

Covert Channel over ICMP

By: Debasish Mandal

<http://www.debasish.in/>

Date: 15/08/2011

Contents:

- ❖ Introduction
- ❖ Some key points about ICMP
- ❖ Some key points about firewalls
- ❖ Uses of ICMP covert channel
- ❖ Breaking an ICMP ping
- ❖ Handmade raw ping using python
- ❖ Establish a simple ICMP covert channel between two host using python
- ❖ Server and Client source Code.
- ❖ Few other well known covert channel tools
- ❖ Mitigation

Introduction and Overview:

An ICMP covert tunnel establishes a covert connection between two computers using ICMP echo requests and reply packets. An example of this technique is tunneling complete TCP traffic over ping requests and replies. More technically we can say ICMP covert tunneling works by injecting arbitrary data into an echo packet sent to a remote computer. The remote computer replies in the same manner, injecting an answer into another ICMP packet and sending it back.

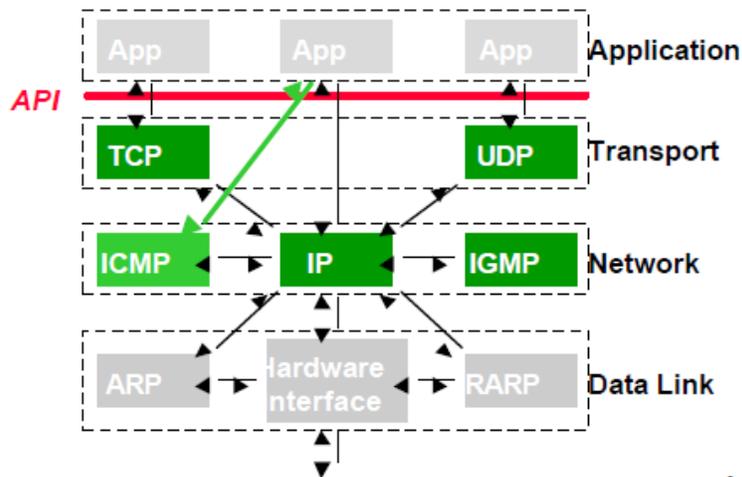
Some Key Points about ICMP:

ICMP Doesn't Have Ports:

We can't actually ping a port. When someone speaks of "pinging a port" they are actually referring to using a layer 4 protocol (such as TCP or UDP) to see if a port is open or not. So if someone "pings" port 80 on a box, that usually means send it a TCP SYN to that system in order to see if it's responding. Real ping uses ICMP, which doesn't use ports at all.

ICMP Works At Layer Three (3)

While ICMP sits "on top of" IP, ICMP is not a layer 4 protocol. It's still considered to be at layer 3 rather than one layer higher.



Network layer routes the packets according to the unique network addresses. Router works as the post office and network layer stamps the letters (data) for the specific destinations.

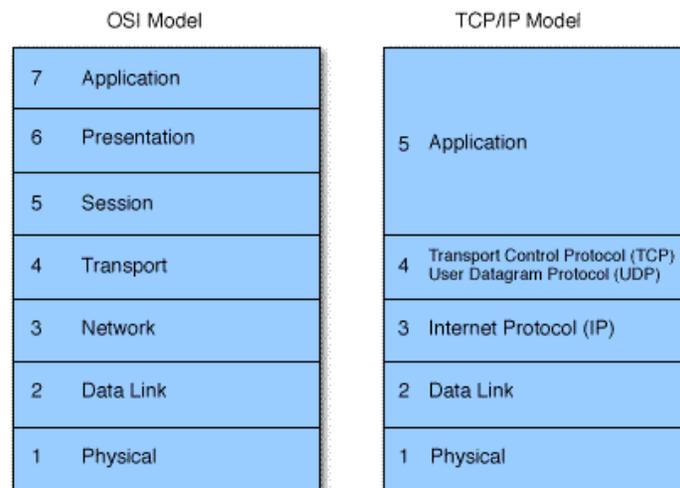
Protocols: These protocols work on the network layer IP, ICMP, ARP, RIP, OSI, IPX and OSPF.

Network Devices: Network devices including Router, Brouter, Frame Relay device and ATM switch devices work on the network layer.

Some Key Points about Firewalls:



Firewalls operate at different layers to use different criteria to restrict traffic. The lowest layer at which a firewall can work is layer three. In the OSI model this is the network layer. In TCP/IP it is the Internet Protocol layer. This layer is concerned with routing packets to their destination. At this layer a firewall can determine whether a packet is from a trusted source, but cannot be concerned with what it contains or what other packets it is associated with. Firewalls that operate at the transport layer know a little more about a packet, and are able to grant or deny access depending on more sophisticated criteria. At the application level, firewalls know a great deal about what is going on and can be very selective in granting access.



It would appear then, that firewalls functioning at a higher level in the stack must be superior in every respect. This is not necessarily the case. The lower in the stack the packet is intercepted, the more secure the firewall. If the intruder cannot get past level three, it is impossible to gain control of the operating system.

Uses of ICMP Covert Channel:



ICMP tunneling can be used to bypass firewalls rules through obfuscation of the actual traffic. Depending on the implementation of the ICMP tunneling software, but this type of connection can also be categorized as an encrypted communication channel between two computers. Without proper deep packet inspection or log review, network administrators will not be able to detect this type of traffic through their network.

Breaking an ICMP packet:



To break an ICMP packet we are going to send ICMP echo requests to a remote host and sniff the network traffic.

Let's have a look at a normal ping.

Here we are sending normal ICMP echo request to the remote host 192.168.157.1.

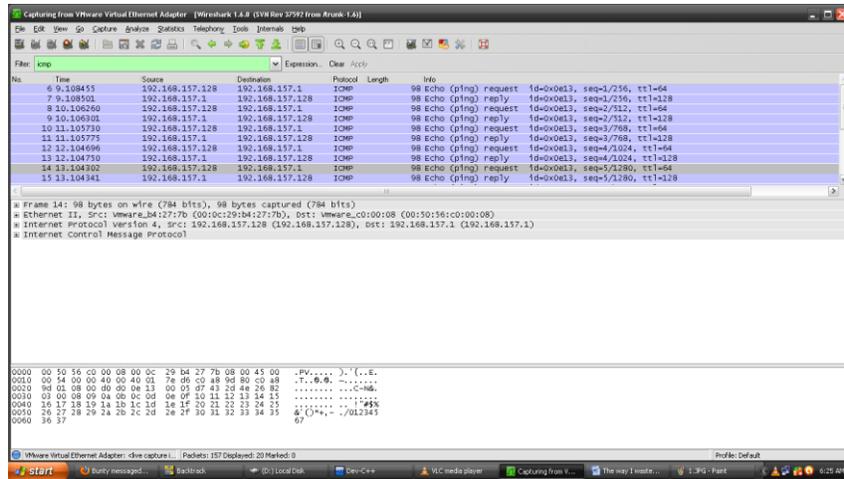
```

root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# ping 192.168.157.1
PING 192.168.157.1 (192.168.157.1) 56(84) bytes of data:
64 bytes from 192.168.157.1: icmp_seq=1 ttl=128 time=4.72 ms
64 bytes from 192.168.157.1: icmp_seq=2 ttl=128 time=0.202 ms
64 bytes from 192.168.157.1: icmp_seq=3 ttl=128 time=0.460 ms
64 bytes from 192.168.157.1: icmp_seq=4 ttl=128 time=0.242 ms
64 bytes from 192.168.157.1: icmp_seq=5 ttl=128 time=0.171 ms
64 bytes from 192.168.157.1: icmp_seq=6 ttl=128 time=0.250 ms
64 bytes from 192.168.157.1: icmp_seq=7 ttl=128 time=0.252 ms
64 bytes from 192.168.157.1: icmp_seq=8 ttl=128 time=0.409 ms
64 bytes from 192.168.157.1: icmp_seq=9 ttl=128 time=0.497 ms
64 bytes from 192.168.157.1: icmp_seq=10 ttl=128 time=0.244 ms
^C
--- 192.168.157.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.171/0.745/4.728/1.332 ms
root@bt:~#

```

Capturing the traffic using Wire shark we can see this:



As we have used the basic ping utility with no options, it's sent multiple ICMP echo requests to the host. So it will be bit complicated for us to analyze the entire traffic. So let's send a single ICMP packet with 0 bytes of data.

Here we can use following command to send an ICMP packet with 0 bytes of payload.

```
ping -c 1 -s 0 <host>
```

```

10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.171/0.745/4.728/1.332 ms
root@bt:~# ping -c 1 -s 0 192.168.157.1
PING 192.168.157.1 (192.168.157.1) 0(28) bytes of data:
8 bytes from 192.168.157.1: icmp_seq=1 ttl=128
--- 192.168.157.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms

root@bt:~#

```

Now it will be easier for us to analyze the traffic.

In sniffer in the remote host we can see we have received 42 bytes of data.

```
00 50 56 c0 00 08 00 0c 29 b4 27 7b 08 00 45 00 .PV.... ).' {...E.
00 1c 00 00 40 00 40 01 7f 0e c0 a8 9d 80 c0 a8 ....@.@. ....
9d 01 08 00 e8 eb 0f 13 00 01 ..... ..
```

If we look at a typical ICMP packet structure it will like below figure.

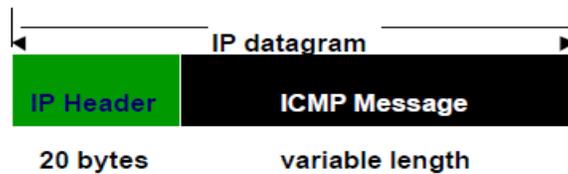
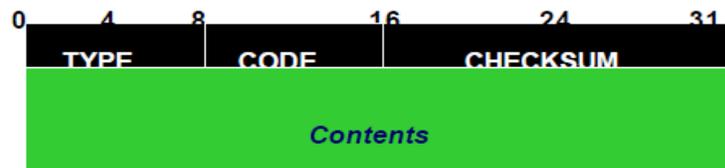


Figure 1

A typical ICMP Header structure is like:



- **TYPE:** Type of ICMP message
- **CODE:** Used by some types to indicate a specific condition
- **CHECKSUM:** Checksum over full message
- **Contents** depend on TYPE and CODE

Analyzing those 42 bytes of data we can understand 1st 14 bytes is the Ethernet Header.

```
00 50 56 c0 00 08 00 0c 29 b4 27 7b 08 00 45 00 .PV.... ).' {...E.
00 1c 00 00 40 00 40 01 7f 0e c0 a8 9d 80 c0 a8 ....@.@. ....
9d 01 08 00 e8 eb 0f 13 00 01 ..... ..
```

In the Ethernet header the 1st 12 bytes you can see that this is nothing but the Hardware addresses of the Destination and source machine.

The next 20 BYTE of the received datagram is the IP Header. (This supports the Figure 1?)

```
00 50 56 c0 00 08 00 0c 29 b4 27 7b 08 00 45 00 .PV.... ).' {...E.
00 1c 00 00 40 00 40 01 7f 0e c0 a8 9d 80 c0 a8 ....@.@. ....
9d 01 08 00 e8 eb 0f 13 00 01 ..... ..
```

The IP header structure in more details:

```

45     VERSION IPV4
00     Different Services
00 1c   Total length 28
00 00   Identification 0x000000
40 01   Dont Fragment offset
40     Time to live
01     ICMP version 1
7f 0e   Header Check sum
c0 a8 9d 80   --- Source Ip   192.168.157.128
c0 a8 9d 01   -- Destination IP   192.168.157.1

```

Now the next 8 bytes is ICMP Header:

```

00 50 56 c0 00 08 00 0c 29 b4 27 7b 08 00 45 00  .PV..... ).' {...E.
00 1c 00 00 40 00 40 01 7f 0e c0 a8 9d 80 c0 a8  ....@.@. ....
9d 01 08 00 e8 eb 0f 13 00 01  .....
```

ICMP Header Structure in more details:

```

08   ---  >>  Type ECHO 8
00   ---->>  CODE = 0
e8 eb ---->>  Checksum
0f 13  -----  >>  Identifier
00 01  ---  >>>  Sequence Number|

```

This also supports figure 2.

The ping utility present in a typical Linux system actually allows us to control number of packets, size of packets, but if we want to craft a ping in more detail it will not allow us to do that. As our main objective is to manipulate the data portion ICMP packet so using normal utility ping is not a good choice.

HPING2:

Hping is a free packet generator and analyzer for the TCP/IP protocol distributed by Salvatore Sanfilippo (also known as Antirez) using this we can also control / and manipulate the data part of a single ICMP packet. Here again we are going to send a single ICMP echo packet with the help of hping but this time we are going to add some junk data with the ICMP packet.

```

Session Edit View Bookmarks Settings Help
root@bt:~# hping2 -1 -c 1 192.168.157.1 -e "AAAA"
HPING 192.168.157.1 (eth0 192.168.157.1): icmp mode set, 28 headers + 4 data bytes
es
[main] memlockall(): Success
Warning: can't disable memory paging!
len=46 ip=192.168.157.1 ttl=128 id=894 icmp_seq=0 rtt=3.0 ms

--- 192.168.157.1 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 3.0/3.0/3.0 ms
root@bt:~#
```

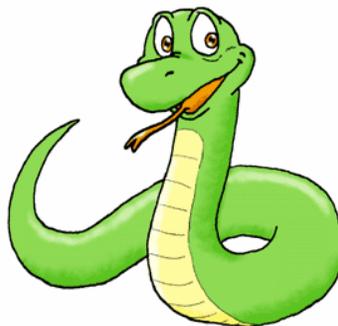
You can see in the screenshot that we have added 4 "A" s with the ICMP packet and sent it to the HOST 192.168.157.1.

Let's sniff the traffic and analyze it.

```
00 0c 29 b4 27 7b 00 50 56 c0 00 08 08 00 45 00 ..).'.{.P V.....E.
00 20 03 7e 00 00 80 01 7b 8c c0 a8 9d 01 c0 a8 . .~..... {.....
9d 80 00 00 4b 6a 32 13 00 00 41 41 41 41 ....Kj2. ..AAAA
```

Now you can see we have captured 46(42+4) bytes of data and in the screen shot you can easily see our payload that's highlighted.41 is the HEX representation of "A".

Handmade raw PING: Life is short, I always prefer python:



In the above examples we have used normal ping utility and hping to send ICMP echo request to any host. Now we are going to use a python to send a handmade raw ICMP echo packet.

```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
GNU nano 2.0.7 File: send1.py
#!/usr/bin/env python
from socket import *
from binascii import hexlify, unhexlify
s = socket(AF_PACKET, SOCK_RAW)
s.bind(("eth0", 0))
dump = "\x00\x50\x56\xc0\x00\x08\x00\x0c\x29\xb4
\x27\x7b\x08\x00\x45\x00\x00\x21\x5d\x68
\x00\x00\x40\x01\x61\xa2\xc0\xa8\x9d\x80
\xc0\xa8\x9d\x01\x08\x00\x00\x1d\x19\x12
\x00\x00\x41\x41\x41\x41"
s.send(dump)
```

In this case we are not only crafting ICMP header and the payload of the packet but also the Ethernet Header, IP header part.

The variable “dump” in the above mentioned script is holding entire datagram. Including Ethernet header IP header and ICMP header and also the payload. Here I have put 41 that mean 4 no. of “A”s.

Now from the above discussion it’s clear that we can easily send arbitrary data to any host by injecting the data into an echo packet. Now if we can plant a demon in the remote host which can replies in the same manner, injecting an answer into another ICMP packet and sending it back.

Packing all these together:



If we can implement following feature into one program then we can do exactly the same thing

- 1)** A daemon will be running which will sniff all ICMP packets.
- 2)** After receiving a ping, extract the payload (valid query from attackers’ side) from ping packets.
- 3)** Do necessary things depending on the query received through ping.
- 4)** Ping back with the answers of the query.

Here I have written a simple ICMP tunneling server and a client in python to establish a covert channel between two hosts.

Code for Server:

<http://pastebin.com/JLD6grb2>

Code for Client:

<http://pastebin.com/gH3zzHdQ>

This covert channel daemon basically receives OS commands from ping and executes the command on the remote host and ping us back the command output. Once the daemon is started on host it will start sniffing ICMP packets. After receiving commands from the client it will extract the payload from the ping packets and execute the command on that host and ping us back the output. If the command output is long the daemon will send the output through multiple pings.

The client side sniffer will receive the response packets and extract the command output from payload. After parsing the command out put it will show the output. The tool Hping can also be used as a client of this daemon. In that case the payload should be crafted in a particular pattern so that the daemon can recognize the query. In that case the client script should be used as a response parser or in wire-shark.

A video demo of this Covert Channel can be found here:

<http://www.youtube.com/watch?v=ADHtjwwkErl>

Few other well known covert channel tools:

❖ **LOKI**

LOKI is an Information tunneling program. It uses Internet Control message Protocol echo response packet to carry its payload.

❖ **NCovert**

It hides file transfer by cloaking it in seemingly harmless data using packet forgery.

Sometime its possible to hide actual IP of users.

❖ **007 Shell:**

007 Shell is a simple client/server C program used for remotely administering a system over a network using techniques similar to Loki. Those covert ICMP ECHO_REPLY packets encapsulate the command and response message within the packet payload.

❖ **ICMPTX (IP-over-ICMP):**

ICMPTX is a program that allows a user with root privledges to create a virtual network link between two computers, encapsulating data inside of ICMP packets.

Mitigation:



Preventing covert channels reminds the cat-and-dog chase. You can prevent certain channels only if you're aware of them, can analyze the traffic they generate and then configure your IDS accordingly, e.g. write Snort rules. Still, when covert channels generate different traffic with each packet (e.g. bit-flipping) or use advanced timing techniques, it becomes impossible to prevent them

Although the only way to prevent this type of tunneling is to block ICMP traffic altogether, this is not realistic for a production or real-world environment. One method for mitigation of this type of attack is to only allow fixed sized ICMP packets through firewalls to virtually eliminate this type of behavior. Limit the size of ICMP packets. Large ICMP packet can be seen as suspicious by an IDS system that could inspect the ICMP packet and raise an alarm. However, since there are legitimate uses for large ICMP packets it is difficult to determine if a large ICMP packet is malicious. For example, large echo request packets are used to check if a network is able to carry large packets. Differentiating legal from illegal large packets is even more difficult if covert communication is encrypted. An IDS needs to be able to determine if a packet is encrypted or not. Distinguishing encrypted from non-encrypted packet still remains an open interesting research problem.

Snort provides one or two rules that can help in detecting ICMP payload covert channels. This is one example. Another example is to detect the "Dont Fragment" bit covert channel and so on. These are simple examples. However, snort doesn't provide such rules and one needs to write them customized. In addition, there are some smart covert channels that are very difficult to detect like the ISN covert channel and timing covert channels.

Reference:

<http://danielmiessler.com/study/icmp/>

http://en.wikipedia.org/wiki/Covert_channel

[http://en.wikipedia.org/wiki/Firewall_\(computing\)](http://en.wikipedia.org/wiki/Firewall_(computing))

<http://www.2factor.us/icmp.pdf>