D Roshik
187213

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/ip.h>

#define __FAVOR_BSD
#include <netinet/udp.h>
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <ifaddrs.h>

typedef u_int32_t ip4_t;

#define DHCP_CHADDR_LEN 16
#define DHCP_SNAME_LEN  64
#define DHCP_FILE_LEN   128

typedef struct dhcp
{
   u_int8_t    opcode;
   u_int8_t    htype;
   u_int8_t    hlen;
   u_int8_t    hops;
   u_int32_t   xid;
   u_int16_t   secs;
   u_int16_t   flags;
   ip4_t       ciaddr;
   ip4_t       yiaddr;
   ip4_t       siaddr;
   ip4_t       giaddr;
   u_int8_t    chaddr[DHCP_CHADDR_LEN];
   char        bp_sname[DHCP_SNAME_LEN];
   char        bp_file[DHCP_FILE_LEN];
   uint32_t    magic_cookie;
   u_int8_t    bp_options[0];
} dhcp_t;

#define DHCP_BOOTREQUEST                1
#define DHCP_BOOTREPLY                  2

#define DHCP_HARDWARE_TYPE_10_EHTHERNET     1
```

```
#define MESSAGE_TYPE_PAD                0
#define MESSAGE_TYPE_REQ_SUBNET_MASK    1
#define MESSAGE_TYPE_ROUTER             3
#define MESSAGE_TYPE_DNS                6
#define MESSAGE_TYPE_DOMAIN_NAME        15
#define MESSAGE_TYPE_REQ_IP             50
#define MESSAGE_TYPE_DHCP               53
#define MESSAGE_TYPE_PARAMETER_REQ_LIST 55
#define MESSAGE_TYPE_END                255

#define DHCP_OPTION_DISCOVER            1
#define DHCP_OPTION_OFFER               2
#define DHCP_OPTION_REQUEST             3
#define DHCP_OPTION_PACK                4

typedef enum {
   VERBOSE_LEVEL_NONE,
   VERBOSE_LEVEL_ERROR,
   VERBOSE_LEVEL_INFO,
   VERBOSE_LEVEL_DEBUG,
}verbose_level_t;

#define PRINT(verbose_level, fmt, args...)                      \
do{                                                             \
   if( verbose_level <= program_verbose_level ) {               \
      if ( verbose_level == VERBOSE_LEVEL_DEBUG ) {             \
         printf("%s:%d:%s::",    __FILE__, __LINE__, __FUNCTION__);  \
      }                                                         \
      printf(fmt, ##args);                                      \
      printf("\n");                                             \
   }                                                            \
}while(0)

#define DHCP_SERVER_PORT    67
#define DHCP_CLIENT_PORT    68

#define DHCP_MAGIC_COOKIE   0x63825363

verbose_level_t program_verbose_level = VERBOSE_LEVEL_DEBUG;
pcap_t *pcap_handle;
u_int32_t ip;

static int
get_mac_address(char *dev_name, u_int8_t *mac)
{
#ifdef __linux__
   struct ifreq s;
   int fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP);
   int result;

   strcpy(s.ifr_name, dev_name);
   result = ioctl(fd, SIOCGIFHWADDR, &s);
```

```
    close(fd);
    if (result != 0)
        return -1;

    memcpy((void *)mac, s.ifr_addr.sa_data, 6);
    return 0;
#else
    struct ifaddrs *ifap, *p;

    if (getifaddrs(&ifap) != 0)
        return -1;

    for (p = ifap; p; p = p->ifa_next)
    {
        /* Check the device name */
        if ((strcmp(p->ifa_name, dev_name) == 0) &&
            (p->ifa_addr->sa_family == AF_LINK))
        {
            struct sockaddr_dl* sdp;

            sdp = (struct sockaddr_dl*) p->ifa_addr;
            memcpy((void *)mac, sdp->sdl_data + sdp->sdl_nlen, 6);
            break;
        }
    }
    freeifaddrs(ifap);
#endif

    return 0;
}

/*
 * Return checksum for the given data.
 * Copied from FreeBSD
 */
static unsigned short
in_cksum(unsigned short *addr, int len)
{
    register int sum = 0;
    u_short answer = 0;
    register u_short *w = addr;
    register int nleft = len;
    /*
     * Our algorithm is simple, using a 32 bit accumulator (sum), we add
     * sequential 16 bit words to it, and at the end, fold back all the
     * carry bits from the top 16 bits into the lower 16 bits.
     */
    while (nleft > 1)
    {
        sum += *w++;
        nleft -= 2;
    }
```

```
    /* mop up an odd byte, if necessary */
    if (nleft == 1)
    {
        *(u_char *)(&answer) = *(u_char *) w;
        sum += answer;
    }
    /* add back carry outs from top 16 bits to low 16 bits */
    sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
    sum += (sum >> 16);             /* add carry */
    answer = ~sum;              /* truncate to 16 bits */
    return (answer);
}

/*
 * This function will be called for any incoming DHCP responses
 */
static void
dhcp_input(dhcp_t *dhcp)
{
    if (dhcp->opcode != DHCP_OPTION_OFFER)
        return;

    /* Get the IP address given by the server */
    ip = ntohl(dhcp->yiaddr);

    /* We are done - lets break the loop */
    pcap_breakloop(pcap_handle);
}

/*
 * UDP packet handler
 */
static void
udp_input(struct udphdr * udp_packet)
{
    /* Check if there is a response from DHCP server by checking the source Port */
    if (ntohs(udp_packet->uh_sport) == DHCP_SERVER_PORT)
        dhcp_input((dhcp_t *)((char *)udp_packet + sizeof(struct udphdr)));
}

/*
 * IP Packet handler
 */
static void
ip_input(struct ip * ip_packet)
{
    /* Care only about UDP - since DHCP sits over UDP */
    if (ip_packet->ip_p == IPPROTO_UDP)
        udp_input((struct udphdr *)((char *)ip_packet + sizeof(struct ip)));
}

/*
```

```
 * Ethernet packet handler
 */
static void
ether_input(u_char *args, const struct pcap_pkthdr *header, const u_char *frame)
{
    struct ether_header *eframe = (struct ether_header *)frame;

    PRINT(VERBOSE_LEVEL_DEBUG, "Received a frame with length of [%d]", header->len);


    if (htons(eframe->ether_type) == ETHERTYPE_IP)
        ip_input((struct ip *)(frame + sizeof(struct ether_header)));
}

/*
 * Ethernet output handler - Fills appropriate bytes in ethernet header
 */
static void
ether_output(u_char *frame, u_int8_t *mac, int len)
{
    int result;
    struct ether_header *eframe = (struct ether_header *)frame;

    memcpy(eframe->ether_shost, mac, ETHER_ADDR_LEN);
    memset(eframe->ether_dhost, -1,  ETHER_ADDR_LEN);
    eframe->ether_type = htons(ETHERTYPE_IP);

    len = len + sizeof(struct ether_header);

    /* Send the packet on wire */
    result = pcap_inject(pcap_handle, frame, len);
    PRINT(VERBOSE_LEVEL_DEBUG, "Send %d bytes\n", result);
    if (result <= 0)
        pcap_perror(pcap_handle, "ERROR:");
}

/*
 * IP Output handler - Fills appropriate bytes in IP header
 */
static void
ip_output(struct ip *ip_header, int *len)
{
    *len += sizeof(struct ip);

    ip_header->ip_hl = 5;
    ip_header->ip_v = IPVERSION;
    ip_header->ip_tos = 0x10;
    ip_header->ip_len = htons(*len);
    ip_header->ip_id = htons(0xffff);
    ip_header->ip_off = 0;
    ip_header->ip_ttl = 16;
    ip_header->ip_p = IPPROTO_UDP;
```

```
    ip_header->ip_sum = 0;
    ip_header->ip_src.s_addr = 0;
    ip_header->ip_dst.s_addr = 0xFFFFFFFF;

    ip_header->ip_sum = in_cksum((unsigned short *) ip_header, sizeof(struct ip));
}

/*
 * UDP output - Fills appropriate bytes in UDP header
 */
static void
udp_output(struct udphdr *udp_header, int *len)
{
    if (*len & 1)
        *len += 1;
    *len += sizeof(struct udphdr);

    udp_header->uh_sport = htons(DHCP_CLIENT_PORT);
    udp_header->uh_dport = htons(DHCP_SERVER_PORT);
    udp_header->uh_ulen = htons(*len);
    udp_header->uh_sum = 0;
}

/*
 * DHCP output - Just fills DHCP_BOOTREQUEST
 */
static void
dhcp_output(dhcp_t *dhcp, u_int8_t *mac, int *len)
{
    *len += sizeof(dhcp_t);
    memset(dhcp, 0, sizeof(dhcp_t));

    dhcp->opcode = DHCP_BOOTREQUEST;
    dhcp->htype = DHCP_HARDWARE_TYPE_10_EHTHERNET;
    dhcp->hlen = 6;
    memcpy(dhcp->chaddr, mac, DHCP_CHADDR_LEN);

    dhcp->magic_cookie = htonl(DHCP_MAGIC_COOKIE);
}

/*
 * Adds DHCP option to the bytestream
 */
static int
fill_dhcp_option(u_int8_t *packet, u_int8_t code, u_int8_t *data, u_int8_t len)
{
    packet[0] = code;
    packet[1] = len;
    memcpy(&packet[2], data, len);

    return len + (sizeof(u_int8_t) * 2);
}
```

```
/*
 * Fill DHCP options
 */
static int
fill_dhcp_discovery_options(dhcp_t *dhcp)
{
    int len = 0;
    u_int32_t req_ip;
    u_int8_t parameter_req_list[] = {MESSAGE_TYPE_REQ_SUBNET_MASK,
MESSAGE_TYPE_ROUTER, MESSAGE_TYPE_DNS, MESSAGE_TYPE_DOMAIN_NAME};
    u_int8_t option;

    option = DHCP_OPTION_DISCOVER;
    len += fill_dhcp_option(&dhcp->bp_options[len], MESSAGE_TYPE_DHCP, &option,
sizeof(option));
    req_ip = htonl(0xc0a8010a);
    len += fill_dhcp_option(&dhcp->bp_options[len], MESSAGE_TYPE_REQ_IP, (u_int8_t
*)&req_ip, sizeof(req_ip));
    len += fill_dhcp_option(&dhcp->bp_options[len],
MESSAGE_TYPE_PARAMETER_REQ_LIST, (u_int8_t *)&parameter_req_list,
sizeof(parameter_req_list));
    option = 0;
    len += fill_dhcp_option(&dhcp->bp_options[len], MESSAGE_TYPE_END, &option,
sizeof(option));

    return len;
}

/*
 * Send DHCP DISCOVERY packet
 */
static int
dhcp_discovery(u_int8_t *mac)
{
    int len = 0;
    u_char packet[4096];
    struct udphdr *udp_header;
    struct ip *ip_header;
    dhcp_t *dhcp;

    PRINT(VERBOSE_LEVEL_INFO, "Sending DHCP_DISCOVERY");

    ip_header = (struct ip *)(packet + sizeof(struct ether_header));
    udp_header = (struct udphdr *)(((char *)ip_header) + sizeof(struct ip));
    dhcp = (dhcp_t *)(((char *)udp_header) + sizeof(struct udphdr));

    len = fill_dhcp_discovery_options(dhcp);
    dhcp_output(dhcp, mac, &len);
    udp_output(udp_header, &len);
    ip_output(ip_header, &len);
    ether_output(packet, mac, len);
```

```
    return 0;
}

int
main(int argc, char *argv[])
{
    int result;
    char errbuf[PCAP_ERRBUF_SIZE];
    char *dev;
    u_int8_t mac[6];

    if (argc < 2 || (strcmp(argv[1], "-h") == 0))
    {
        printf("Usage: %s <interface>\n", argv[0]);
        return 0;
    }
    dev = argv[1];

    /* Get the MAC address of the interface */
    result = get_mac_address(dev, mac);
    if (result != 0)
    {
        PRINT(VERBOSE_LEVEL_ERROR, "Unable to get MAC address for %s", dev);
        return -1;
    }

    /* Open the device and get pcap handle for it */
    pcap_handle = pcap_open_live(dev, BUFSIZ, 0, 10, errbuf);
    if (pcap_handle == NULL)
    {
        PRINT(VERBOSE_LEVEL_ERROR, "Couldn't open device %s: %s", dev, errbuf);
        return -1;
    }

    /* Send DHCP DISCOVERY packet */
    result = dhcp_discovery(mac);
    if (result)
    {
        PRINT(VERBOSE_LEVEL_ERROR, "Couldn't send DHCP DISCOVERY on device %s:
%s", dev, errbuf);
        goto done;
    }

    ip = 0;
    PRINT(VERBOSE_LEVEL_INFO, "Waiting for DHCP_OFFER");
    /* Listen till the DHCP OFFER comes */
    pcap_loop(pcap_handle, -1, ether_input, NULL);
    printf("Got IP %u.%u.%u.%u\n", ip >> 24, ((ip << 8) >> 24), (ip << 16) >> 24, (ip << 24) >>
24);

done:
```

```
    pcap_close(pcap_handle);

    return result;
}
```