

# Ping – Manually create and send ICMP/IP packets

This chapter requires some knowledge about the internet protocol (IP) and its structure. We are going to manually create ICMP packets to ping a target host. If you skipped the previous chapter you can find a [short recap on the IP protocol header here](#).

## What is ICMP?

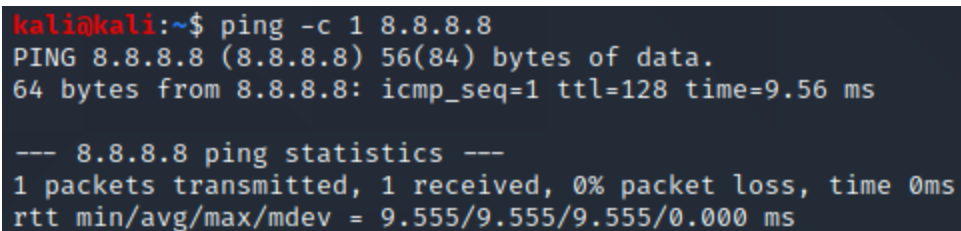
---

The Internet Control Message Protocol (ICMP) is a network-layer protocol of the internet protocol suite (IP) used for sending error messages and diagnostic information indicating success or failure when communicating with another IP address. Many common network utilities like traceroute and ping are based on ICMP messages.

## What is ping?

---

Ping is a tool to test the reachability of a host in a network. This kind of tool is typically available on all operating systems with networking capability. It works by sending a ICMP echo request packet to a target host and waiting for the ICMP echo reply. A typical output of the tool can be seen in the screenshot below:



```
kali@kali:~$ ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=9.56 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9.555/9.555/9.555/0.000 ms
```

The screenshot shows one ICMP echo requests being sent to the host at 8.8.8.8

(Google DNS service). Looking at the sent and received ICMP packets in wireshark we can see the echo request and the corresponding reply:

icmp					
No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	192.168.146.131	8.8.8.8	ICMP	98 Echo (ping) request id=0x0497, seq=1/256, ttl=64 (reply in 4)
4	0.011499781	8.8.8.8	192.168.146.131	ICMP	98 Echo (ping) reply id=0x0497, seq=1/256, ttl=128 (request in 1)

## Structure of a ICMP echo request

ICMP packets are encapsulated in IP packets. We are going to reuse our IP header created in a [previous chapter on IP headers](#) and update it according to our needs:

Bit				
0	4	8	16	31
Version 4	IHL 5	Type of Service 00	Total Length 00 1C	
Identification AB CD			Flags 000 <sub>2</sub>	Fragment Offset 000000000000 <sub>2</sub>
Time to Live 40	Protocol 01		Header Checksum ?? ??	
Source Address C0 A8 92 83 (= 192.168.164.131)				
Destination Address 08 08 08 08 (= 8.8.8.8)				

Important changes to the previous usage of the IP header in combination with TCP is the change of the “Protocol” to **01** for ICMP. The “Total Length” is including the ICMP packet below, therefore changes made there must be met accordingly here. In a previous chapter, the [calculation of the IP header checksum is described](#).

The general structure of a IMCP packet looks like this:

Type of Message	Code	Checksum
Header Data		
Payload Data (Optional)		

**Type of Message:** ICMP Type, [full table](#). E.g.:  for ICMP Echo request,  for ICMP Echo reply.

**Code:** Additional codes for the type of message, [full list](#). ICMP Echo requests have code .

**Checksum:** Used for error checking, calculation shown below.

**Header Data:** In this case (ICMP echo request and replies), will be composed of identifier (16 bits) and sequence number (16 bits). The Identifier and Sequence Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number within that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.

Adjusted to our usage as ICMP Echo request it changes to:

Type of Message 08	Code 00	Checksum ?? ??
Identifier 12 34		Sequence Number 00 01
Payload Data (Optional)		

## ICMP Checksum

The ICMP checksum is easy to calculate. It consists out of all values in the ICMP header added in 16 bit words:

Description	Value	Additional Description
Type of Message and Code + Checksum placeholder	0x0800 + 0x0000 +	-

^

Identifier + Sequence Number	0x1234 + 0x0001 =	–
<b>Subtotal</b>	<b>0x1A35</b>	–
Negation with 0xFFFF	0xFFFF – 0x1A35 =	–
<b>Checksum</b>	<b>0xE5CA</b>	–

## Sending our packet

After we have now manually created our ICMP/IP packet, let's put it on the wire. We slightly adjust our code used in the [previous chapter](#) to send the packet:

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

ip_header = b'\x45\x00\x00\x1c' # Version, IHL, Type of Service | Total
ip_header += b'\xab\xcd\x00\x00' # Identification | Flags, Fragment Offs
ip_header += b'\x40\x01\x6b\xd8' # TTL, Protocol | Header Checksum
ip_header += b'\xc0\xa8\x92\x83' # Source Address
ip_header += b'\x08\x08\x08\x08' # Destination Address

icmp_header = b'\x08\x00\xe5\xca' # Type of message, Code | Checksum
icmp_header += b'\x12\x34\x00\x01' # Identifier | Sequence Number

packet = ip_header + icmp_header
s.sendto(packet, ('8.8.8.8', 0))
```

When working with raw sockets in scripts, most operating system require higher privileges (e.g. root user) to run them:

```
root@kali:~# python3 send_icmp_packet.py
```

^

Utilizing Wireshark, we observe what happens when we send the packet:

icmp						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.146.131	8.8.8.8	ICMP	42	Echo (ping) request id=0x1234, seq=1/256, ttl=64 (reply in 2)
2	0.009108735	8.8.8.8	192.168.146.131	ICMP	60	Echo (ping) reply id=0x1234, seq=1/256, ttl=128 (request in 1)

Here we can see that our packet was received by the destination host and it returned a Echo reply. All these information and more can be found in detail in the [RFC for ICMP](#).

If you want to go even one level lower, have a [look at this script and adjust it to ICMP packets](#) as shown in this post.

## TCP/IP packets

---

### Introduction

---

- 1 Recap on network layers and protocols
- 2 Analysis of a raw TCP/IP packet
- 3 Manually create and send raw TCP/IP packets
- 4 Creating a SYN port scanner

[Ping – Manually create and send ICMP/IP packets](#)

---

### Contact

---

Twitter: [@inc0x0](#)