Part I
Network

1 pcap

まず pcap.hを使用する.pcap.hをインストールするにはターミナル上で
sudo apt-get install libpcap-dev
と入力する. pcap.hを使用したソースコードをコンパイルする際は
gcc -lpcap -Wall ソースファイル.c -o 実行ファイル
と入力する. できた実行ファイルは管理者権限下で実行する必要があり,
実行の際には
sudo ./実行ファイル名
とターミナルに入力する.

Listing 1: pcap_loopに使用するcallback関数の仕様

```
void callback
(u_char *args,const struct pcap_pkthdr *cap_header, const u_char *packet)
```

pcap_pkthdr構造体は以下のようになっている.

Listing 2: pcap_pkthdr構造体

```
struct pcap_pkthdr
{
  struct timeval ts;
  buf_u_int32 caplen;
  buf_uint32 len;
}
```

2 Ethernet

これは net/inet.hで定義されている.

Listing 3: ethernet_header構造体

```
struct ether_header
{
  uint8_t  ether_dhost[ETH_ALEN]; /* destination eth addr */
  uint8_t  ether_shost[ETH_ALEN]; /* source ether addr */
  uint16_t ether_type;                 /* packet type ID field */
} __attribute__ ((__packed__));
```

3 中の __attribute__ ((__packed__))は構造体の隙間を詰めるものであり,本質で
はない. ether_typeは以下のように定義される.
ETHERTYPE_PUP
ETHERTYPE_IP: IPv4
ETHERTYPE_ARP:Address Resolution Protocol
ETHERTYPE_AT:Apple Talk Protocol
ETHERTYPE_AARP:Apple Talk ARP
ETHERTYPE_REVARP:Reverse ARP

ETHERTYPE_VLAN: VLAN tagging
ETHERTYPE_IPX:IPX
ETHERTYPE_IPv6:IPv6
ETHERTYPE_LOOPBACK:loopback
なお,実装の際には短整数をネットワークバイトオーダーからホストバイトオーダーへと変換させる関数htonsを使用して ntohs(ether_type)とする必要がある. 次のような実装を考えることができる.

Listing 4: hoge

```
switch(ntohs(eth_hdr->ether_type)){
    case ETHERTYPE_PUP:
        printf("(Xerox PUP)\n");
        break;

    case ETHERTYPE_IP:
        printf("(IP)\n");
        break;

    case ETHERTYPE_ARP:
        printf("Address resolution\n");
        break;

    case ETHERTYPE_AT:
        printf("Appe Talk protocol\n");
        break;

    case ETHERTYPE_AARP:
        printf("Appe Talk ARRP\n");
        break;

    case ETHERTYPE_REVARP:
        printf("Reverse ARP\n");
        break;

    case ETHERTYPE_VLAN:
        printf("VLAN tagging\n");
        break;

    case ETHERTYPE_IPX:
        printf("IPX\n");
        break;

    case ETHERTYPE_IPV6:
        printf("IPv6\n");
        break;

    case ETHERTYPE_LOOPBACK:
      printf("loop back\n");
      break;
```

```
        default:
            printf("unknown");
            break;
```

配列ether_dhostとether_shostは次のように書式指定子を用いて16進数に書き換えることが必要.

Listing 5: hoge

```
dmac[18]={0};
u_char *hwaddr = ether_dhost;
snprintf(dmac,sizeof(dmac),"%02x:%02x:%02x:%02x:%02x:%02x",
                        hwaddr[0],hwaddr[1],hwaddr[2],
                        hwaddr[3],hwaddr[4],hwaddr[5]);
```

以上をまとめるとpcap_loopに次のようなcallback関数を作ることができる.

Listing 6: hoge

```
char * convmac_tostr(u_char *hwaddr,char *mac,size_t size){
  snprintf(mac,size,"%02x:%02x:%02x:%02x:%02x:%02x",
                        hwaddr[0],hwaddr[1],hwaddr[2],
                        hwaddr[3],hwaddr[4],hwaddr[5]);
  return mac;
}

void start_pktfunc( u_char *user,
                    const struct pcap_pkthdr *h ,
                    const u_char *p
                     ){
  char dmac[18] = {0};
  char smac[18] = {0};
  struct ether_header *eth_hdr = (struct ether_header *)p;

  printf("ether header----------\n");
  printf("dest mac %s\n",convmac_tostr(eth_hdr->ether_dhost,dmac,sizeof(dmac)));
  printf("src mac %s\n",convmac_tostr(eth_hdr->ether_shost,smac,sizeof(smac)));
  printf("ether type %x\n\n",ntohs(eth_hdr->ether_type));
    switch(ntohs(eth_hdr->ether_type)){


    switch(ntohs(eth_hdr->ether_type)){
        case ETHERTYPE_PUP:
            printf("(Xerox PUP)\n");
            break;

        case ETHERTYPE_IP:
            printf("(IP)\n");
            break;
```

```
        case ETHERTYPE_ARP:
            printf("Address resolution\n");
            break;

        case ETHERTYPE_AT:
            printf("Appe Talk protocol\n");
            break;

        case ETHERTYPE_AARP:
            printf("Appe Talk ARRP\n");
            break;

        case ETHERTYPE_REVARP:
            printf("Reverse ARP\n");
            break;

        case ETHERTYPE_VLAN:
            printf("VLAN tagging\n");
            break;


        case ETHERTYPE_IPX:
            printf("IPX\n");
            break;

        case ETHERTYPE_IPV6:
            printf("IPv6\n");
            break;

        case ETHERTYPE_LOOPBACK:
          printf("loop back\n");
          break;

        default:
            printf("unknown");
            break;
    }


}
```

packetキャプチャにおいてプロミスキャスモードに設定した上で,etherヘッダーを覗き見ることでターゲット端末のMACアドレスを調べることが可能になる.(etherヘッダは暗号化できないため)

3 Macアドレス ioctlの設定

ifreq構造体を用いる. ifreq構造体は/linux/if.hで定義されている. 詳しくはman コマンドで確認できる. 一部の内容を次に記す.

Listing 7: ifreq

```
struct ifreq
{
#define IFHWADDRLEN     6
        union
        {
                char    ifrn_name[IFNAMSIZ];            /* if name, e.g. "en0"
        } ifr_ifrn;

        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                struct  sockaddr ifru_broadaddr;
                struct  sockaddr ifru_netmask;
                struct  sockaddr ifru_hwaddr;
                short   ifru_flags;
                int     ifru_ivalue;
                int     ifru_mtu;
                struct  ifmap ifru_map;
                char    ifru_slave[IFNAMSIZ];   /* Just fits the size */
                char    ifru_newname[IFNAMSIZ];
                void *  ifru_data;
                struct  if_settings ifru_settings;
        } ifr_ifru;
};
#define ifr_name        ifr_ifrn.ifrn_name      /* interface name       */
#define ifr_hwaddr      ifr_ifru.ifru_hwaddr    /* MAC address          */
#define ifr_addr        ifr_ifru.ifru_addr        /* address            */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-p lnk */
#define ifr_broadaddr   ifr_ifru.ifru_broadaddr /* broadcast address    */
#define ifr_netmask     ifr_ifru.ifru_netmask    /* interface net mask  */
#define ifr_flags       ifr_ifru.ifru_flags     /* flags                */
#define ifr_metric      ifr_ifru.ifru_ivalue    /* metric               */
#define ifr_mtu         ifr_ifru.ifru_mtu       /* mtu                  */
#define ifr_map         ifr_ifru.ifru_map       /* device map           */
#define ifr_slave       ifr_ifru.ifru_slave     /* slave device         */
#define ifr_data        ifr_ifru.ifru_data              /* for use by interface
#define ifr_ifindex     ifr_ifru.ifru_ivalue    /* interface index      */
#define ifr_bandwidth   ifr_ifru.ifru_ivalue   /* link bandwidth        */
#define ifr_qlen        ifr_ifru.ifru_ivalue    /* Queue length        */
#define ifr_newname     ifr_ifru.ifru_newname    /* New name           */
#define ifr_settings    ifr_ifru.ifru_settings  /* Device/proto settings*/
```

注目すべきポイントは ifreq構造体に ifr_hwaddr要素がある, これは厳密にはifru_hwaddrとして宣言されているが, #defineの部分で ifr_ifru.ifru_hwaddrにすると書いてある(# define ifr_hwaddr ifr_ifru.ifru_hwaddr の部分.) よく見ると ifr_hwaddr構造体は　struct sockaddr ifr_hwaddr;　となっており,これはsockaddr構造体で宣言されている.つまりMACアドレスを見るにはsockaddr構造体としてアクセスする必要がある. sockaddr構造体は /include/bits/socket.hに次のように定義されている.

Listing 8: sockaddr

```
struct sockaddr
  {
    __SOCKADDR_COMMON (sa_);     /* Common data: address family and length.
*/
    char sa_data[14];            /* Address data.  */
  };
```

このsa_data[14]のうち, sa_data[0] ~ sa_data[5]の部分にMACアドレスが格納される. sa_data[6] ~ sa_data[13]は0が入る.

3.1 MACアドレスの表示

インターフェースの操作には　ifreq構造体と ioctlシステムコールを用いる. ioctlシステムコールを用いるには事前に,リンクレイヤを扱うためにSOCK_RAWを指定したソケットディスクリプタが必要となる. ifreqメンバの一つであるifr_nameにはインターフェース名が入り, これはターミナル上で iifconfigコマンドを用いて表示される(en0, eth0など) 続いて iosctlシステムコールを用いる. SIOCGIFHWADDR フラグを用いてインタフェースのMACアドレスを取得する.このフラグを設定するとifr_hwaddrメンバにアドレスを格納してくれるようになる. 以下がMACアドレスを表示するプログラム全体である.

Listing 9: showMACaddress

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/ioctl.h>
#include<sys/socket.h>
#include<net/if.h>
#include<net/ethernet.h>
#include<netpacket/packet.h>
#include<arpa/inet.h>

int main(void)
{
  struct ifreq ifreq;
  int s;
  if( (s=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL)))<0 )
  {
    perror("socket"); //creating a raw socket in advance
    exit(1);
  }

  memset(&ifreq, 0, sizeof(struct ifreq));
  strncpy(ifreq.ifr_name, "enp0s3", sizeof(ifreq.ifr_name)-1);
  if(ioctl(s, SIOCGIFHWADDR, &ifreq)!=0) perror("ioctl");

  printf("%02x:%02x:%02x:%02x:%02x:%02x \n",
  (unsigned char)ifreq.ifr_hwaddr.sa_data[0],
  (unsigned char)ifreq.ifr_hwaddr.sa_data[1],
```

```
    (unsigned char)ifreq.ifr_hwaddr.sa_data[2],
    (unsigned char)ifreq.ifr_hwaddr.sa_data[3],
    (unsigned char)ifreq.ifr_hwaddr.sa_data[4],
    (unsigned char)ifreq.ifr_hwaddr.sa_data[5]);

  close(s);
  return 0;
}
```

3.2 MACアドレスの偽装

MACアドレスはデバイス固有の番号であり,NICカードに記載されており,書き換えはできない. しかし,プログラムにより, OS側に対して偽のMACアドレスを認識できるように書き換えることはできる. このことからMACアドレスだけを用いるのはセキュリティ面では不十分. このプログラムを試すのは仮想環境で行うことを勧める. MACアドレスの書き換えは ifreq.ifr_hwaddr.sa_data[15]配列を書き換えた後に ioctlのフラグを SIOCSIFHWADDRにするだけである. なお,MACアドレスは16進数の6桁からなるが,実際に書き換えることができるのは下位3桁であり,上位3桁はOUI(ベンダーコード)と呼ばれ,デバイスを作った製造元の値が入るため, この部分の書き換えになると ioctlでエラーが発生する. (もしかしたらうまく書き換える方法があるのかもしれない) 以下がソースコードである.

Listing 10: changeMACaddress

```
#include<stdio.h>
#include<string.h>
#include<sys/ioctl.h>
#include<sys/socket.h>
#include<net/if.h>
#include<net/ethernet.h>
#include<arpa/inet.h>

int main(void)
{
  struct ifreq ifreq;

  int s;
  if( (s=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL)))<0)
  {//creating a raw socket
    perror("[-]socket()");
    exit(1);
  }

  memset(&ifreq, 0, sizeof(struct ifreq));
  strncpy(ifreq.ifr_name, "eth1", sizeof(ifreq.ifr_name)-1);
  if(ioctl(s, SIOCGIFHWADDR, &ifreq)<0) perror("[-]ioctl()");

  printf("Mac Address\n");
  printf("before <%02x:%02x:%02x:%02x:%02x:%02x>\n",
```

```
        (unsigned char)ifreq.ifr_hwaddr.sa_data[0],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[1],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[2],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[3],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[4],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[5]);


  ifreq.ifr_hwaddr.sa_data[0] = 0x08;//OUI
  ifreq.ifr_hwaddr.sa_data[1] = 0x00;//OUI
  ifreq.ifr_hwaddr.sa_data[2] = 0x27;//OUI
  ifreq.ifr_hwaddr.sa_data[3] = 0xaa;
  ifreq.ifr_hwaddr.sa_data[4] = 0xaa;
  ifreq.ifr_hwaddr.sa_data[5] = 0xaa;
  if(ioctl(s, SIOCSIFHWADDR, &ifreq)<0)perror("[-]ioctl()");

  printf("after <%02x:%02x:%02x:%02x:%02x:%02x>\n",
        (unsigned char)ifreq.ifr_hwaddr.sa_data[0],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[1],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[2],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[3],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[4],
        (unsigned char)ifreq.ifr_hwaddr.sa_data[5]);

  return 0;
}
```

4 IPアドレス ioctlの設定

ioctlシステムコールを用いる,フラグに次の二つを用いる. SIOCGIFADDRで指定したインターフェスのIPアドレスを取得し,その情報をifreq構造体のifr_addrメンバに格納する. SIOCSIFADDRで指定したインターフェスのIPアドレスを取得し,その情報をifreq構造体のifr_addrメンバに書き込む. ioctlを用いるためにはRAWのソケットディスクリプタをあらかじめ生成する必要がある.

Listing 11: changeIPaddress

```
#include<stdio.h>
#include<sys/ioctl.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<unistd.h>
#include<string.h>
#include<arpa/inet.h>
#include<net/if.h>

int main(void){
  struct ifreq ifreq;
  struct sockaddr_in *sin;
  char buf[INET_ADDRSTRLEN];
```

```
  int s;
  if( ( s=socket(PF_PACKET, SOCK_RAW, 0) )<0 ){
    perror("[-]socket()");
    exit(1);
  }

  //get IP address
  memset(&ifreq, 0, sizeof(struct ifreq));
  strncpy(ifreq.ifr_name, "eth1", sizeof(ifreq.ifr_name)-1);
  if(ioctl(s, SIOCGIFADDR, &ifreq)<0){
    perror("[-]ioctl(SIOCGIFADDR)");
    close(soc);
    return 0;
  }


  sin = (struct sockaddr_in *)&ifreq.ifr_addr;
  inet_ntop(AF_INET, &sin->sin_addr.s_addr, buf, sizeof(buf));
  printf("[+]Before: %s\n", buf);

  // write out IP address
  sin->sin_addr.s_addr = inet_addr("10.24.94.100");
  if(ioctl(s, SIOCSIFADDR, &ifreq)<0){
    perror("[-]ioctl(SIOCSIFADDR)");
    close(s);
    return 0;
  }
  else printf("[+]changed IP address\n");

  close(s);
  return 0;
}
```

IPアドレスの変更だけでなく, サブネットマスクとブロードキャストアドレスを変更することも行う必要がある, ioctlのフラグとして次の二つを用いる. SIOCSIFBRDADDR これはifreq構造体のifr_broadaddrメンバに格納された情報を指定したインターフェスのブロードキャストアドレスとして書き込む(更新する) SIOCSIFNETMASK これはifreq構造体のifr_netmaskメンバに格納された情報を指定したインターフェスのサブネットマスクとして書き込む(更新する) コードを追加.

Listing 12: changeSubnetmaskandBroadcastaddress

```
sin = (struct sockaddr_in *)&ifreq.ifr_broadaddr;
  sin->sin_addr.s_addr = inet_addr("10.24.95.255");
  if(ioctl(soc, SIOCSIFBRDADDR, &ifreq)<0){
    perror("[-]ioctl(SIOCSIFBRDADDR)");
    close(soc);
    return 0;
  }else printf("[+]set broadcast address\n");
```

```
  sin = (struct sockaddr_in *)&ifreq.ifr_netmask;
  sin->sin_addr.s_addr = inet_addr("255.255.240.0");
  if(ioctl(soc, SIOCSIFNETMASK, &ifreq)<0){
    perror("[-]iocrl(SIOCSIFNETMASK)");
    close(soc);
    return 0;
  }else printf("[+]set network mask\n");
```

5 ファイル処理

openシステムコール,stat,fstatシステムコールを用いる. stat関数は次のような書式になっている.

int stat(const char *path, struct stat *buffer)

ファイルのありかをpathに格納し, stat構造体の変数 buffに格納する.

一方でfstat関数の書式は次のようになっている.

int fstat(int fd, struct stat *buffer)

fdはファイルディスクリプタである. ファイルディスクリプタfdを用いて

fd=open(path,O_RDONLY,0)

でファイルを開くことができる. 両者ともに構造体stat のインスタンスを作り, stat関数に参照と,ファイルディスクリプタを渡す.

stat構造体は以下のようになっている. 詳しくはmanコマンドで参照できる.

Listing 13: structStat

```
struct stat {
    dev_t     st_dev;     /* device ID */
    ino_t     st_ino;     /* inode num */
    mode_t    st_mode;    /* access */
    nlink_t   st_nlink;   /* the number of hard link */
    uid_t     st_uid;     /* user  ID */
    gid_t     st_gid;     /* user group ID */
    dev_t     st_rdev;    /* device ID  */
    off_t     st_size;    /* total size (byte unit) */
    blksize_t st_blksize; /* file I/O block size */
    blkcnt_t  st_blocks;  /* alloacted 512B block num */
};
```

このstat構造体のメンバの一つである st_sizeにはファイルの大きさを格納されるため,ファイルの大きさ取得にはこれを用いる. 以下のコードはpath="/Users/fujiwara/desktop/make-practice/makeserver/file/test.txt"; に存在するtest.txtのファイルの大きさを取り出す関数である. fstatには ファイルディスクリプタfd , statにはファイルのありかのpathを指定している. ともにstat構造体の参照を引数に取り入れていることに注目する.

Listing 14: fileexample

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

```
#include<sys/stat.h>

int main(){
    struct stat statBuf;

    const char path[]
        ="/Users/fujiwara/desktop/make-practice/makeserver/file/test.txt";

    int size=0;
    if(stat(path,&statBuf)==0){
        size = statBuf.st_size;
    }

    fprintf(stdout,"%d\n",size);

    int fd;
    if( (fd = open(path,O_RDONLY,0) )<0){
        fprintf(stdout,"Couldn't open file");
        exit(1);
    }

    if(fstat(fd,&statBuf)==-1){
        fprintf(stdout,"Couldn't open file");
        exit(1);
    }
    size = statBuf.st_size;

    fprintf(stdout,"%d\n",size);

    return 0;

}
```

このように socket()関数と同じような手順でファイルを扱うことができる点に注目してほしい.