

LABORATORIO DI RETI E SISTEMI DISTRIBUITI

HANDS-ON 5

ICMP con socket raw

Antonio Mastrolembro Ventura

543644

March 17, 2025

Indice

1	Introduzione	2
2	Definizione del problema	2
3	Analisi del codice	3
3.1	Inclusione librerie	3
3.2	Creazione di un pacchetto ICMP	3
3.3	Argomento del programma	6
3.4	Opzione del socket: timer	6
3.5	Configurazione della destinazione	6
3.6	Build e invio del pacchetto	7
3.7	Ricezione della risposta	8
3.8	Analisi della risposta - Header IP	8
3.9	Analisi della risposta - Header ICMP	9
3.10	Analisi della risposta - ECHO REPLY	10
4	Test del programma	11
5	Analisi del pacchetto - Wireshark	11
5.1	Cattura del pacchetto	11
5.2	Pacchetto inviato	12
5.3	Pacchetto ricevuto	13
6	Conclusioni	13

1 Introduzione

In questo hands-on vedremo un tipo particolare di socket, i cosiddetti **Socket Raw**. Fino ad ora ci siamo occupati di socket tradizionali, come `SOCK_STREAM` per la comunicazione TCP e `SOCK_DGRAM` per la comunicazione UDP. Questi tipi di socket però operano a livello di trasporto e non ci consentono di controllare in maniera diretta i pacchetti inviati e ricevuti. I socket raw ci permettono di incapsulare pacchetti di livello più basso del livello 4, ovvero direttamente a livello di rete o data link, manipolando gli header di protocolli come IP e ICMP. Consentono di "forgiare", con la tecnica del **Packet Crafting**, i pacchetti a nostro piacimento; in questo modo possiamo realizzare strumenti di sniffing del traffico, diagnostica di rete (come il ping che vedremo a breve) e anche tool di sicurezza.

2 Definizione del problema

L'oggetto dell'analisi di questo hands-on sarà un'implementazione semplificata del comando **ping**, che utilizza pacchetti ICMP. Il programma in particolare, sfrutta i socket raw per creare e inviare pacchetti ICMP direttamente a livello di rete. L'obiettivo del programma è inviare pacchetti ICMP ECHO REQUEST a un host specificato e ricevere le eventuali risposte ICMP ECHO REPLY, verificando la raggiungibilità della destinazione; il nostro obiettivo invece consiste nel comprendere al meglio come funziona questa manipolazione dei pacchetti. Andiamo a vedere nel dettaglio come avviene il tutto prendendo in esame il file `ICMP_raw.c`.

3 Analisi del codice

3.1 Inclusione librerie

```
1 #include <netinet/in.h>
2 #include <netinet/ip.h>
3 #include <netinet/ip_icmp.h>
```

Oltre alle librerie standard per l'inclusione delle funzionalità base e le librerie utili per la gestione del socket, abbiamo anche queste tre librerie. In particolare, troviamo il loro contenuto (nei sistemi Linux) nel seguente path:

```
$ cd /usr/include/netinet
```

A breve vedremo nel dettaglio come accedervi adeguatamente.

3.2 Creazione di un pacchetto ICMP

```
1 void build_icmp_packet(struct icmp_hdr *icmp, int sequence) {
2     icmp->type = ICMP_ECHO;
3     icmp->code = 0;
4     icmp->un.echo.id = htons(getpid());
5     icmp->un.echo.sequence = htons(sequence);
6     icmp->checksum = 0;
7
8     // Aggiunge un payload qualunque
9     char *data = (char *) (icmp + 1);
10    memset(data, 'A', PACKET_SIZE - sizeof(*icmp));
11
12    icmp->checksum = compute_checksum(icmp, PACKET_SIZE);
13 }
```

La seguente funzione prende in ingresso una struttura fondamentale: `struct icmp_hdr` (icmp header). La sua definizione è presente all'interno di un file: `ip_icmp.h`, che si trova nel percorso specificato al punto 3.1. Vediamola nel dettaglio:

```

1 struct icmp_hdr
2 {
3     uint8_t type;           /* message type */
4     uint8_t code;           /* type sub-code */
5     uint16_t checksum;
6     union
7     {
8         struct
9         {
10             uint16_t id;
11             uint16_t sequence;
12         } echo;             /* echo datagram */
13         uint32_t gateway;    /* gateway address */
14         struct
15         {
16             uint16_t __glibc_reserved;
17             uint16_t mtu;
18         } frag;             /* path mtu discovery */
19     } un;
20 };

```

Questa struct rappresenta naturalmente l'header del pacchetto ICMP. Nel nostro caso stiamo impostando:

- `type = ICMP_ECHO` ovvero una ECHO REQUEST di ping;
- `code = 0` cioè il valore predefinito per i pacchetti di tipo ECHO REQUEST;
- Con `un.echo.id = htons(getpid())` impostiamo il campo id della struct consentendo di riconoscere a quale processo appartiene un determinato pacchetto ICMP quando arriva la risposta. Questo valore sarà dato dal process id del processo chiamante (`getpid()`) convertito in big-endian tramite `htons()` (host to network short). Ricordiamo che le reti funzionano sempre in rappresentazione big-endian, per cui se la macchina è big-endian non avrà problemi (in quanto la traduzione con `htons()`

non avrà effetto) ma se è in little-endian, le stringhe che spediamo in rete dovranno essere tradotte in big-endian.

- Con `un.echo.sequence = htons(sequence)` indichiamo il numero di sequenza, ovvero un valore che serve a distinguere i diversi pacchetti all'interno della stessa "sessione" di ping. Anche in questo caso facciamo la conversione in big-endian come visto prima.
- `checksum = 0` e la successiva `compute_checksum()` si occupano di calcolare il checksum ICMP, un valore che serve per verificare l'integrità del pacchetto ricevuto.
- A riga 9 e 10 bisogna fare attenzione: facendo `icmp + 1` andiamo a prendere l'indirizzo alla fine dell'header ICMP (perché `icmp` è grande quanto una struct `icmphdr`) che sarà adesso la parte del payload vero e proprio e lo riempiamo con dei dati casuali (in questo caso delle 'A').

Type(8 bit)	Code(8 bit)	Checksum(16 bit)
Extended Header(32 bit)		
Data/Payload(Variable Length)		

Figure 1: Header ICMP

3.3 Argomento del programma

```
1 int main(int argc, char *argv[]) {  
2     if (argc != 2) {  
3         printf("Utilizzo: %s <indirizzo IP>\n", argv[0]);  
4         exit(1);  
5     }
```

Entriamo adesso nel cuore del programma. La prima istruzione che viene eseguita è un if che controlla se il numero di argomenti passati al programma è corretto. In particolare, `argc` deve essere uguale a 2: il primo argomento (`argv[0]`) è il nome del programma stesso, mentre il secondo (`argv[1]`) deve essere l'indirizzo IP della destinazione.

3.4 Opzione del socket: timer

```
1 // Configura un timeout in ricezione  
2 struct timeval tv;  
3 tv.tv_sec = TIMEOUT_SEC;  
4 tv.tv_usec = 0;  
5 setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
```

Adesso vediamo un elemento già incontrato, ovvero una `struct timeval` che ci serve come argomento per la `setsockopt`: una funzione di libreria che serve per settare le opzioni dei socket. In questo caso stiamo settando un timer che definisce il tempo massimo di attesa per la ricezione di una risposta ICMP. In questo modo, se entro il tempo specificato non viene ricevuta alcuna risposta, la `recvfrom()` restituirà un errore.

3.5 Configurazione della destinazione

```
1 // Configura la destinazione  
2 memset(&dest, 0, sizeof(dest));  
3 dest.sin_family = AF_INET;  
4 dest.sin_addr.s_addr = inet_addr(argv[1]);
```

Dopo aver creato il socket, il programma deve specificare l'indirizzo della destinazione a cui inviare il pacchetto ICMP. Questo viene fatto attraverso la struttura `sockaddr_in`.

- `inet_addr(argv[1])`: converte l'indirizzo IP passato come argomento da stringa a formato numerico binario e lo assegna alla struttura.

In questo modo, la struttura `dest` può essere utilizzata nella chiamata successiva a `sendto()`, che invierà il pacchetto ICMP all'indirizzo specificato.

3.6 Build e invio del pacchetto

```
1 // Costruisce e invia il pacchetto
2 build_icmp_packet(icmp, 1);
3 printf("Invio ping a %s...\n", argv[1]);
4 if (sendto(sock, packet, PACKET_SIZE, 0, (struct sockaddr *)&dest
5 , sizeof(dest)) <= 0) {
6     perror("sendto");
7     close(sock);
8     exit(1);
9 }
```

Dopo aver configurato la destinazione, il programma procede alla costruzione e all'invio del pacchetto ICMP: viene chiamata la funzione che abbiamo descritto precedentemente (`build_icmp_packet()`) passando come argomenti una `struct icmphdr *icmp` e 1 che rappresenta il numero di sequenza. A questo punto viene chiamata la funzione `sendto()` che si occupa di inviare il pacchetto ICMP attraverso il socket precedentemente creato. Se l'invio non va a buon fine (`sendto()` restituisce un valore minore o uguale a zero).

3.7 Ricezione della risposta

```
1  int bytes = recvfrom(sock, recv_buffer, sizeof(recv_buffer), 0, (  
    struct sockaddr *)&src, &src_len);  
2  if (bytes < 0) {  
3      perror("Nessuna risposta");  
4      close(sock);  
5      exit(1);  
6  }
```

Dopo aver inviato il pacchetto ICMP, il programma attende una risposta. Il tutto avviene tramite la funzione `recvfrom()`, che legge i dati ricevuti dal socket. Se la funzione `recvfrom()` restituisce un valore negativo, significa che non è stata ricevuta alcuna risposta entro il tempo limite impostato.

3.8 Analisi della risposta - Header IP

```
1  struct iphdr *ip_hdr = (struct iphdr *)recv_buffer;
```

Andiamo ad analizzare adesso la ricezione della risposta. Ricordiamo che i pacchetti ICMP viaggiano incapsulati all'interno di pacchetti IP, di conseguenza il buffer ricevuto (`recv_buffer`) inizia con l'header IP. La sua definizione è contenuta nel file `ip.h` che si trova nello stesso path di `ip_icmp.h`:

```
1  struct iphdr  
2  {  
3      #if __BYTE_ORDER == __LITTLE_ENDIAN  
4          unsigned int ihl:4;  
5          unsigned int version:4;  
6      #elif __BYTE_ORDER == __BIG_ENDIAN  
7          unsigned int version:4;  
8          unsigned int ihl:4;  
9      #else  
10     # error "Please fix <bits/endian.h>"  
11     #endif  
12     uint8_t tos;  
13     uint16_t tot_len;
```

```

14     uint16_t id;
15     uint16_t frag_off;
16     uint8_t ttl;
17     uint8_t protocol;
18     uint16_t check;
19     uint32_t saddr;
20     uint32_t daddr;
21     /*The options start here. */
22 };

```

Dunque la prima riga effettua anche un cast del puntatore per accedere ai campi dell'header IP.

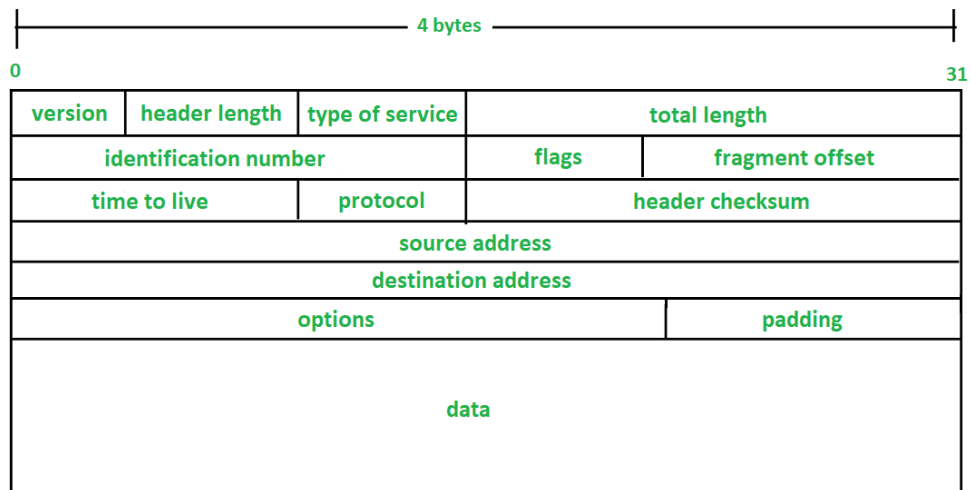


Figure 2: Header IP

3.9 Analisi della risposta - Header ICMP

```

1     struct icmphdr *icmp_reply = (struct icmphdr *) (recv_buffer + (
        ip_hdr->ihl * 4));

```

Dopo aver definito l'header IP dobbiamo passare all'header ICMP (ricordando che è incapsulato nell'header IP). Definiamo la stessa struct che abbiamo visto inizialmente (`icmphdr`) e, per spostarci all'inizio dell'header ICMP, facciamo un calcolo semplice: sommiamo al `recv_buffer` il campo `ihl` e moltiplichiamo per 4.

Il campo `ihl` (internet header length), secondo la definizione dell'RFC 791, rappresenta la lunghezza dell'header di internet in parole da 32 bit, cosa vuol dire? Una parola da 32 bit corrisponde a 4 byte, quindi, per ottenere la lunghezza in byte dell'header IP, dobbiamo moltiplicare il valore di `ihl` per 4.

Questo calcolo ci consente di determinare il punto in cui inizia l'header ICMP all'interno del buffer ricevuto, dato che l'header IP precede l'header ICMP nel pacchetto.

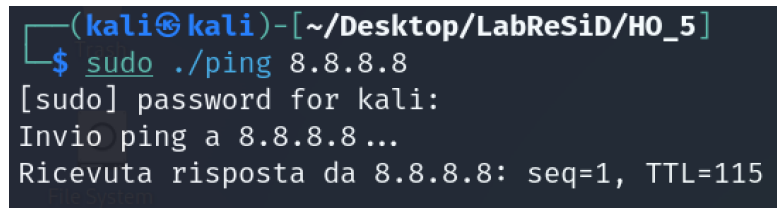
3.10 Analisi della risposta - ECHO REPLY

```
1  if (icmp_reply->type == ICMP_ECHOREPLY &&
2      ntohs(icmp_reply->un.echo.id) == getpid()) {
3      printf("Ricevuta risposta da %s: seq=%d, TTL=%d\n",
4             inet_ntoa(src.sin_addr),
5             ntohs(icmp_reply->un.echo.sequence),
6             ip_hdr->ttl);
7  } else {
8      printf("Risposta non valida\n");
9  }
```

Dopo aver estratto l'header ICMP, verifichiamo se la risposta ricevuta è una ECHO REPLY, quindi una risposta valida per un pacchetto di tipo ECHO REQUEST. L'if nel codice fa effettivamente questo controllo e verifica anche che il valore del campo `id` all'interno dell'header ICMP corrisponda all'identificatore del processo che ha inviato il pacchetto (questa volta utilizziamo `ntohs` - *network to host short* - per convertire il valore dal formato di rete a quello dell'host). Se entrambe le condizioni sono soddisfatte, il programma stampa un messaggio con i dettagli della risposta ICMP ricevuta.

4 Test del programma

Per avviare il programma ci basterà compilare il file .c ed avviarlo con i permessi di amministratore. Il programma utilizza il raw socket, di conseguenza è necessario eseguirlo con i privilegi di root.

A screenshot of a terminal window with a dark background. The prompt is (kali@kali)-[~/Desktop/LabReSid/H0_5]. The user enters the command \$ sudo ./ping 8.8.8.8. The terminal shows the password prompt [sudo] password for kali: and then the output: Invio ping a 8.8.8.8... followed by Ricevuta risposta da 8.8.8.8: seq=1, TTL=115.

```
(kali@kali)-[~/Desktop/LabReSid/H0_5]
$ sudo ./ping 8.8.8.8
[sudo] password for kali:
Invio ping a 8.8.8.8...
Ricevuta risposta da 8.8.8.8: seq=1, TTL=115
```

Figure 3: Test del programma

In questo test ho utilizzato il DNS pubblico di google (8.8.8.8).

5 Analisi del pacchetto - Wireshark

Possiamo fare un'analisi un pò più approfondita per vedere cosa avviene "al di sotto" dell'invio della richiesta, attraverso uno strumento di analisi del traffico di rete: Wireshark. Questo software permette di catturare i pacchetti in transito sulla rete e di esaminarne i dettagli.

5.1 Cattura del pacchetto

Per catturare il pacchetto ICMP generato dal programma:

1. Aprendo Wireshark, selezioniamo l'interfaccia di rete in base alla connessione utilizzata. Nel mio caso eth0;
2. Nella barra in alto per filtrare si inserisce `icmp`, in modo da mostrare solo i pacchetti ICMP ingorando il resto del traffico;
3. A questo punto è possibile eseguire il programma come fatto precedentemente in fase di test;

The image shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for packet capture and analysis. The main display area shows a packet list with two entries:

No.	Time	Source	Destination	Protocol	Length	Info
2	0.142684556	192.168.0.108	8.8.8.8	ICMP	98	Echo (ping) request
3	0.168077163	8.8.8.8	192.168.0.108	ICMP	98	Echo (ping) reply

5.2 Pacchetto inviato

[illegible]

Il type è 8, il code 0, il checksum è corretto (segnalato da wireshark), infine possiamo vedere l'ID restituito dal `getpid()`, il sequence number corretto ed il payload inviato (i caratteri 'A').

5.3 Pacchetto ricevuto

Analogamente possiamo verificare il pacchetto ricevuto:

[illegible]

Figure 6: Pacchetto ricevuto

Il type è 0 in quanto è una echo reply, il checksum è corretto, l'ID è lo stesso presente nel pacchetto inviato, il sequence number idem. Infine, aprendo l'header possiamo verificare anche il TTL (time to live):

```

▼ Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.0.108
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x0000 (0)
  ▶ 000. .... = Flags: 0x0
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 115
  Protocol: ICMP (1)

```

Figure 7: Pacchetto ricevuto

6 Conclusioni

In questo hands-on abbiamo approfondito il funzionamento dei socket raw, con una particolare attenzione verso i pacchetti ICMP. Abbiamo implementato una versione semplificata del comando `ping` che invia pacchetti ICMP ECHO REQUEST e attende delle risposte ICMP ECHO REPLY. La vera potenza dei socket

raw è proprio questa: offrono la possibilità di costruire e inviare pacchetti personalizzati con parametri modificabili a nostro piacimento. Questo tipo di manipolazione ci ha permesso di avere un forte controllo sul livello di rete e ha dato la possibilità di vedere (anche attraverso Wireshark) come si comporta un pacchetto in fase di trasmissione e ricezione.