рыбная ловля в локальной сети – sniffering

крис касперски ака мыщъх no-email

сетевой трафик содержит уйму интересного (пароли, номера кредитных карт, конфиденциальную переписку) и все это может стать вашим, если забросить в сеть сниффер. перехват информации – занятие настолько же интересное, насколько и небезопасное. популярные снифферы никак не скрывают своего присутствия и легко обнаруживаются администраторами. хотите избежать расправы? пишите свой собственный сниффер, а эта статья подскажет как.

цели и методы атаки

Сниффером (от английского *sniff* – вынюхивать) называют утилиты для перехвата сетевого трафика, адресованному другому узлу или в более общем случае – всего доступного трафика проходящего или не проходящего через данный хост. Большинство синфферов представляют собой вполне легальные средства мониторинга и не требуют установки дополнительного оборудования. Тем не менее, их использование в общем случае незаконно и требует соответствующих полномочий (например, монтер может подключаться к телефонным проводам, а вы – нет). Кстати говоря, слово "sniffer" является торговой маркой компании Network Associates, распространяющий сетевой анализатор "Sniffer(r) Network Analyzer". Использовать этот термин в отношении других программ с юридической точки зрения незаконно, но… XEROX тоже торговая марка, но в просторечии все копировальные аппараты независимо от их происхождения называют ксероксами и никто от этого еще не пострадал.

Объектом атаки может выступать: локальная сеть (как хабовой, так и свитчевой архитекторы), глобальная сеть (даже при модемном подключении!), спутниковый и мобильный Интернет, беспроводные средства связи (ИК, голубой зуб) и т. д. Главным образом мы будем говорить о локальных сетях, а все остальные рассмотрим лишь кратко, т. к. они требуют совсем другого подхода.

По методу воздействия на жертву существующие атаки можно разделит на два типа: пассивные и активные. Пассивный снифферинг позволяет перехватывать лишь ту часть трафика, которая физически проходит через данный узел. Все остальное может быть получено лишь путем прямого вмешательства в сетевые процессы (модификация таблиц маршрутизации, отправка подложных пакетов и т. д.).

>>>> врезка хабы и ухабы

Хабом (от английского *hub* – ступица колеса) или, иначе говоря, **концентратором**, – называют многопортовой *репитор* (*повторитель*). Получив данные на один из портов, репитор немедленно перенаправляет их на остальные порты. В коаксиальных сетях репитор не является обязательным компонентом и при подключении методом общей шины можно обойтись без него (см. рис. 1)

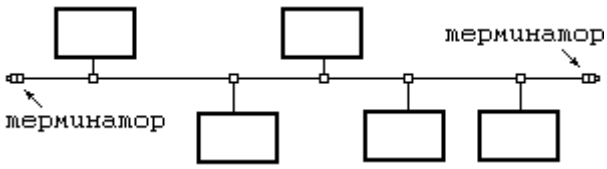

Рисунок 1 топология общей шины

В сетях на витой паре и коаксиальных сетях, построенных по топологии звезда, репитор присутствует изначально (см. рис. 2).

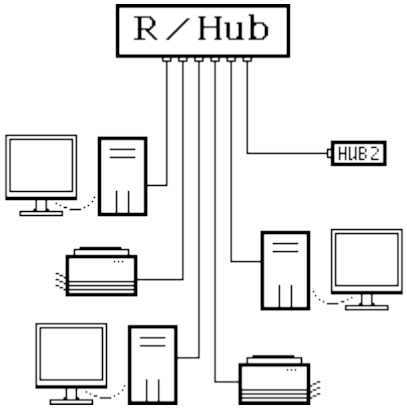

Рисунок 2 топология звезда

Свитч (от английского *switch* – коммутатор), так же называемый интеллектуальным хабом/маршрутизатором, представляет собой разновидность репитора, передающего данные только на порт того узла, которому они адресованы, что предотвращает возможность перехвата трафика (во всяком случае теоретически).

пассивный перехват трафика

Локальная сеть уже давно стала синонимом слова Ethernet, а в Ethernet-сетях, построенных по топологии общей шины, каждый испускаемый пакет доставляется всем участникам сети. Сетевая карта на аппаратном уровне анализирует заголовки пакетов и сверяет свой физический адрес (так же называемый MAC-адресом) с адресом, прописанном в Ethernet-заголовке, передавая на IP-уровень только "свои" пакеты.

Для перехвата трафика карту необходимо перевести в ***неразборчивый*** (***promiscuous***) режим, в котором на IP-уровень передается все принятые пакеты. Неразборчивый режим поддерживает подавляющее большинство стандартных карт, провоцируя излишне любопытных пользователей на проникновение в интимную жизнь остальных участников сети.

Переход на витую пару с неинтеллектуальным хабом ничего не меняет – отправляемые пакеты дублируется на каждый выход хаба и грабятся по той же самой схеме. Интеллектуальный хаб, самостоятельно анализирующий заголовки пакетов и доставляющий их только тем узлам, для которых они предназначены, предотвращает пассивный перехват, вынуждая атакующего переходить к активным действиям, разговор о которых нас ждет впереди.

Таким образом, **для реализации пассивного сниффера мы должны перевести сетевую карту в неразборчивый режим и создать сырой (raw) сокет, дающий доступ ко всему, что валится на данный IP-интерфейс**. Обычные сокеты для этой цели не подходят, поскольку принимают только явно адресованные им пакеты, поступающие на заданный порт. Легальные снифферы чаще всего используют кросс-платформенную библиотеку **libcap**, однако, настоящие хакеры предпочитают разрабатывать ядро сниффера самостоятельно.

Операционные системы семейства UNIX блокируют прямой доступ к оборудованию с прикладного уровня (так что перепрограммировать сетевую карту просто так не удастся), однако, все же предоставляют специальные рычаги для перевода интерфейса в неразборчивый режим, правда, в различных UNIX'ах эти рычаги сильно неодинаковы, что существенно усложняет нашу задачу.

В состав BSD входит специальный пакетный фильтр (***BPF*** – ***B****SD* ***P****acket* ***F****ilter*), поддерживающий гибкую схему выборочного перехвата чужих пакетов и соответствующий устройству **`/dev/bpf`**. Перевод интерфейса в неразборчивый режим осуществляется посредством IOCTL и выглядит приблизительно так: `ioctl(fd, BIOCPROMISC, 0)`, где fd – дескриптор интерфейса, а BIOCPROMISC – управляющий IOCTL-код. В Solaris'e все осуществляется аналогично, не совпадает только IOCTL-код и устройство называется не bpf, а **`hme`**. Похожим образом ведет себя и SUNOS, предоставляющая потоковый драйвер псевдоустройства **`nit`**, так же называемый *краником в сетевом интерфейсе* (***NIT*** – ***N****etwork* ***I****nterface* ***T****ap*). В отличии от пакетного фильтра BPF, потоковый фильтр NIC перехватывает только входящие пакеты, позволяя исходящим прошмыгнуть мимо него. К тому же он намного медленнее работает. Иную схему грабежа трафика реализует LINUX, поддерживающая специальные IOCTL-коды для взаимодействия с сетью на уровне драйверов. Просто создайте сырой сокет вызовом `socket (PF_PACKET, SOCK_RAW, int protocol)` и переведите связанный

с ним интерфейс в неразборчивый режим –`ifr.ifr_flags |= IFF_PROMISC; ioctl (s, SIOCGIFFLAGS, ifr)`, где s – дескриптор сокета, а `ifr` – интерфейс.

Ниже приводится полностью готовая к употреблению функция подготавливающая сырой сокет к работе с переводом интерфейса в неразборчивый режим и поддерживающая большое количество различных операционных систем, так то: SUN OS, LUNUX, Free BSD, IRIX и Solaris, выдернутая из сниффера, исходный текст которого можно найти по адресу: http://packetstormsecurity.org/sniffers/gdd13.c.

```
/*==========================================================
  Ethernet Packet Sniffer 'GreedyDog' Version 1.30
  The Shadow Penguin Security (http://shadowpenguin.backsection.net)
  Written by UNYUN (unewn4th@usa.net)

#ifdef SUNOS4 /*--------< SUN OS4 >-----------*/
#define NIT_DEV                 "/dev/nit"                                      */
#define DEFAULT_NIC             "le0"                                           */
#define CHUNKSIZE               4096                                            */
#endif

#ifdef LINUX   /*--------< LINUX >-------------*/
#define NIT_DEV                 ""
#define DEFAULT_NIC             "eth0"                                          */
#define CHUNKSIZE               32000                                           */
#endif

#ifdef FREEBSD /*--------< FreeBSD >-----------*/
#define NIT_DEV                 "/dev/bpf"                                      */
#define DEFAULT_NIC             "ed0"                                           */
#define CHUNKSIZE               32000                                           */
#endif

#ifdef IRIX /*-----------< IRIX >--------------*/
#define NIT_DEV                 ""
#define DEFAULT_NIC             ""
#define CHUNKSIZE               60000                                           */
#define ETHERHDRPAD             RAW_HDRPAD(sizeof(struct ether_header))
#endif

#ifdef SOLARIS /*--------< Solaris >-----------*/
#define NIT_DEV                 "/dev/hme"                                      */
#define DEFAULT_NIC             ""
#define CHUNKSIZE               32768                                           */
#endif

#define S_DEBUG                                                                 */
#define SIZE_OF_ETHHDR          14                                              */
#define LOGFILE                 "./snif.log"                                    */
#define TMPLOG_DIR              "/tmp/"                                         */

struct conn_list{
        struct conn_list        *next_p;
        char                    sourceIP[16],destIP[16];
        unsigned long           sourcePort,destPort;
};

struct conn_list *cl; struct conn_list *org_cl;

#ifdef SOLARIS
        int     strgetmsg(fd, ctlp, flagsp, caller)
        int     fd;
        struct  strbuf  *ctlp;
        int     *flagsp;
        char    *caller;
        {
                int     rc;
                static  char errmsg[80];

                *flagsp = 0;
                if ((rc=getmsg(fd,ctlp,NULL,flagsp))<0) return(-2);
                if (alarm(0)<0) return(-3);
                if ((rc&(MORECTL|MOREDATA))==(MORECTL|MOREDATA)) return(-4);
                if (rc&MORECTL) return(-5);
                if (rc&MOREDATA) return(-6);
                if (ctlp->len<sizeof(long)) return(-7);
```

```
                return(0);
        }
#endif

int     setnic_promisc(nit_dev,nic_name)
char    *nit_dev;
char    *nic_name;
{
        int sock; struct ifreq f;

#ifdef SUNOS4
        struct strioctl si; struct timeval timeout;
        u_int chunksize = CHUNKSIZE; u_long if_flags = NI_PROMISC;

        if ((sock = open(nit_dev, O_RDONLY)) < 0)           return(-1);
        if (ioctl(sock, I_SRDOPT, (char *)RMSGD) < 0)       return(-2);
        si.ic_timout = INFTIM;
        if (ioctl(sock, I_PUSH, "nbuf") < 0)                return(-3);

        timeout.tv_sec = 1; timeout.tv_usec = 0; si.ic_cmd = NIOCSTIME;
        si.ic_len = sizeof(timeout); si.ic_dp  = (char *)&timeout;
        if (ioctl(sock, I_STR, (char *)&si) < 0)      return(-4);

        si.ic_cmd = NIOCSCHUNK; si.ic_len = sizeof(chunksize);
        si.ic_dp  = (char *)&chunksize;
        if (ioctl(sock, I_STR, (char *)&si) < 0)            return(-5);

        strncpy(f.ifr_name, nic_name, sizeof(f.ifr_name));
        f.ifr_name[sizeof(f.ifr_name) - 1] = '\0'; si.ic_cmd = NIOCBIND;
        si.ic_len = sizeof(f); si.ic_dp  = (char *)&f;
        if (ioctl(sock, I_STR, (char *)&si) < 0)            return(-6);

        si.ic_cmd = NIOCSFLAGS; si.ic_len = sizeof(if_flags);
        si.ic_dp  = (char *)&if_flags;
        if (ioctl(sock, I_STR, (char *)&si) < 0)      return(-7);
        if (ioctl(sock, I_FLUSH, (char *)FLUSHR) < 0) return(-8);
#endif

#ifdef LINUX
        if ((sock=socket(AF_INET,SOCK_PACKET,768))<0) return(-1);
        strcpy(f.ifr_name, nic_name); if (ioctl(sock,SIOCGIFFLAGS,&f)<0) return(-2);
        f.ifr_flags |= IFF_PROMISC;   if (ioctl(sock,SIOCSIFFLAGS,&f)<0) return(-3);
#endif

#ifdef FREEBSD
        char device[12]; int n=0; struct bpf_version bv; unsigned int size;

        do{
                sprintf(device,"%s%d",nit_dev,n++); sock=open(device,O_RDONLY);
        } while(sock<0 && errno==EBUSY);
        if(ioctl(sock,BIOCVERSION,(char *)&bv)<0) return(-2);
        if((bv.bv_major!=BPF_MAJOR_VERSION)||(bv.bv_minor<BPF_MINOR_VERSION))return -3;
        strncpy(f.ifr_name,nic_name,sizeof(f.ifr_name));
        if(ioctl(sock,BIOCSETIF,(char *)&f)<0) return-4;
        ioctl(sock,BIOCPROMISC,NULL);if(ioctl(sock,BIOCGBLEN,(char *)&size)<0)return-5;
#endif

#ifdef IRIX
        struct sockaddr_raw sr; struct snoopfilter sf;
        int size=CHUNKSIZE,on=1; char *interface;
        if((sock=socket(PF_RAW,SOCK_RAW,RAWPROTO_SNOOP))<0) return -1;
        sr.sr_family = AF_RAW; sr.sr_port = 0;
        if (!(interface=(char *)getenv("interface")))
        memset(sr.sr_ifname,0,sizeof(sr.sr_ifname));
        else strncpy(sr.sr_ifname,interface,sizeof(sr.sr_ifname));
        if(bind(sock,&sr,sizeof(sr))<0) return(-2); memset((char *)&sf,0,sizeof(sf));
        if(ioctl(sock,SIOCADDSNOOP,&sf)<0) return(-3);
        setsockopt(sock,SOL_SOCKET,SO_RCVBUF,(char *)&size,sizeof(size));
        if(ioctl(sock,SIOCSNOOPING,&on)<0) return(-4);
#endif

#ifdef SOLARIS
        long buf[CHUNKSIZE]; dl_attach_req_t ar; dl_promiscon_req_t pr;
        struct strioctl si; union DL_primitives *dp; dl_bind_req_t bind_req;
        struct strbuf c; int flags;

        if ((sock=open(nit_dev,2))<0) return(-1);
```

```
        ar.dl_primitive=DL_ATTACH_REQ; ar.dl_ppa=0; c.maxlen=0;
        c.len=sizeof(dl_attach_req_t); c.buf=(char *)&ar;
        if (putmsg(sock,&c,NULL,0)<0) return(-2);

        c.maxlen=CHUNKSIZE; c.len=0; c.buf=(void *)buf;
        strgetmsg(sock,&c,&flags,"dlokack"); dp=(union DL_primitives *)c.buf;
        if (dp->dl_primitive != DL_OK_ACK) return(-3);

        pr.dl_primitive=DL_PROMISCON_REQ; pr.dl_level=DL_PROMISC_PHYS; c.maxlen = 0;
        c.len=sizeof(dl_promiscon_req_t); c.buf=(char *)&pr;
        if (putmsg(sock,&c,NULL,0)<0) return(-4);

        c.maxlen=CHUNKSIZE; c.len=0; c.buf=(void *)buf;
        strgetmsg(sock,&c,&flags,"dlokack"); dp=(union DL_primitives *)c.buf;
        if (dp->dl_primitive != DL_OK_ACK) return(-5);

        bind_req.dl_primitive=DL_BIND_REQ; bind_req.dl_sap=0x800;
        bind_req.dl_max_conind=0; bind_req.dl_service_mode=DL_CLDLS;
        bind_req.dl_conn_mgmt=0; bind_req.dl_xidtest_flg=0; c.maxlen=0;
        c.len=sizeof(dl_bind_req_t); c.buf=(char *)&bind_req;
        if (putmsg(sock,&c,NULL,0)<0) return(-6);

        c.maxlen=CHUNKSIZE; c.len=0; c.buf=(void *)buf;
        strgetmsg(sock,&c,&flags,"dlbindack"); dp=(union DL_primitives *)c.buf;
        if (dp->dl_primitive != DL_BIND_ACK) return(-7);

        si.ic_cmd=DLIOCRAW; si.ic_timout=-1; si.ic_len=0; si.ic_dp=NULL;
        if (ioctl(sock, I_STR, &si)<0) return(-8);
        if (ioctl(sock,I_FLUSH,FLUSHR)<0) return(-9);
#endif
        return(sock);
}
```

Листинг 1 создание сырого сокета (дескриптора) с переводом интерфейса в неразборчивый режим

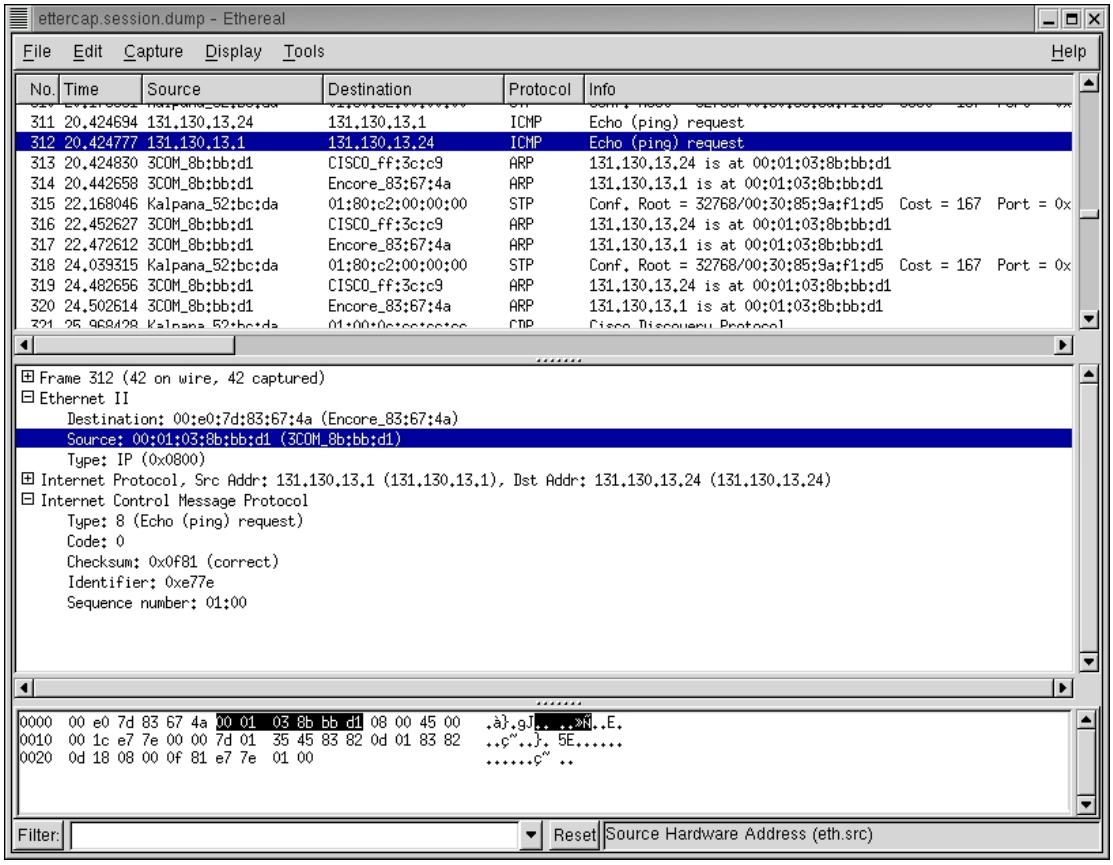

Рисунок 3 сниффер за работой

обнаружение пассивного перехвата

Перевод интерфейса в неразборчивый режим не проходит бесследно и легко обнаруживается утилитой ipconfig, отображающей его статус, правда для этого администратор должен иметь возможность удаленного запуска программ на машине атакующего, чему атакующий может легко воспрепятствовать или модифицировать код ipconfig (и других аналогичных ей утилит) так, чтобы она выдавала подложные данные. Кстати говоря, засылая сниффер на чужой компьютер, помните, что его присутствие в подавляющем большинстве случаев обнаруживается именно по ipconfig!

Многие легальные синифферы автоматически резолвят все полученные IP-адреса, выдавая атакующего с головой. Администратор посылает пакет на несуществующий MAC-адрес от/на несуществующего IP. Узел, поинтересовавшийся доменным именем данного IP, и будет узлом атакующего. Естественно, если атакующий использует собственный сниффер, вырубит DNS в настройках сетевого соединения или оградит себя локальным брандмауэром, администратор останется наедине со своей задницей.

Как вариант, администратор может послать на несуществующий MAC-адрес пакет, предназначенный для атакующего (с действительным IP-адресом и портом отвечающей службы, например, ICMP ECHO более известной как ping). Работая в неразборчивом режиме, сетевая карта передаст такой пакет на IP-уровень, и тот будет благополучно обработан системой, автоматически генерирующий эхо-ответ. Чтобы не угодить в ловушку, атакующий должен отключить ICMP и закрыть все TCP-порты, что можно сделать с помощью того же брандмаузера, конечно, при условии, что тот не открывает никаких дополнительных портов (а большинство брандмауэров их открывают).

Между прочим, грабеж трафика требует ощутимых процессорных ресурсов и машина начинает заметно тормозить. Ну тормозит и хрен с ней – какие проблемы? А вот какие. Администратор делает узлу атакующего ping и засекает среднее время отклика. Затем направляет шторм пакетов на несуществующие (или существующие) MAC-адреса, после чего повторяет ping вновь. Изменение времени отклика полностью демаскирует факт перехвата и, чтобы этому противостоять, атакующий должен либо запретить ICMP ECHO (но это вызовет серьезные подозрения), либо стабилизировать время отклика, вставляя то или иное количество холостых задержек (для этого ему придется модифицировать код эхо-демона).

Разумеется, существуют и другие способы обнаружения пассивного перехвата трафика, однако и перечисленных вполне достаточно, чтобы убедиться в его не безопасности. Например, администратор пускает по сети подложный пароль, якобы принадлежащий root'у, а сам залегает в засаду и ждет, какой зверь на это клюнет, после чего направляет по соответствующему адресу бригаду каратистов быстрого реагирования.

активный перехват или arp-spoofing

Отправляя пакет на заданный IP адрес мы, очевидно, должны доставить его какому-то узлу. Но какому? Ведь сетевая карта оперирует исключительно физическими MAC-адресами, а про IP ничего не знает! Следовательно, нам необходима таблица соответствия MAC и IP адресов. Построением такой таблицы занимается операционная система и делает это она при помощи протокола **ARP** (***A**ddress **R**esolution **P**rotocol* – протокол разрешения адресов). Если физический адрес получателя неизвестен, в сеть отправляется широковещательный запрос типа "обладатель данного IP, сообщите свой MAC". Получив ответ, узел заносит его в локальную ARP-таблицу, для надежности периодически обновляя ее (фактически, ARP-таблица представляет собой обыкновенный кэш). В зависимости от типа операционной системы и ее конфигурации интервал обновления может варьироваться от 30 сек. до 20 мин.

Никакой авторизации для обновления ARP-таблицы не требуется, более того, большинство операционных систем благополучно переваривают ARP-ответы даже если им не предшествовали соответствующие ARP-запросы (SUNOS – одна из немногих, кто не позволяет обмануть себя подобным образом и потому, подложный ARP-пакет должен быть отправлен только после соответствующего ARP-запроса, но до прихода подлинного ответа).

Для захвата чужого IP, атакующему достаточно послать подложный ARP-запрос, который может быть как целенаправленным, так и широковещательным (для отправки/приема ARP-пакетов необходим доступ к сырым сокетам или специальному API операционной системы, подробности можно расковывать в утилите arp). Допустим, атакующий хочет перехватить трафик между узлами "A" и "B". Он посылает узлу "A" подложный ARP-ответ, содержащий IP-адрес узла "B" и свой MAC-адрес, а узлу "B" – ARP-ответ с IP-адресом узла "A" и своим MAC-адресом. Оба узла обновляют свои ARP-таблицы и все отправляемые ими пакеты

попадают на узел злоумышленника, который либо блокирует, либо доставляет их получателю (возможно, в слегка измененном виде, т. е работает как proxy). Если послать подложный ARP-пакет маршрутизатору, атакующий сможет перехватывать и пакеты, приходящие извне данного сегмента сети. Атака такого типа называется **MiM** (сокращение от ***Man In Middle*** – мужик в середине) и схематично изображена на рис. 5.

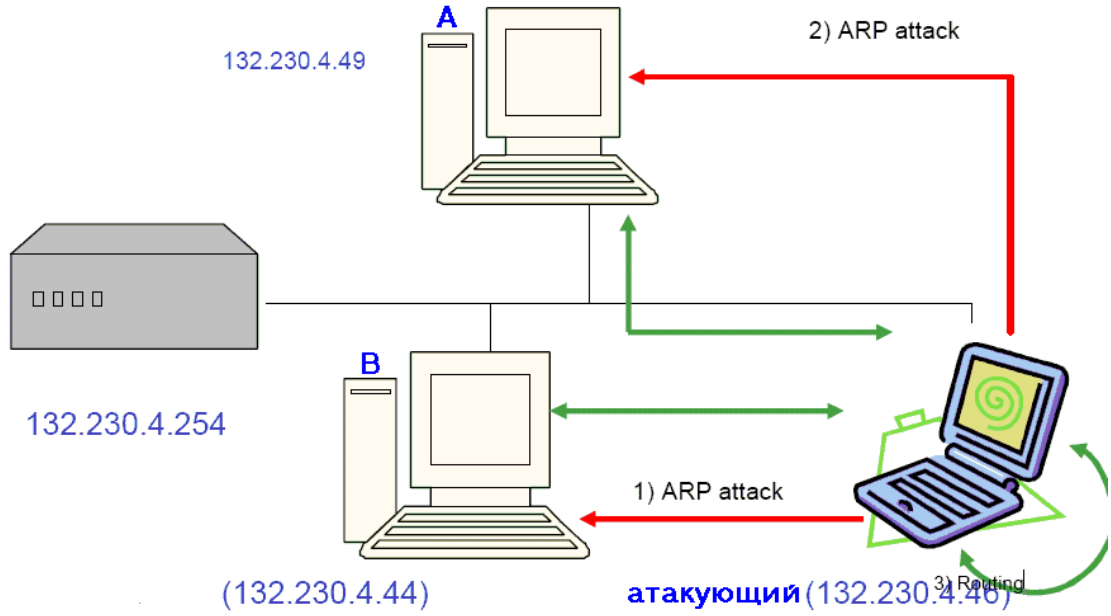

Рисунок 4 атака типа MiM позволяет перехватывать трафик даже в сетях с интеллектуальным хабом

Как вариант, можно послать подложный ARP-ответ с несуществующим MAC-адресом, тогда связь между "A" и "B" будет утеряна, впрочем, через некоторое время она автоматически восстановится вновь (ведь ARP-таблица динамически обновляется!) и, чтобы этого не произошло, атакующий должен направить на жертву мощный шторм подложных пакетов (см. рис. 5).

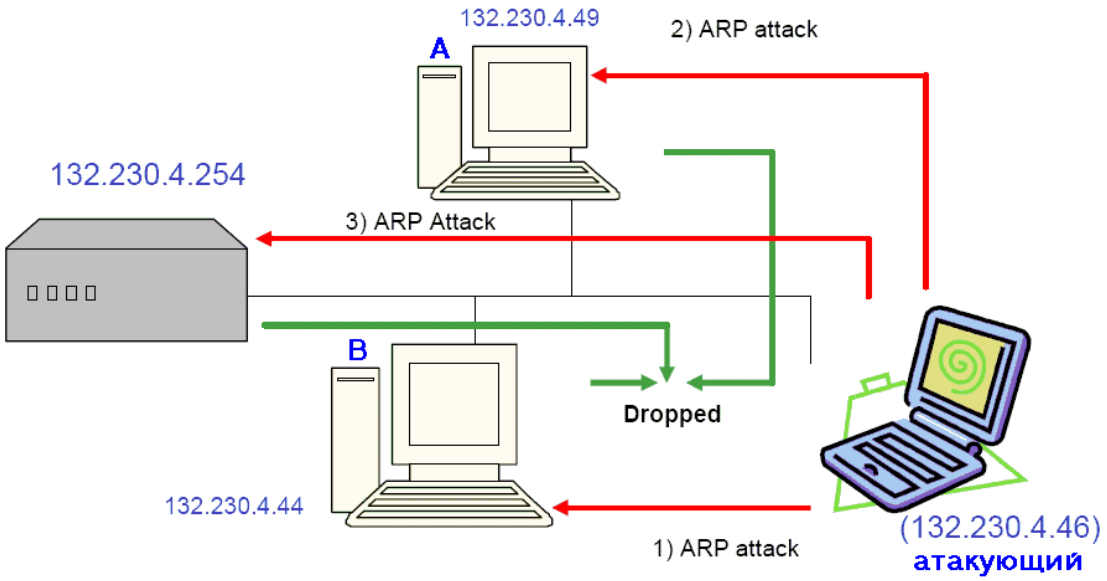

Рисунок 5 разрыв соединения между узлами

Кстати, о штормах. Если маршрутизатор не успевает маршрутизировать поступающие пакеты, он автоматически переключается в широковещательный режим, становясь обычным

хабом. Загрузив маршрутизатор работой по самые помидоры (или дождавшись пиковой загрузки сети), атакующий может преспокойно синффать траффик и в пассивном режиме.

обнаружение активного перехвата

Активная природа ARP-атаки демаскирует злоумышленника и сетевые анализаторы типа **arpwatch** легко обнаруживают перехват. Они грабят все пролетающие по сети пакеты (т. е. работают как сниффер), вытаскивают ARP-ответы и складывают их в базу данных, запоминая какому MAC-адресу какой IP принадлежит. При обнаружении несоответствия администратору отправляется e-mail к моменту получения которого нарушитель обычно успевает скрыться со всем награбленным трафиком. К тому же в сетях с DHCP (сервером динамической раздачи IP-адресов) arpwatch выдает большое количество ложных срабатываний, т. к. одному и тому же MAC-адресу назначаются различные IP-адреса.

Некоторые операционные системы самостоятельно обнаруживают факт захвата своего IP-адреса посторонним узлом, но только в том случае, если злоумышленник использовал широковещательную рассылку (он, что, дурак?!). К тому же, по малопонятным для меня мотивам, ось не отправляет ARP-ответ, отбирая похищенный IP-назад, а просто отделывается многоэтажным предупреждением, смысл которого до рядового пользователя все равно не дойдет (спросите свою секретаршу, что такое айпи и чем он отличается от фаллоса).

Статическая ARP-таблица, формируемая вручную, в этом плане выглядит намного более привлекательной, правда, даже после перехода на нее многие операционные системы продолжают принимать подложные ARP-ответы, безропотно отдавая себя в лапы злоумышленника и убедить их не делать этого, очень трудно, особенно если вы не гуру.

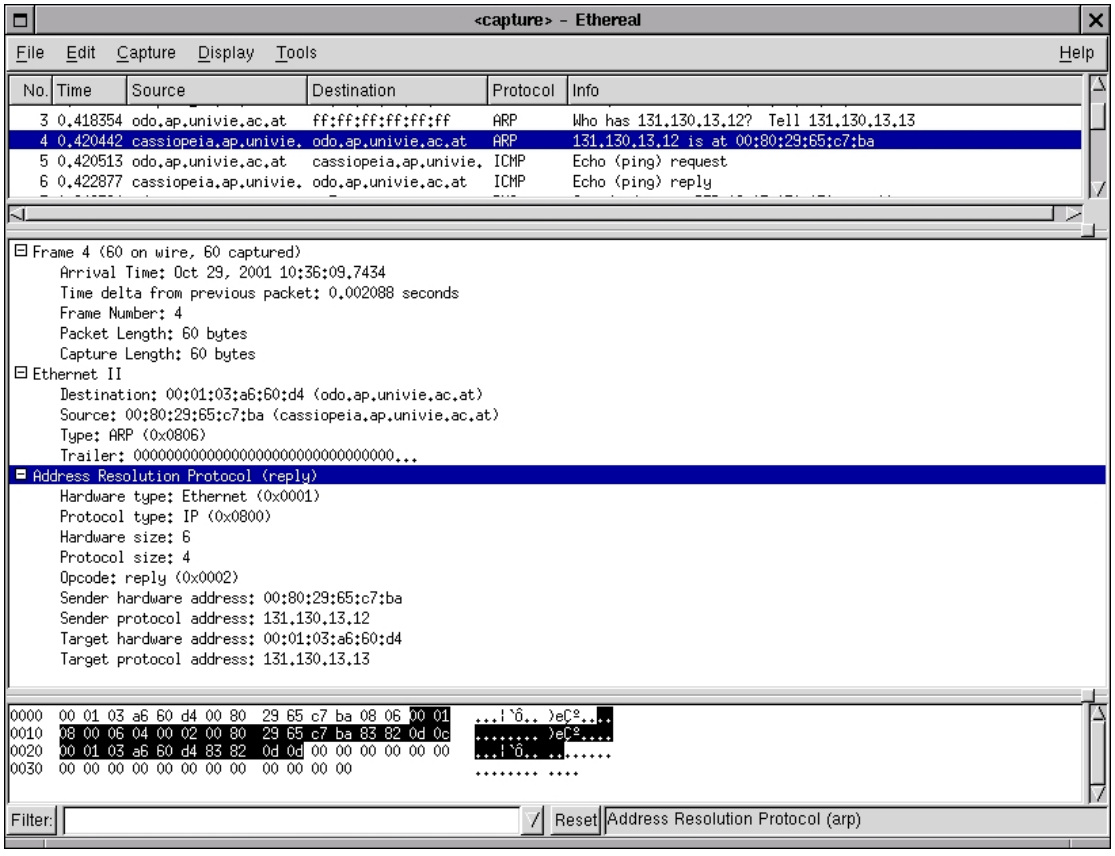

Рисунок 6 перехват подложного ARP-запроса

клонирование карты

Физический адрес сетевой карты обычно жестко прошит в ПЗУ и по Стандарту никакой MAC не может использоваться дважды. Тем не менее, всякое ПЗУ можно перепрограммировать (особенно, если это перепрограммируемое ПЗУ типа EEPROM, каким на новых картах оно обычно и бывает). К тому же, некоторые карты позволяют изменять свой MAC вполне

легальными средствами (например, все той же многострадальной ipconfig). Наконец, заголовок Ethernet-пакета формируется программными, а не аппаратными средствами, поэтому нечестный драйвер может запросто прописать чужой MAC!

Клонирование MAC-адреса позволяет перехватывать трафик даже без присвоения чужого IP и без перевода карты в неразборчивый режим.

обнаружения клонирования и противостояние ему

Факт клонирования легко обнаружить с помощью протокола **RARP** (***R**everse **ARP***), позволяющему определить какой IP-адрес соответствует данному MAC'у. Каждому MAC'у должен соответствовать только один IP-адрес, в противном случае здесь что-то не так. Естественно, если атакующий не только клонирует MAC, но и захватит IP, этот прием не сработает.

Качественные маршрутизаторы позволяют биндить порты (от английского bind – связывание), закрепляя за каждым "поводом" строго определенный MAC, обессмысливая тем самым его клонирование.

>>>> перехват трафика на Dial-Up'e

Для перехвата трафика на модемном подключении через обычную или электронную АТС (т. е. не через кабельный модем) необходимо перепрограммировать маршрутизатор, находящийся у провайдера, что не так-то просто сделать, однако, у большинства провайдеров он так криво настроен, что посторонний трафик сыплется сам – только успевай принимать. В основном он состоит из обрывков бессвязного мусора, но порой в нем встречается кое-что интересное (например, пароли на почтовые ящики).

С другой стороны, перехват Dial-Up трафика позволяет исследовать все пакеты принимаемые/отправляемые вашей машиной. Когда огонек модема возбуждающе мигает, но ни браузер, ни почтовый клиент, ни файлокачалка не активны – разве не интересно узнать какая зараза ломиться в сеть, что и куда она передает? Вот тут-то локальный сниффер и помогает!

Не все снифферы поддерживают соединения типа PPP, хотя с технической точки зрения это даже проще, чем грабить Ethernet. Переводить сетевую карту в неразборчивый режим не нужно, достаточно лишь сформировать сырой IP-сокет. Правда, если операционная система создает для PPP-соединения виртуальный сетевой адаптер, то ситуация становится неоднозначной. Некоторые драйвера требуют перехода в неразборчивый режим, некоторые – нет. За подробностями обращаетесь к документации на свою операционную систему.

>>>> когда сниффер бесполезен

Последнее время наблюдается устойчивая тенденция перехода на протоколы аутентификации, устойчивые к перехвату трафика, которые никогда не передают готовый к употреблению пароль по сети. Вместо этого передается его хеш, причем каждый раз разный (т. е. повторное использование добытого хеша невозможно).

Аутентификация осуществляется приблизительно так. Клиент передает серверу свой логин (обычно открытым текстом), сервер извлекает из своей базы соответствующий хэш, генерирует случайную последовательность байт (challenge) и передает ее клиенту. Клиент вычисляет хэш своего пароля, шифрует его ключом challenge, возвращая результат своей жизнедеятельности серверу. Сервер выполняет аналогичную операцию и сравнивает ее с ответом клиента, после чего либо дает добро, либо отправляет пользователя восвояси.

Операция шифрования хэша необратима, а ее лобовой подбор требует значительного времени, что обессмысливает перехват трафика.

>>>> Stealth-сниффинг

Чтобы сниффать трафик и гарантированно остаться незамеченным, достаточно настроить карту только на прием пакетов, запретив передачу на аппаратном уровне. На картах с витой парой для этого достаточно просто перерезать передающие провода (обычно оранжевого цвета). И хотя существует оборудование, позволяющие засечь левое подключение, подавляющему большинству организаций оно недоступно, поэтому реальная угроза разоблачения хакера исчезающе мала.

Разумеется, stealth-сниффинг поддерживает только пассивный перехват и потому в сетях с интеллектуальным хабом придется дожидаться пиковой загрузки последнего, при которой он дублирует поступающие данные на все порты, как обычный хаб.

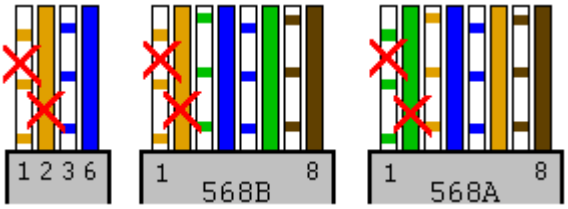

Рисунок 7 легкий взмах ножницами превращает обычную карту в stealth

>>>> *ссылки*

ETTERCAP

Мощный сниффер реализующий атаку Man-In-Middle. Абсолютно бесплатен. Распространяется в исходных текстах. Основное оружие хакера. http://ettercap.sourceforge.net

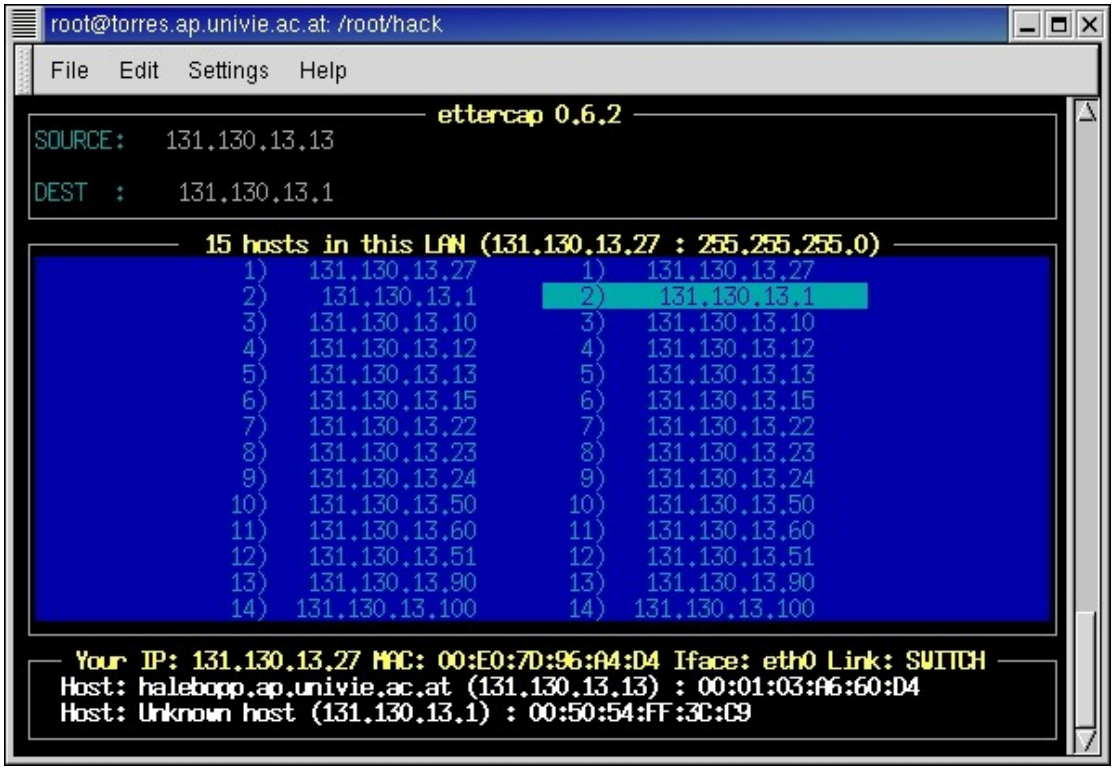

Рисунок 8 главное меню программы ETTERCAP

ARPOISON

Утилита для генерации и отправки подложных ARP-пакетов с заданными MAC и IP адресами. Надежное средство борьбы с интеллектуальными хабами. Бесплатна. Распространяется в исходных текстах http://arpoison.sourceforge.net/

ARPMONIROR

Программа для слежения за ARP-запросами/ответами. В основном используется администраторами для мониторинга сети и выявления людей с лишними яйцами. Бесплатна. http://planeta.terra.com.br/informatica/gleicon/code/index.html

REMOTE ARPWATCH

Автоматизированное средство выявления активного перехвата. Следит за целостностьюARP-таблиц всех членов сети и оперативно уведомляет администратора о подозрительных изменениях. Бесплатна. http://www.raccoon.kiev.ua/projects/remarp/

FAQ

Большой FAQ по снифферам на английском языке. Поимо Ethernet так же затрагивает кабельные модемы и некоторые другие средства связи. www.robertgraham.com/pubs/sniffing-faq.html

заключение

Разработка собственного сниффера – это хороший способ поупражняться в программировании, покопаться в недрах операционной системы и изучить большое количество сетевых протоколов. Короче говоря, совместить приятное с полезным. Можно, конечно, использовать и готовые утилиты, но это все равно, что стрелять в кабана привязанного к дереву – ни азарта, ни удовлетворения.