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;

}