result from the digital signature, which was computed when the message was digitally signed.

Various asymmetric cryptosystems create and verify digital signatures using different mathematical formulas and procedures, but all share this overall operational pattern.

The processes of creating a digital signature and verifying it accomplish the essential effects desired of a signature:

- **Signer authentication**: If a public and private key pair is associated with an identified signer as described below, a digital signature by the private key effectively identifies the signer with the message. The digital signature cannot be forged by a person other than the proper signer, unless the proper signer loses control of the private key, such as by divulging it or losing a computer-readable card and its associated personal identification number (PIN) or pass phrase.
- **Message authentication**: The process of digitally signing also identifies the matter to be signed, typically with far greater certainty and precision than paper signatures. Verification also reveals any tampering with the message, since processing the hash results (one made at signing and the other made at verifying) discloses whether the message is the same as when signed.
- **Affirmative act**: Creating a digital signature requires the signer to provide her private key and invoke a software function to create a digital signature. This act can be the basis of a ceremony and can be used in staging the completion of a transaction.

- **Efficiency**: The processes of creating and verifying a digital signature provide a high level of assurance that the digital signature is genuinely the signer's and are almost entirely automated or capable of automation. They can be set up to run with great speed and accuracy, with human interaction only for non-routine processing decisions. Compared to paper methods such as checking bank signature cards, methods so impracticable that they are rarely actually used, digital signatures yield a high degree of assurance without adding greatly to the resources required for processing.

The core of the programs used for digital signatures have undergone thorough peer review, and an extensive scientific and technical literature underlies them. Digital signatures have been accepted in several national and international standards developed in cooperation with and accepted by many corporations, banks, and government agencies. The likelihood of malfunction or a security problem in a digital signature cryptosystem designed and implemented as prescribed in the industry standards is extremely remote, and far less than the risk of undetected forgery or alteration on paper or of using other less secure electronic signature techniques.

5.4. Public Key Certificates

To verify a digital signature, the verifier must obtain a public key and have assurance that that public key corresponds to the signer's private key. However, a public and private key pair has no intrinsic association with any person; it is simply a pair of numbers. The association between a particular person and key pair must be made by people using the fact-finding capabilities of their senses.

In a transaction involving two parties, for example, the parties could bilaterally identify each other with the key pair each party will use, but making such an identification is no small task, especially when the parties are geographically distant from each other, communicate over an open, insecure information network, are not

natural persons but rather corporations or similar artificial entities, and act through agents whose authority must be ascertained. Since reliably identifying a remote party involves considerable effort, establishing a remote party's digital signature capability specially for each of many transactions is inefficient. Instead, a prospective digital signer will often wish to identify itself with a key pair and reuse that identification in multiple transactions over a period of time.

To that end, a prospective signer could issue a statement such as: "Signatures verifiable by the following public key are mine". However, others doing business with the signer may well be unwilling to take the signer's own purported word for its identification with the key pair. Especially for electronic transactions made over worldwide information networks rather than face to face, a party would run a great risk of dealing with a phantom or an impostor, or of facing a disavowal of a digital signature by claiming it to be the work of an impostor, particularly if a transaction proves disadvantageous for the purported signer. To assure that each party is indeed identified with a particular key pair, one or more third parties trusted by both of the others must associate an identified person on one end of the transaction with the key pair creating the digital signature received at the other end, and vice versa. That trusted third party is termed a "**certification authority**" in the ABA Guidelines, the Utah Act, and most technical standards.

To associate a key pair with a prospective signer, a certification authority issues a certificate, an electronic record that sets forth a public key and represents that the prospective signer identified in the certificate holds the corresponding private key. That prospective signer is termed the "subscriber". Thus, a certificate's principal function is to identify a key pair with a subscriber, so that a person verifying a digital signature by the public key listed in the certificate can have assurance that the corresponding private key is held by the subscriber also listed in the certificate.

To assure the authenticity and inviolability of the certificate, the certification authority digitally signs it. The issuing certification authority's digital signature on the certificate can be verified using the public key listed in another certificate, and that other certificate can be verified by the public key listed in yet another certificate, and so on, until the person relying on the digital signature is adequately assured of its genuineness.

To make a public key and its identification with a specific subscriber readily available for use in verification, the certificate may be published in a repository. Repositories are on-line databases of certificates available for retrieval and use in verifying digital signatures. Often, retrieval is accomplished automatically by having the verification program inquire of the repository to obtain certificates as needed.

Once issued, a certificate may prove to be unreliable, such as in situations where the subscriber misrepresents his identity to the certification authority. In other situations, a certificate may be reliable enough when issued but come to be unreliable sometime thereafter. For example, if the subscriber loses control of the private key, the certificate becomes unreliable, since digital signatures created by the lost private key would appear to be the subscriber's according to the certificate. In such situations where the certificate has become unreliable, the certification authority, perhaps at the subscriber's request, may suspend (temporarily invalidate) or revoke (permanently invalidate) the certificate. Immediately upon suspending or revoking a certificate, the certification authority must publish notice of the revocation or suspension, or at least notify persons who inquire or who are known to have received a digital signature verifiable by reference to the unreliable certificate.

5.5 Authenticated Transaction Illustration

Consider the scenario described below.

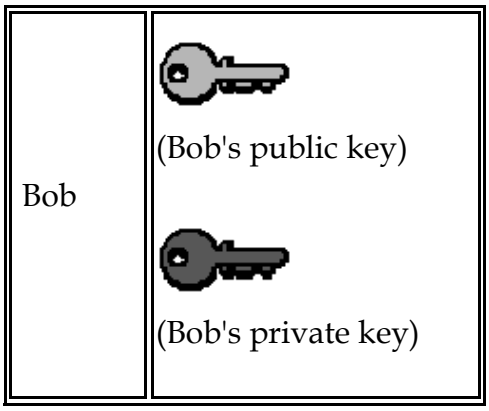


Figure 5.3. example- key pair

Bob has been given two keys. One of Bob's keys is called a Public Key, the other is called a Private Key.

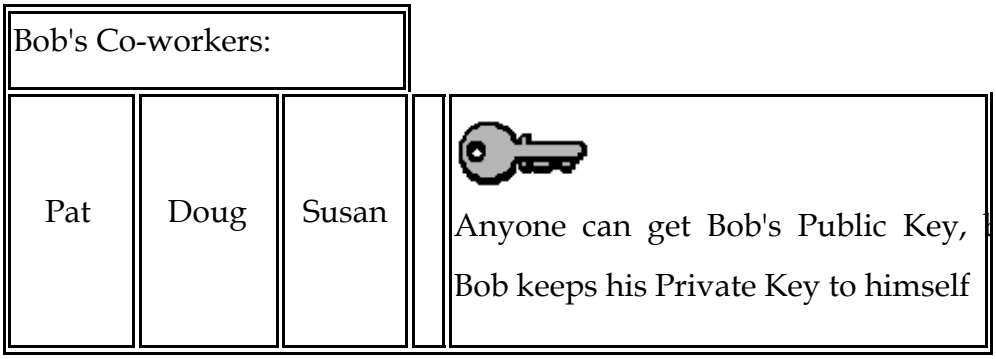


Figure 5.4. example-other persons

Bob's Public key is available to anyone who needs it, but he keeps his Private Key to himself. Keys are used to encrypt information. Encrypting information means "scrambling it up", so that only a person with the appropriate key can make it readable again. Either one of Bob's two keys can encrypt data, and the other key can decrypt that data.

Susan (shown below) can encrypt a message using Bob's Public Key. Bob uses his Private Key to decrypt the message. Any of Bob's coworkers might have access to the message Susan encrypted, but without Bob's Private Key, the data is worthless.

Susan	"Hey Bob, how about lunch at Taco Bell. I hear they have free refills!"		HNFmsEm6Un BejhhyCGKOK JUxhiygSBCEiC 0QYIh/Hn3xgiK BcyLK1UcYiY lxx2lCFHDC/A
Doug	HNFmsEm6Un BejhhyCGKOK JUxhiygSBCEiC 0QYIh/Hn3xgiK BcyLK1UcYiY lxx2lCFHDC/A		"Hey Bob, how about lunch at Taco Bell. I hear they have free refills!"

Figure 5.6. example-encryption - decryption

With his private key and the right software, Bob can put digital signatures on documents and other data. A digital signature is a "stamp" Bob places on the data which is unique to Bob, and is very difficult to forge. In addition, the signature assures that any changes made to the data that has been signed can not go undetected.

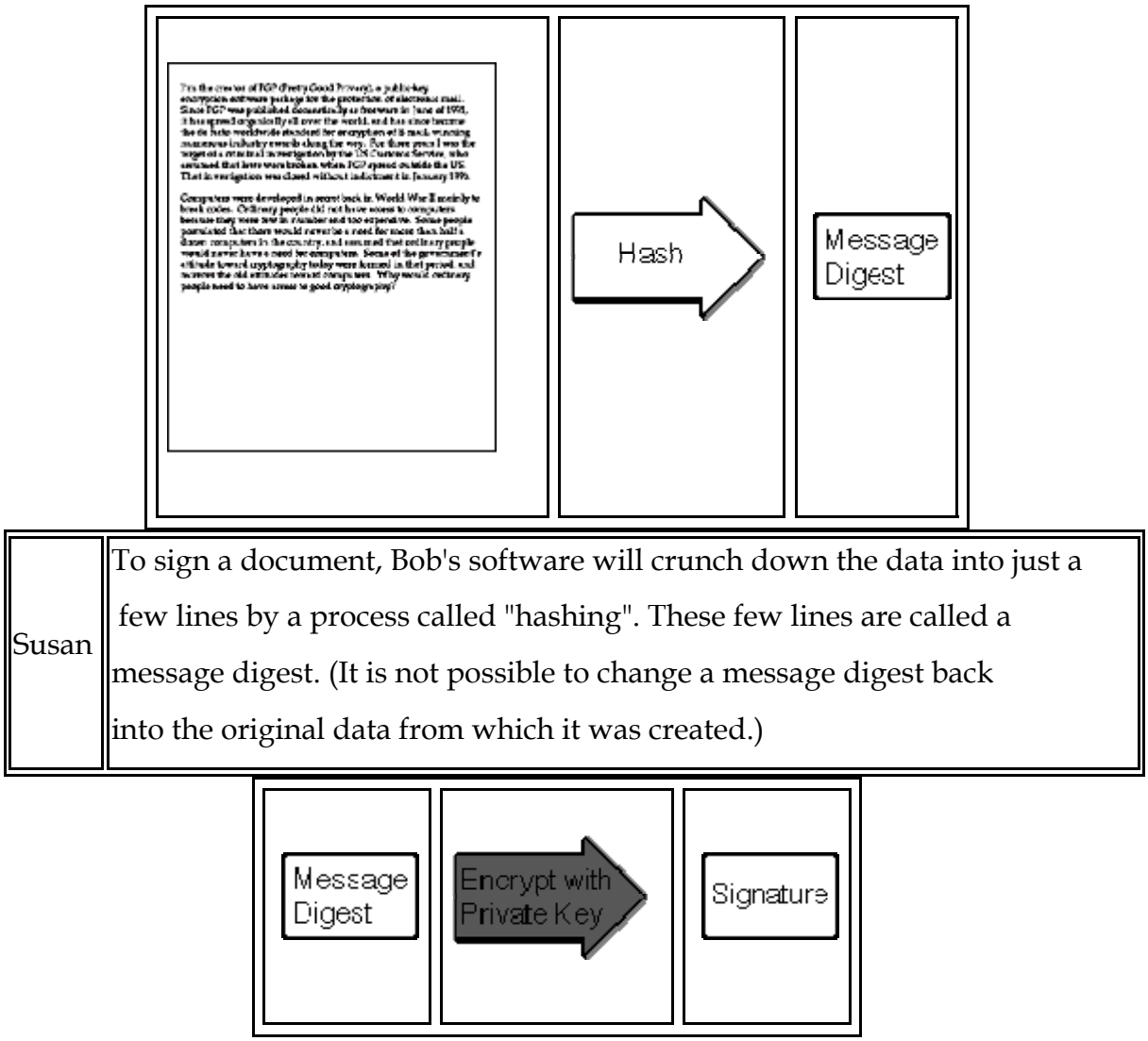


Figure 5.7. Example- message hashing and signature generation

Bob's software then encrypts the message digest with his private key. The result is the digital signature.

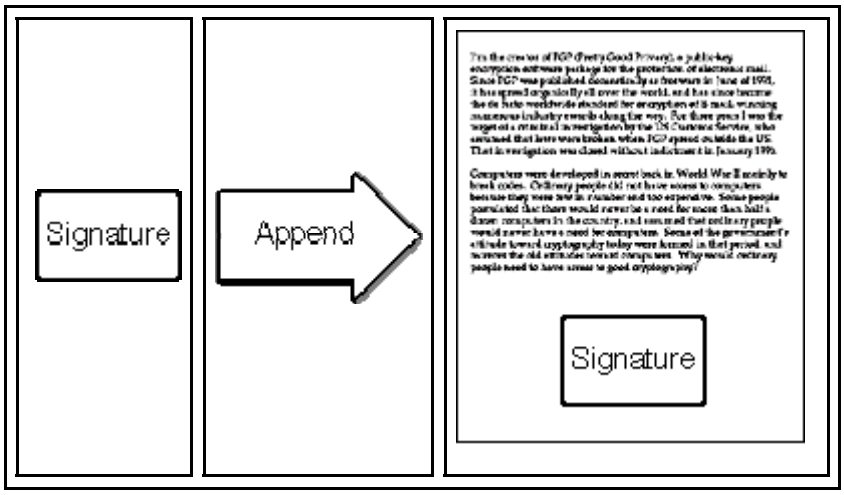


Figure 5.8. Example- Authenticated message

Finally, Bob's software appends the digital signature to document. All of the data that was hashed has been signed.

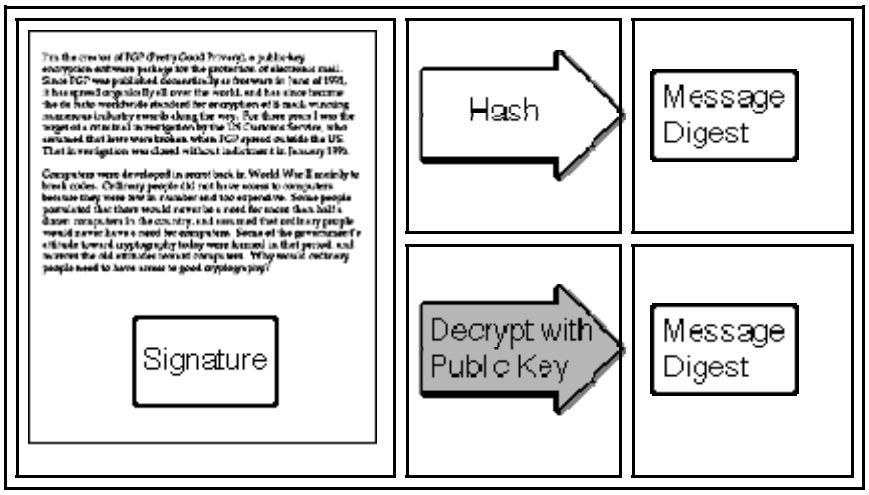


Figure 5.9. Example- received message hashing

Bob now passes the document on to Pat.

Pat	First, Pat's software decrypts the signature (using Bob's public key) changing it back into a message digest. If this worked, then it proves that signed the document, because only Bob has his private key. Pat's software

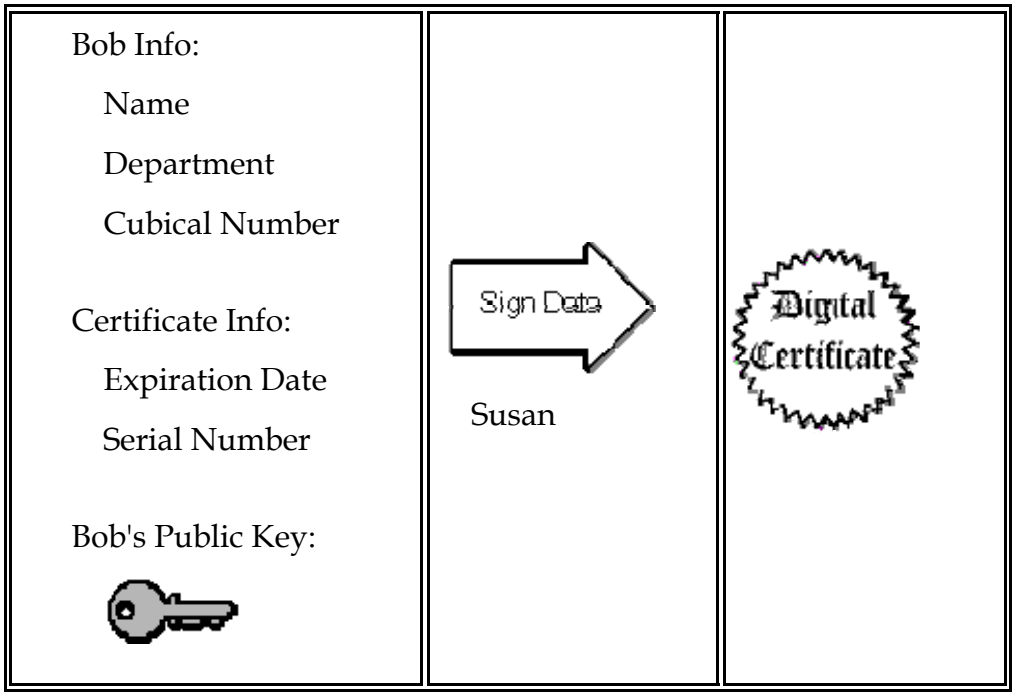
	then hashes the document data into a message digest. If the message digest is the same as the message digest created when the signature was decrypted, then Pat knows that the signed data has not been changed

Figure 5.10. Example- received message verificarion

Plot complication...

Doug	Doug (our disgruntled employee) wishes to deceive Pat. Doug makes sure that Pat receives a signed message and a public key that appears belong to Bob. Unbeknownst to Pat, Doug deceitfully sent a key pair created using Bob's name. Short of receiving Bob's public key from him person, how can Pat be sure that Bob's public key is authentic?

It just so happens that Susan works at the company's certificate authority center. Susan can create a digital certificate for Bob simply by signing Bob's public key as well as some information about Bob.



Now Bob's co-workers can check Bob's trusted certificate to make sure that his public key truly belongs to him. In fact, no one at Bob's company accepts a signature for which there does not exist a certificate generated by Susan. This gives Susan the power to revoke signatures if private keys are compromised, or no longer needed. There are even more widely accepted certificate authorities that certify Susan.

Let's say that Bob sends a signed document to Pat. To verify the signature on the document, Pat's software first uses Susan's (the certificate authority's) public key to check the signature on Bob's certificate. Successful de-encryption of the certificate proves that Susan created it. After the certificate is de-encrypted, Pat's software can check if Bob is in good standing with the certificate authority and that all of the certificate information concerning Bob's identity has not been altered.

Pat's software then takes Bob's public key from the certificate and uses it to check Bob's signature. If Bob's public key de-encrypts the signature successfully, then Pat is assured that the signature was created using Bob's private key, for Susan has certified the matching public key. And of course, if the signature is valid, then we know that Doug didn't try to change the signed content.

5.6 Digital Siganture Standard(DSS)

When a message is received, the recipient may desire to verify that the message has not been altered in transit. Furthermore, the recipient may wish to be certain of the originator's identity. Both of these services can be provided by the DSA. A digital signature is an electronic analogue of a written signature in that the digital signature can be used in proving to the recipient or a third party that the message was, in fact, signed by the originator. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any late time.

This section describes the DSA for digital signature generation and verification. In addition, the criteria for the public and private keys required by the algorithm are provided.

5.6.1. Use if the DSA Algorithm

The DSA is used by a signatory to generate a digital signature on data and by a verifier to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process. For both signature generation and verification, the data which is referred to as a message, M, is reduced by means of the Secure Hash Algorithm (SHA). An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a correctly signed message.

Means of associating public and private key pairs to the corresponding users is required. That is, there must be a binding of a user's identity and the user's public key. This binding may be certified by a mutually trusted party. For example, a certifying authority could sign credentials containing a user's public key and identity to form a certificate.

5.6.2. DSA Parameters

The DSA makes use of the following parameters:

1. p = a prime modulus, where $2^{L-1} < p < 2^{L}$ for 512 = < L = <1024 and L a multiple of 64

2. q = a prime divisor of p - 1, where $2^{159} < q < 2^{160}$

3. $g = h^{(p-1)/q} \bmod p$, where h is any integer with 1 < h < p - 1 such that

$h^{(p-1)/q} \bmod p > 1$ (g has order q mod p)

4. x = a randomly or pseudorandomly generated integer with $0 < x < q$

5. $y = g^x \bmod p$

6. k = a randomly or pseudorandomly generated integer with $0 < k < q$

The integers p, q, and g can be public and can be common to a group of users. A user's private and public keys are x and y, respectively. They are normally fixed for a period of time. Parameters x and k are used for signature generation only, and must be kept secret. Parameter k must be regenerated for each signature.

Parameters p and q shall be generated as specified in FIPS (Federal Information Processing Standards Publications) approved security methods. Parameters x and k shall be generated as specified in FIPS approved security methods.

5.6.3. Signature Generation

The signature of a message M is the pair of numbers r and s computed according to the equations below:

$r = (g^k \bmod p) \bmod q$ and

$s = (k^{-1}(SHA(M) + xr)) \bmod q.$

In the above, k-1 is the multiplicative inverse of k, mod q; i.e., (k-1 k) mod q = 1 and $0 < k\text{-}1 < q$. The value of SHA(M) is a 160-bit string output by the Secure Hash Algorithm specified in FIPS 180. For use in computing s, this string must be converted to an integer.

As an option, one may wish to check if r = 0 or s = 0. If either r = 0 or s = 0, a new value of k should be generated and the signature should be recalculated (it is extremely unlikely that r = 0 or s = 0 if signatures are generated properly).

The signature is transmitted along with the message to the verifier.

5.6.4. Signature Verification

Prior to verifying the signature in a signed message, p, q and g plus the sender's public key and identity are made available to the verifier in an authenticated manner.

Let M', r' and s' be the received versions of M, r, and s, respectively, and let y be the public key of the signatory. To verifier first checks to see that $0 < r' < q$ and $0 < s' < q$; if either condition is violated the signature shall be rejected. If these two conditions are satisfied, the verifier computes

$w = (s')^{-1} \bmod q$

$u1 = ((SHA(M')w) \bmod q$

$u2 = ((r')w) \bmod$

$v = (((g)^{ul} (y)^{u2}) \bmod p) \bmod q.$

If v = r', then the signature is verified and the verifier can have high confidence that the received message was sent by the party holding the secret key x corresponding to y. For a proof that v = r' when M' = M, r' = r, and s' = s.

If v does not equal r', then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message should be considered invalid.

5.7. Benefits of Digital Signatures

There are three common reasons for applying a digital signature to communications:

Authentication

Public-key cryptosystems allow encryption of a message with a user's private key. The message itself need not be sent in ciphertext. If a hash of the document is generated and and then protected via encryption, the document cannot be altered in any way without changing the hash to match, which, if quality algorithms are properly used, will be quite difficult. By decrypting the hash using the sender's public key, and checking the result against a newly generated hash of the alleged plaintext, the recipient can confirm (with high confidence) that the encryption was done with the sender's private key (and so presumably by the user who should have been the only person able to use that key), and that the message hasn't been altered since it was signed. No recipient can ever be absolutely certain the purported sender is indeed the signer -- ie, the person who used the private key -- since the cryptosystem might have been broken, the key copied, or the whole scheme evaded using social engineering.

The importance of high confidence in both the message integrity and sender authenticity is especially obvious in a financial context. For example, suppose a bank's branch office sends instructions to the central office in the form *(a,b)* where *a* is the account number and *b* is the amount to be credited to the account. A devious customer may deposit £100, observe the resulting transmission and repeatedly retransmit *(a,b)*, getting a deposit each time and getting rich in the process. This is an example of a *replay attack*.

Integrity

Both parties will always wish to be confident that a message has not been altered during transmission. Encryption of the message makes it difficult for a third party to *read* it, but that third party may still be able to *alter* it, perhaps maliciously, without actually reading it. An example is the *homomorphism attack*: consider a bank which sends instructions from branch offices to the central office in the form *(a,b)* where *a* is the account number and *b* is the amount to be credited

to the account. A devious customer may deposit £100, intercept the resulting transmission and then transmit (a,b^3) to become an instant millionaire. hi there

Non-repudiation

In a cryptographic context, the word *repudiation* refers to the act of disclaiming responsibility for a message (ie, claiming it was sent by some third party, certainly not me; 'I repudiate this message and its contents!"). A message's recipient may insist the sender attach a signature in order to make later repudiation more difficult, since the recipient can show the signed message to a third party (eg, a court) to reinforce a claim as to its origin. However, loss of control over a user's private key will mean that all digitally signatures using that key, and so 'from' that user, are suspect. Noticing that such a loss of control has occurred is not a cryptographic problem, but a human space oen, and is unsolved. Short of special purpose protocols to address this issue, digital signatures alone cannot provide inherent non-repudiation.

5.8. Challenges and Opportunities

The prospect of fully implementing digital signatures in general information exchange presents both advantages and disadvantages, or benefits and costs. The costs or disadvantages consist mainly of:

- **Institutional overhead**: The cost of establishing and utilizing certification authorities, repositories, and other important services, as well as assuring quality in the performance of their functions through means such as professional accreditation, oversight by another, superior certification authority, licensing and governmental regulation, periodic auditing, or legal and financial responsibility for errors and omissions.
- **Product cost**: A digital signer will require software that may well be more expensive than a simple pen, and may probably also have to pay a

certification authority to issue a certificate. Equipment to secure one's private key may also be advisable. Recipients of digital signatures will incur expenses for verification software and perhaps for access to certificates in a repository.

On the plus side, the principal advantage to be gained is more reliable authentication of messages. Digital signatures, if properly implemented and utilized:

- **Impostors**: Minimize the risk of dealing with impostors or persons who can escape responsibility by claiming to have been impersonated.
- **Message corruption**: Minimize the risk of tampering with messages, altering the terms of a transaction and covering up the traces of the alteration, or false claims that a message was altered after it was sent.
- **Formal legal requirements**: Strengthen the support for concluding that legal requirements of form, such as writing, signature, and an original document, are satisfied, since digital signatures are functionally on a par with or superior to paper forms.
- **Open systems**: Retain a high degree of information security, even for information sent over open, insecure, but inexpensive and widely used communication channels.

Considering the alternatives, such as paper signatures, computerized images of handwritten signatures, or typed signatures such as "John Smith", the benefits of digital signatures outweigh their burdens. The ABA Guidelines and Utah Act are intended to advance legal recognition of digital signatures and establish an institutional infrastructure to support digital authentication.

Chapter 6
Digital Signature Based ARP Protocol

The major flaw in ARP is the lack of message authentication. For the remainder of this chapter, we classify ARP vulnerabilities as falling into one of the two following categories:

Reply spoofing: forging an ARP reply to inject a new address association into the victim's cache

Entry poisoning: forging an ARP reply to replace an address association in the victim's cache

We address these vulnerabilities through the Digital Signature based Address Resolution Protocol.

6.1. LAN Architecture

DS based ARP implements security by distributing centrally generated attestations. These attestations, called certificates, authenticate the association between MAC and IP addresses through statements signed by the local Certificate Authority System (CAS).

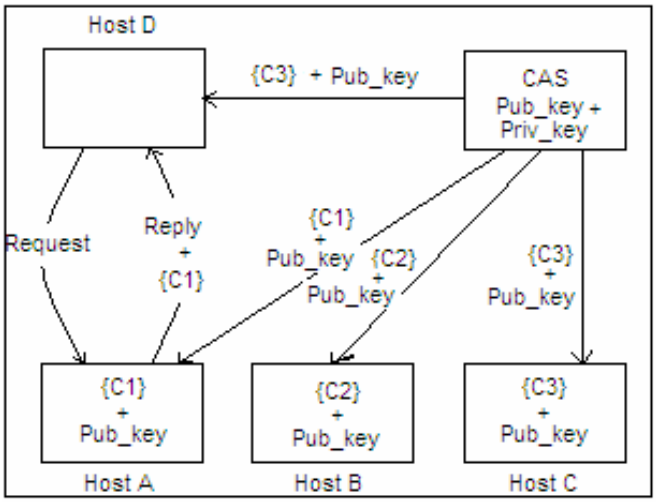


Figure 6.1. Digital Signature Based ARP Protocol.

A host out of the LAN is given the responsibility of CAS (Certificate Authority System) to generate the certificates for the host joining the LAN. The Certificates are nothing but the digital signature generated by the CAS for each host by applying private key to their IP/MAC address pairs.

The CAS generates a public-private key pair to generate signature. Suppose CAS has generated a key pair Pub_key – Priv_key, where Pub_key is the public key that has to be distributed to each host as they join the network, and Priv_key is the private key that is used to generate the signatures for the host as they join LAN.

Consider a host A joining the LAN with MAC address A_{MAC} . Host A is assigned an IP address $A_{IP.}$ Now it is the responsibility of LAN administrator to generate the signature for the host A joining the LAN to let the host A to authenticate its outgoing ARP reply.

In the simple TCP/IP LAN, as the system joins LAN, it is assigned IP address, Subnet mask, and default gateway. In the LAN proposed in this thesis will require the host to be assigned the Digital signature(certificates) and public key, generated by CAS.

The Rules to generate the digital signature for the any host A is as follows-

$A_{SIGNATURE} = Priv\_key\ (A_{IP} + A_{MAC})$

Now in our example host A will be given two more information, $A_{SIGNATURE}$ and Pub_key.

6.2. DS based ARP Frame Format

The frame format for the standard ARP protocol can be modified to extend it to DS based ARP protocol. The frame format for the ARP request remain the same, Only the frame format of ARP reply is modified to accommodate authentication information. The ARP reply frame format is shown below.

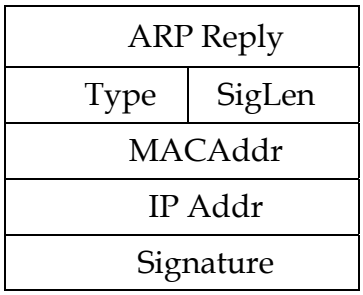


Figure 6.2. Frame format of ARP reply DS based ARP protocol.

1. ARP Reply – this field is same as the standard ARP message field
2. Type – Type fiels is used to distinguish between different types of Digital Signatures being using.
3. SigLen – The Siglen field specifies the length of the Signature being used.
4. MACAddr - MACAddr fields is MAC address of the host sending ARP reply.
5. IPAddr - IPAddr fields is IP address of the host sending ARP reply.
6. Signature – It is the digital signature generated by the CAS and assigned at the time of network joining.

6.3. DS based ARP Protocol.

6.3.1. DS based ARP Request-

~ Outgoing Request – The outgoing ARP request goes as normal as in the standard ARP protocol.

~ Incoming Request – The incoming ARP request is process normally except the way ARP reply is sent as described in next subsection below.

6.3.2. DS based ARP Reply-

~ Outgoing Reply – The ARP reply is sent in the frame format explained above. The sending host fills out the information in ARP reply with its IP address, MAC address, type of signature, length of signature and signature.

~ Incoming ARP reply – The host receiving the ARP reply collects the various information from the ARP reply messages for further processing. The receiving host will update its arp cache only if the following condition satisfies (suppose

host B receives an ARP reply message from host A)-

$A_{SIGNATURE}$ = Pub_key (A_{IP} + A_{MAC})

Received_ $A_{SIGNATURE}$ = $A_{SIGNATURE}$

Only if the condition above gets satisfied, the cache is updated to reflect the ARP reply received. The above condition reveals that the reply is from the correct host and not an ARP spoof message.

6.4. Implementation

The approach explained had been implemented in Linux platform. This implementation is only from the point of demonstration on small scale LAN. This project has not been tested on commercial basis. Though this can be extended for the large scale production as a prevention measure for the ARP spoofing for the commercial production.

The major components of implementation are-

- Loadable kernel module – this module is loaded to disable the processing of ARP reply messages and to put new module. All ARP reply messages has to be verified a non-spoofing massages. For this has to be done all frames should not go to the default kernel module for processing. This kernel module just does the same things like existing kernel module except filtering out the ARP replies.
- User space daemon – this daemon just process the incoming and outgoing ARP replies to accommodate the digital signature. In incoming ARP replies it just checks that condition for DS-based ARP protocol (as described in section 6.3.2) holds or not. If the condition is true, the ARP reply is accepted and ARP cache is modified otherwise the ARP reply is simply dropped without cache modification or cache entry.

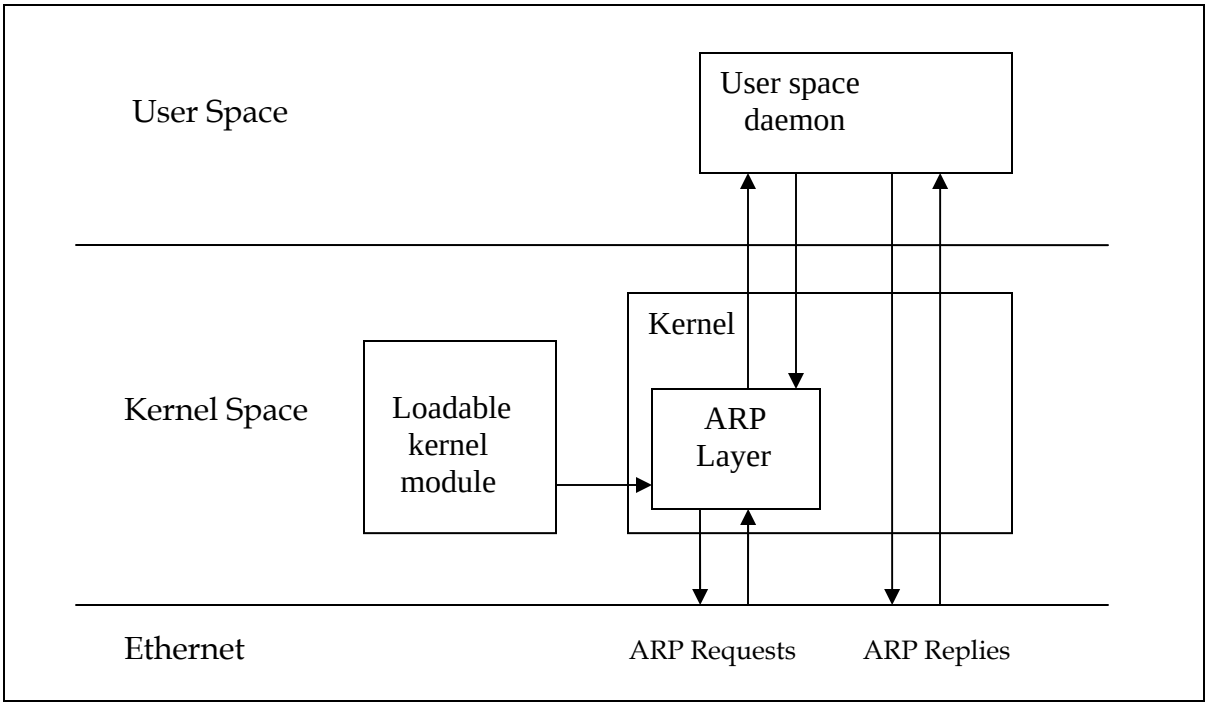


Figure 6.3. Architechture of DS based ARP protocol.

The figure above shows the architechture of DS-based ARP protocol from the point of view of implementation.

6.5. Proof of Concept

In DS-based ARP protocol network, an attacker's attempt to send an ARP spoof message containing IP address of victim machine with its own MAC address would fail because that protocol needs the certificates generated by CAS based on IP/MAC address pair. An attacker has to have the knowledge of CAS's private key to generate the certificate based on IP(of victim host)/MAC(its own) address pair, that is not available to him because it is kept safe with LAN administrator on CAS machine.

Chapter 7
Results

The protocol described in this thesis has been tested on LAN. It can be found experimentally that this approach to handle ARP spoof can eliminate the spoofing problem. But generation of digital signature(public-private key pair and certificates for each system in LAN) is time consuming, moreover certificate verification takes time to process. All these operations thus result in time consuming ARP protocol. Existing ARP protocol is faster as compared to DS based ARP protocol due to less procession, but the problem of spoofing remains live.

The problem of ARP spoofing can be handled by using DS based ARP protocol, but only at the cost of time, which is taken by certificate processing. But it much better than the other handling techniques which involve changes in existing LAN technologies.

The DS-based Arp protocol can be easily accommodated in existing TCP/IP networks for the prevention of ARP based spoofing attacks on LAN.

Chapter 8
Related Work

Defenses Against ARP Poisoning

A possible defense against ARP poisoning is using static entries in the ARP cache. Static entries cannot be updated by ARP replies and can be changed only manually by the system administrator. Such an approach however is not viable for networks with hundreds of hosts because those entries must be inserted manually on each host. Automating such a solution via a network script is not recommendable since it relies on higher levels of the ISO/OSI stack. Relying on higher levels when the data link layer has not been secured yet may be dangerous because the protocol used to exchange the list can be hijacked using ARP poisoning before the list is distributed. Even worse, some operating system (such as Windows) may accept dynamic updates even if an entry is set as static, thus making static Ethernet routing useless [21].

"Port security" is another mechanism for tackling the problem. It is a feature present in many modern switches that allows the switch to recognize only one MAC address on a physical port. This is often suggested as an effective protection against ARP poisoning, but it is not. If the attacker does not spoof its own MAC address, it can poison the two victims' cache without letting the switch interfere with the poisoning process.

Besides static cache entries and port security, the only other defense that will not modify ARP behaviour is detection. IDS and personal firewalls usually notice the ARP switch and warn the user that the entry in the cache is changed. As it often happens in the computer security domain, the decision is left to the user and his/her awareness. Given the particularly sophisticated level of operation in this case, we doubt the average user will take the proper actions.

Some kernel patches exist that try to defend against ARP poisoning. "Anticap" [22] does not update the ARP cache when an ARP reply carries a different MAC address for a given IP from then one already in cache and will issue a kernel alert that someone is trying to poison the ARP cache. Such a solution is against ARP definition itself, since it drops legal gratuitous ARP. "Antidote" [23] is more sophisticated. When a new ARP replies announcing a change in a <IP, MAC> pair is received, it tries to discover if the previous MAC address is still alive. If the previous MAC address replies to the request, the update is rejected and the new MAC address is added to a list of "banned" addresses. In a solution[24] that implements two distinct queues, for requested addresses and received replies, is proposed. The system discards a reply if the corresponding request was never sent, i.e., is not in the queue, and in the received queue an IP address associated with a different Ethernet address is already present.

All these solutions have the same problem. If the malicious ARP reply is sent before the real one is put in the cache, for a real request, the victim caches the wrong reply and discards the real one. A race condition exists between the attacker and the victim. When the first ARP request is broadcast, both the victim and the attacker receive the message. The first one who replies will take over the other forever. Furthermore, the attacker could also spoof an ICMP echo request message and immediately send after it a false ARP reply. When the victim receives the ICMP echo request, it performs an ARP request, but the false reply is already in its queue of received packet, so it accepts it a the valid one. If Antidote is installed, a host can spoof the sender MAC address and force a host to ban another host.

Solutions such as a centralized ARP cache or a DHCP server broadcasting ARP information, as they are deployed in IP over ATM networks, have not been considered as the attacker could spoof the source of the broadcast and poison the whole LAN. A digitally signed or MAC-ed broadcast packet would not be

vulnerable to spoofing, yet broadcasting ARP tables could generate large traffic on the LAN. Since an entry for each host needs to be broadcast, on large networks this will generate considerable traffic and every host would have to store the entire ARP table even if it might not be needed at the moment. The main problem with centralized ARP cache is that if a host goes down, the central server will not notice the event. Thus, when a host that wishes to communicate with the one currently down asks for ARP information to the central server, it will receive the information even if the host is down. At this point an attacker could impersonate the offline host using its MAC address and receive all the packets sent to it.

Chapter 9
Conclusions and Future Work

The thesis presents a feasible solution to the problem of ARP poisoning attacks. The cause of ARP poisoning is the lack of message authentication, so that any host in the LAN is able to spoof messages pretending to be someone else. This thesis propose an authentication scheme for ARP replies using public key cryptography, which extends ARP to be secured. Adding strong authentication to ARP messages resolves the problem, thus denying any attempt of ARP poisoning.

Future work includes porting authenticated ARP to other platforms so as to allow interoperability. Better kernel integration will be implemented since the upcoming Linux kernel (2.6.0) will be fully preemptible. Once the implementation of cryptographic routine will be moved to kernel space, even authenticated ARP request will be signed and the receiver will cache the information on the request, thus speeding up the whole authentication process.

When firewall and gateway appliances will be equipped with cryptographic co-processors, the implementation of the approach given in this thesis on embedded systems could be considered.

Chapter 10
References

1. Man-in-the-middle attack.
   http://en.wikipedia.org/wiki/Man_in_the_middle_attack.

2. Man-in-the-Middle Phishing Attack Successful Against Citibank's 2-Factor Token Authentication.
   http://www.tricipher.com/news/pr120.htm.

3. ARP Vulnerabilities: The Complete Documentation.
   http://www.l0t3k.org/security/docs/arp/

4. Defenses against the MITM attack.
   http://www.answers.com/topic/man-in-the-middle-attack-1.

5. ARP-Spoofing attacks.
   http://de.wikipedia.org/wiki/ARP-Spoofing.

6. Tutorials. Beware of ARP Attacks.
   http://www.wirelessnetworkingacademy.com/learning_center/ tutorials/ Beware_of_ARP_Attacks.htm

7. How to cope with ARP attacks on LANs
   http://www.techworld.com/security/features/index.cfm?featureid=727 &pagtype=samecatsamechan

8. DNS cache poisoning.
   http://en.wikipedia.org/wiki/DNS_cache_poisoning.

9. DNS Spoofing techniques.
   http://www.securesphere.net/download/papers/dnsspoof.htm.

10. DNS Cache poisoning, Definition & Prevenation.
    http://www.infosecwriters.com/text_resources/pdf/DNS_TOlzak.pdf

11. What is a Digital Signature? An introduction to Digital Signatures, by David Youd
    http://www. youdzone.com/signature.html.

12. Digital Signature Guidelines Tutorial.
http://www.abanet.org/scitech/ec/isc/dsg-tutorial.html.

13. Digital signature encyclopedia.
http://en.wikipedia.org/wiki/Digital_signature.

14. How do digital signatures work?
http://computer.howstuffworks.com/question571.htm.

15. Address Resolution Protocol.
http://www.l0t3k.org/security/docs/arp/

16. Simplistic overview of Address Resolution Protocol.
http://webpages.cs.luc.edu/~mt/student/george/ARP-Bobeck.pdf

17. The Ingredients to ARP Poison.
http://www.governmentsecurity.org/articles/TheIngredientstoARP Poison.php

18. ARP Guard - Protection from ARP Spoofing Attacks.
http://www.3mfuture.com/network_security/arp-guard-arp-spoofing.htm.

19. ARP Poisoning in practice.
http://www.infosecwriters.com/text_resources/pdf/ARP_Poisoning_ In_Practice.pdf

20. Denial-of-service attack.
http://en.wikipedia.org/wiki/Denial_of_service

21. S. Whalen. An introduction to arp spoofing.
http://packetstormsecurity.nl/papers/protocols/ intro to arp spoofing.pdf

22. M. Barnaba. anticap.
http://cvs.antifork.org/cvsweb.cgi/anticap, 2003.

23. I. Teterin. Antidote.
http://online.securityfocus.com/archive/ 1/299929.

24. M. V. Tripunitara and P. Dutta. A middleware approach to asynchronous and backward compatible detection and prevention of arp cache poisoning. In *Proc. 15th Annual Computer Security Application Conference (ACSAC)*, pages 303– 309, 1999.

Chapter 11
Bibliography

1. **TCP/IP Illustrated, Volume 1**: The Protocols, Addison-Wesley, 1994, ISBN 0-201-63346-9.
   By W. Richard Stevens.

2. **Unix Network Programming: The Sockets Networking API**
   By W Richard Stevens, Bill Fenner, Andrew M Rudoff

3. **C++ Network Programming: Mastering Complexity with Ace and Patterns**
   By Douglas C Schmidt, Stephen D Huston

4. **Digital Signature:** Network Security Practices
   By Kailash N. Gupta, Kamlesh N. Agarwala, Prateek A. Agarwala

5. **Computer Networks**
   By Tanenbaum, Andrew S Tanenbaum.

6. **Unix** Shell **Programming**
   By Stephen G Kochan, Patrick Wood.

7. **C++ Network Programming**: Systematic Reuse with Ace and Frameworks
   By Douglas C Schmidt, Stephen D Huston.