

Kâr ve eğlence için filtreleri çengellemek: PFIL_HOOKS:

Murat Balaban

< Ülkü SAYILAN tarafından Turkiye'ye çevrilmiştir >

Üretici ve satıcıların ürettikleri ya da sattıkları ürünlerin ticari marka olarak ayırt edilmesini sağlamak için kullandıkları pek çok ad/işaret vardır. Bu belgede bu türden ticari markaların bahsi geçen yerlerde, FreeBSD Projesinin ticari markanın varlığından haberdar olduğu durumlarda söz konusu ad/işaretin ardından ... ya da .@. sembolleri kullanılmıştır. Bu makalede, BSD tabanlı işletim sistemlerinin mevcut ağ altyapısında tek bir satıra dokunmaksızın paket filtreleme modülünü çekirdeğinize (kernel) eklemek için PFIL_HOOKS mekanizmasından ne şekilde yararlanabileceğiniz anlatılmaktadır.

İçindekiler

1. Önsöz.....	1
2. Tarihçe.....	2
3. Dahili Yapı (Internals).....	2
4. Filtreye çengel atmak.....	4
5. Kernel modülüne kendi çengelini atma örneği:	5
6. Referanslar	12

1. Önsöz

Mevcut bir işletim sisteminde ağ I/O (girdi-çıkıtı) altsistemine paket filtre programlarının bağlanması için kullanılan geleneksel yöntem, çekirdek (kernel) kodlarının ilgili kısımlarına müdahale edilerek, belirli bir süre için paket denetimi ve/veya paket dönüşümü için paket filtresinin içine erişim (transfer execution into) sağlayan fonksiyon çağrılarının bu kısımlara yerleştirilmesidir. Bu durumda paket filtreler çekirdeğe (kernele) sabit sabit kablajla (hard-wired) bağlanmış olur ve mevcut protokol kodunda pek çok değişiklik yapılması gerekir. Geçmişte, en ünlü açık kodlu paket filtrelerden biri olan ipfilter'in yazarı Darren Reed, yazdığı programın IP paketi almak için NetBSD,

OpenBSD ve FreeBSD'nin IP kümelerine (stack) yerleşebilmesi için IP protokol kümelerini (özellikle ip_input.c ve ip_output.c) kırmak (hack) zorunda kalmıştır.

2. Tarihçe

Bu yöntemin kullanışlı olmadığı açıkça görülmektedir; böyle bir çengellemenin yapılabilmesi için başka yöntemlerin de olması gerekmektedir. Bu noktadan hareket eden BSD.ciler .paket filtre çengelleri., yani PFIL_HOOKS düşüncesini ortaya atmışlardır. Bu çerçeve ilk olarak NetBSD üzerinde çalışan gruptan Matthew R. Green tarafından tasarlanmış, geliştirilmiş ve 14 Eylül 1996 [1] tarihinde NetBSD ağında duyurulmuştur. Bu girişimdeki temel niyet, IP girdi/çıkışı akışına IPFILTER yerleştirilmesiydi; bununla birlikte, tasarımın amacı, ek paket işleme kodunun gücünün(power)en üst düzeyde taşınabilirlikle (portability) çok hızlı bir biçimde ve çok az bir çabayla çekirdeğe (kernel) eklenebilmesi için genel bir yöntem bulunmasıydı.

PFIL_HOOKS başlangıçta ipfilter için tasarlanmış olduğundan, içerdiği bazı unsurlar .genel. olmasını engelliyordu. Çengeller çağırıldığında, çengel fonksiyonu IP başlıklarının bir kısmını ağ byte sıralamasında ve bir kısmını da host byte sıralamasında bulmaktaydı. Temel amaç çerçeveyi mümkün olduğunca genel kılmaktı ancak uygulamada bu mümkün değildi. Mekanizma IP.den başka hiçbir protokolda çengel noktası sağlayamadı. NetBSD için çalışan gruptan Jason Thorpe bazı değişikliklerin yapılmasını önerdi ve küçük bir tartışma[2] sonrasında bu değişiklikler kabul edildi. Jason 11 Kasım 2000 tarihinde yaptığı değişikliklerle ilgili olarak cvs değişiklik önerisi hakkında aşağıdaki yorumları yaptı [3]

```
PFIL_HOOKS mekanizmasında küçük bir yeniden yapılanma:
```

```
-Tüm paketler kablodan geldikleri gibi PFIL_HOOKS.a geçilir; yani, protokol başlıklarındaki alanlar
```

```
-INET ve INET6 kayıt anahtarı == AF_INET ya da AF_INET6, ve  
dlt == DLT_RAW
```

```
-PFIL_HOOKS böylelikle filtre hook .çengeli-, ve mbuf **, ve ifnet * için işler hale gelir ve
```

```
-IP filtresi için sargı (wrapper function) fonksiyonu işlevi ekleyerek IP Filter ile uyumlulu
```

Bu girişim önemli bir kilometre taşı oldu. Bu güncellemeler PFIL_HOOKS'u daha önce olmadığı kadar genelleştirdi. Özetle, PFIL_HOOKS diğer network protokollerini destekleme yeteneğini kazandı ve API.si daha fazla genellik kazandı. Artık, çoklu paket filtreleyiciler özel protokollere eklenebilir hale gelecekti. Darren Reed ipfilter.ını NetBSD.deki PFIL_HOOKS.u kullanacak şekilde dönüştürerek, 10 Mayıs 2000.de FreeBSD porta eklediğinde[4], FreeBSD de PFIL_HOOKS.u kullanmaya başladı. Daha sonra 22 Eylül 2004 tarihinde PFIL_HOOKS Andre Oppermann tarafından .STABLE.a MFC.ed edildi ve çekirdeğin (kernelin) daimi parçası haline geldi [6]. FreeBSD 5.2 ile birlikte fine-grained locking (koruma mekanizmasında tüm erişim ortamı yerine sadece ihtiyaç duyulan kısmı koruma/ kilitleme) özelliği uygulamaya sokuldu. Darren OpenBSD.nin de bu çalışmayı benimsemesini önerdi; ancak OpenBSD için çalışanların bu çalışmayı kendi CVS repolarına dahil edecek denli ilgi çekici bulmadıkları anlaşılmaktadır. Bu makalenin hazırlandığı tarih itibarıyla, OpenBSD çekirdeğinde (kernelinde) pfil.c pfil.c dosyasına rastlamadım.

3. Dahili Yapı (Internals)

Paket filtre noktası `pfil_head_register` (9) fonsiyonu ile kaydedildir. Bu, genellikle, belirli bir protokolün (mesela, IP) giriş ve çıkış noktalarında yapılır. Örneğin, IP protokolü bir paket filtre başlığını (paket filtre noktası) `PFIL_TYPE_AF` tipinde ve `AF_INET` (IPv4) adres kümesinde kaydeder:

```
/* Initialize packet filter hooks. */
inet_pfil_hook.ph_type = PFIL_TYPE_AF;
inet_pfil_hook.ph_af = AF_INET;
if ((i = pfil_head_register(inet_pfil_hook)) != 0)
```

FreeBSD 7.0-CURRENT `/usr/src/sys/netinet/ip_input.c:ip_init` [7]

Aynı kod IPv6 kodunda da görülebilir: `/usr/src/sys/netinet6/ip6_input.c:ip6_init` [8]

Protokol kümeleriyle kaydedilen paket filtre noktaları salt bağlantılı listede (singly linked list) saklanır. Listenin her bir elemanı `pfil` tipi ve adres kümesinin yanı sıra IN(Girdi) ve OUT(Çıktı) yönü boyunca paket filtre fonsiyonu kuyruk sırası (tail queue) içerir. Kod, filtre noktaları bağlantılı listesine `pfil_head` yapısını ekler.

"`pfil` giriş ve çıkış listeleri başlangıçta `sys/queue.h` LIST yapısı olarak belirlenmişti; ancak, NetBSD 1.4.de bu yapı TAILQ yapısı olarak değiştirildi. Bu değişiklik, giriş ve çıkış filtrelerini ters yönde çalışabilmesine olanak sağlamak, çekirdek (kernel) içine veya çekirdek (kernel) dışına aynı veri yolunun kullanılmasına olanak sağlamak amacıyla yapıldı.. [9] [10]

```
% grep pfil_head_register */*.c
netinet/ip_input.c:      if ((i = pfil_head_register( inet_pfil_hook)) != 0)
netinet6/ip6_input.c:   if ((i = pfil_head_register( inet6_pfil_hook)) != 0)
%
```

Bu bize FreeBSD çekirdeğinin (kernelinin) IPv4 ve IPv6 protokollerine kayıtlı filtreleme noktası olduğunu gösterir. Bununla birlikte, diğer protokoller, yani TCP ve UDP için filtre noktası eklenmesi abes olacaktır/gereksizdir.

`pfil_head` yapısının `busy_count` (meşgul_sayım) üyesi listenin uyarlandığını ya da değer halen -1 olduğunu gösterir; -1 değeri, `pfil_head` yapısına herhangi bir filtre eklenmediğini gösterir, protokolün filtreleri çalıştırmaya çalışmamasını sağlar.

`pfil_run_hooks` [9] (`pfil_head` yapı tipinin `type struct`) `inet_pfil_hook` kuyruk sırasında girişi çapraz geçer (traverses) (`PFIL_IN` hakkındaki tartışmaya bakınız) ve içerisindeki filtreleri çalıştırır. FreeBSD çekirdeği (kerneli) içindeki IPv4 protokol unsurunda `AF_INET` `pfil` çengellerini çalıştıran kod şudur:

```
if (pfil_run_hooks( inet_pfil_hook, m, m->m_pkthdr.rcvif,
    PFIL_IN, NULL) != 0)
    return;
```

FreeBSD 7.0-CURRENT `/usr/src/sys/netinet/ip_input.c:ip_input` [7]

`pfil_head_unregister` [9] fonsiyonu giriş ve çıkış listelerine eklenen herhangi bir filtreyi siler ve ilgili paket filtre başlığını `pfil_heads` bağlantılı listesinden çıkarır.

`pfil_add_hook` [9], belirli bir paket filtre noktasına (`pfil_head`) filtre eklemek için kullanılır. Bu fonsiyon çağrılacak bir fonsiyonu, filtre fonsiyonuna tahsis edilecek void * arg parametresini, flag.leri (yani, `PFIL_IN` / `PFIL_OUT`) ve çengelin ekleneceği `pfil_head` filtre noktası için bir işaretçiyi (pointer) alır.

IPv4 giriş akışı, gelen bir paketin protokolünün alınması için giriş rutinini/akışını belirlemeye yönelik basit ve etkili bir şema kullanır. `ip_input`, IP paketinin `ip_proto` alanındaki protokol numarasının alınan protokolün protokol-anahtar girişi göre adreslenmesi için 256 parçalı adresleme dizilişi (256-element mapping array) kullanır. Daha sonra, sistem içerisinde ayrı bir uygulaması olan her bir protokol için ilgili adresleme girişi IP protokol anahtarındaki protokolün endeksine göre ayarlanır. Bir paket alındığında, adresleme dizilişinde endeksleme yapmak için IP sadece protokol alanını kullanır ve uygun protokolün giriş akışını çağırır. [11]

Şimdi, protokol paketi akışındaki kodların tek bir satırına dokunmaksızın protokol filtrelerinin dinamik bir biçimde çalıştırılabilmesini sağlamak için, bu yapıya `pfil_head` yapı tipinin (type struct) yeni bir `pr_pfh` değişkeni eklenmiş bulunuyor.

```
struct ipprotosw {
    ...
    struct pfil_head      pr_pfh;
};
```

FreeBSD 7.0-CURRENT /usr/src/sys/netinet/ipprotosw.h [12]

`pfil_get_head` uygulandığında, protokole ilgili tüm filtreler geri döndürülür, bu da her bir protokole bir filtre demektir.

4. Filtreye çengel atmak

Şimdi, PFIL_HOOKS. un ne olduğu hakkında yeterince bilgiye sahibiz, tasarımının ve içeriğinin altında yatan dinamikleri ise biraz biliyoruz, canlı bir örnek ile IP Filtre veya IPFW programlarının kayıtlı IPv4 ve IPv6 pfil başlıklarına kendilerini nasıl çengellediklerini gösterebiliriz. IPFW den başlayalım:

```
static int
ipfw_hook(void)
{
    struct pfil_head *pfh_inet;

    if (ipfw_pfil_hooked)
        return EEXIST;

    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return ENOENT;

    pfil_add_hook(ipfw_check_in, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_add_hook(ipfw_check_out, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);

    return 0;
}
```

FreeBSD 7.0-CURRENT /usr/src/sys/netinet/ip_fw_pfil.c [13]

Bu kodun yaptığı şey, ilkönce ipfw IPv4 protokol (PFIL_TYPE_AF, AF_INET), için `pfil_head_structure` isteğinde bulunur, gelen trafik için `ipfw_check_in` ve giden IP trafik için de `ipfw_check_out` araya eklenir.

IPF:

```
# if (__NetBSD_Version__ >= 105110000) || (__FreeBSD_version >= 501108)
```

```
    ph_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
#   ifdef USE_INET6
    ph_inet6 = pfil_head_get(PFIL_TYPE_AF, AF_INET6);

    ..
    ..

    if (ph_inet != NULL)
        error = pfil_add_hook((void *)fr_check_wrapper, NULL,
                               PFIL_IN|PFIL_OUT, ph_inet);
```

FreeBSD 7.0-CURRENT /usr/src/sys/contrib/ipfilter/netinet/ip_fil.c:ipl_enable [17]

PF:

```
    if (pf_pfil_hooked)
        return (0);

    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return (ESRCH); /* XXX */
    pfil_add_hook(pf_check_in, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_add_hook(pf_check_out, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);
```

FreeBSD 7.0-CURRENT /usr/src/sys/contrib/pf/net/pf_ioctl.c [15]

5. Kernel modülüne kendi çengelinizi atma örneği:

PFIL_HOOKS ünlü açık kaynak kodlu firewal tarafından nasıl kullanıldığını artık biliyoruz. Basit bir kernel modülü kodlayabilir ve BSD IPv4 kümelerine çengelleyebiliriz. Modülümüz host üzerinden gelen giden byte sayısını hesaplayacaktır. Eğer yüklenebilir kernel modülü nasıl çalışır ve nasıl kodlanır bir fikriniz yok ise tavsiyemiz FreeBSD Mimasiri El kitabı [16]- FreeBSD Architecture Handbook. Bölüm 9 (FreeBSD Araç Sürücüsü/FreeBSD Device Driver) Kısım II de (Değişken Çekirdek Bağlayıcı Kolaylıkları/ Dynamic Kernel Linker Facility) .Hello world/merhaba dünya. LKM kodu nasıl yazıldığı örneklenmektedir.

Bizim hisar modülü derlemek için Makefile. i kulanacağız:

```
SRCS=hisar.c
KMOD=hisar

rclean:
@make clean
rm -f *~

.include <bsd.kmod.mk>
```

Başka bir makalenin başlığı olduğu için bütün kaynak kodunu ele almayacağım. Önemli kısımları ele alacağım. Aşağıdakiler init_module ve deinit_modülerdir. Modül yüklendiğinde init_module, boşaltıldığında/ ise deinit_modüle olarak adlandırılacaklar. IP paketi alındığında init_module fonksiyon listesinden hisar_chkinput olarak adlandırılan fonksiyonunu ve IP çıktı filtre listesinden hisar_chekoutput fonksiyonu ekler. Sonra da bir

karakter sürücüsü/aracı (device) olan /dev/flwacct.ı yaratır ve kaydeder. deinit modülü daha önce eklenen filtre fonksiyonlarını IPv4 filtrelerinden kaldırır ve karakter sürücüsünü/aracını yokeder.

init_module adds hisar_chkinput function to the list of functions to be called when an IP packet is received; and hisar_chkoutput function to the IP output filters list. Then it creates and registers a character device /dev/flwacct. deinit_module removes the previously added filter functions from IPv4 filters, and destroys the character device.

```
static int
init_module(void)
{
    struct pfil_head *pfh_inet;

    if (hisar_hooked)
        return (0);
    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return ESRCH;
    pfil_add_hook(hisar_chkinput, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_add_hook(hisar_chkoutput, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);
    hisar_hooked = 1;
    flwbuf = (void *)malloc(PAGE_SIZE, M_TEMP, M_WAITOK | M_ZERO);
    sdev = make_dev(&flwacct_cdevsw,
                   FLWACCT_MINOR,
                   UID_ROOT,
                   GID_WHEEL,
                   0600,
                   "flwacct");
    uprintf("Loaded %s %s\n", MODNAME, MODVERSION);
    return 0;
}

static int
deinit_module(void)
{
    struct pfil_head *pfh_inet;

    if (!hisar_hooked)
        return (0);
    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return ESRCH;
    pfil_remove_hook(hisar_chkinput, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_remove_hook(hisar_chkoutput, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);
    hisar_hooked = 0;
    free(flwbuf, M_TEMP);
    destroy_dev(sdev);
    uprintf("Unloaded %s %s\n", MODNAME, MODVERSION);
    return 0;
}
```

Aşağıda tanımlanan filtre fonksiyonlarından `pfil_ekleme_çengeli(pfil_add_hook).a` (9) geçiyoruz:

```
filter_func(void *arg, struct mbuf **m, struct ifnet *ifp, int dir, struct inpcb *inp)
```

Function takes a `void *` argument, which you can pass for every input packet, a pointer to pointer to the `mbuf` structure carrying the entire packet, a pointer to the `ifnet` interface structure, from which the packet is originating, packet direction `PFIL_IN` or `PFIL_OUT` and a pointer to the `inpcb` protocol control block structure.

Our `hisar_chkinput` has been added to `PFIL_IN` filters. So whenever an IP packet arrives, our code will be called from `pfil_run_hooks` in `ip_input`. `hisar_chkinput` only extracts the received packet length from the `mbuf`, and adds to the `in_bytes` global variable.

```
static int
hisar_chkinput(void *arg, struct mbuf **m, struct ifnet *ifp, int dir, struct inpcb *inp)
{
    in_bytes += (*m)->m_len;
    return 0;
}
```

Örneğin, eşik değerinden sonra gelen paket sayılarını sınırlar ve aşağıdaki gibi paketleri bloklatabilirsiniz:

```
static int
hisar_chkinput(void *arg, struct mbuf **m, struct ifnet *ifp, int dir, struct inpcb *inp)
{
    in_bytes += (*m)->m_len;
    if (in_bytes > in_maxbytes)
        return FLW_DROP; /* return (1) */
    return 0;
}
```

Eğer herhangi bir filtre fonksiyonun değeri 0 dan farklı olursa, kalan paket işleme durdurulur, öteki paket işlenmez bunun anlamı paket engellenir. `hisar_chkoutput`, `pfil_head`.

Benzer şekilde, `hisar_chkoutput` IPv4 `pfil_head` yapısının `PFIL_OUT` filtresinden yardım alır. Paket boyunu `out_bytes` küresel değişkenine ekler

```
static int
hisar_chkoutput(void *arg, struct mbuf **m, struct ifnet *ifp, int dir, struct inpcb *inp)
{
    out_bytes += (*m)->m_len;
    return 0;
}
```

`/dev/flwacct` sürücüsü açıldığında ve okuma(`read`)(2) çağrıldığında, modülümüzdeki `dev_read` çağrılacaktır. Kullanıcı alanlarındaki `in_bytes` ve `out_bytes` değerlerini kapsayan diziyi(string) kopyalıyoruz:

```
int
dev_read(struct cdev *dev, struct uio *uio, int ioflag)
{
    int rv = 0;

    sprintf(flwbuf, "%016d,%016d\n", in_bytes, out_bytes);
    rv = uiomove(flwbuf, MIN(uio->uio_resid, strlen(flwbuf)), uio);
}
```

```
return rv;
}
```

Hisar kernel modülümüz için bütün bir kaynak kodu aşağıda:

```
/*
 * A simple network flow accountant
 * exemplifying PFIL_HOOKS
 *
 * Murat Balaban
 * $Id: article.sgml,v 1.15 2005/11/25 20:07:04 murat Exp $
 *
 */

#include <sys/types.h>
#include <sys/module.h>
#include <sys/system.h>
#include <sys/errno.h>
#include <sys/param.h>
#include <sys/kernel.h>
#include <sys/conf.h>
#include <sys/uio.h>
#include <sys/malloc.h>
#include <sys/ioccom.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/sysent.h>
#include <sys/sysproto.h>
#include <sys/proc.h>
#include <sys/syscall.h>

#include <machine/iodev.h>

#include <netinet/in_system.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <net/if.h>
#include <net/pfil.h>

#define MODNAME "EnderUNIX HISAR - A simple network flow accountant"
#define MODVERSION "1.0"
#define FLWACCT_MINOR 11

static volatile int hisar_hooked = 0;

d_open_t dev_open;
d_close_t dev_close;
d_read_t dev_read;
d_write_t dev_write;
static struct cdev *sdev;
static int count = 0; /* Device Busy flag */
static int in_bytes = 0; /* Bytes IN */
```

```
static int out_bytes = 0; /* Bytes OUT */
static char *flwbuf = NULL; /* Priv. Buffer */

static struct cdevsw flwacct_cdevsw = {
    .d_version = D_VERSION,
    .d_open = dev_open,
    .d_close = dev_close,
    .d_read = dev_read,
    .d_name = "flwacct",
    .d_maj = CDEV_MAJOR,
    .d_flags = D_TTY,
};

static int
hisar_chkinput(void *arg, struct mbuf **m, struct ifnet *ifp, int dir,
struct inpcb *inp)
{
    in_bytes += (*m)->m_len;
    return 0;
}

static int
hisar_chkoutput(void *arg, struct mbuf **m, struct ifnet *ifp, int dir,
struct inpcb *inp)
{
    out_bytes += (*m)->m_len;
    return 0;
}

static int
init_module(void)
{
    struct pfil_head *pfh_inet;

    if (hisar_hooked)
        return (0);
    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return ESRCH;
    pfil_add_hook(hisar_chkinput, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_add_hook(hisar_chkoutput, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);
    hisar_hooked = 1;
    flwbuf = (void *)malloc(PAGE_SIZE, M_TEMP, M_WAITOK | M_ZERO);
    sdev = make_dev(&flwacct_cdevsw,
                    FLWACCT_MINOR,
                    UID_ROOT,
                    GID_WHEEL,
                    0600,
                    "flwacct");
    printf("Loaded %s %s\n", MODNAME, MODVERSION);
    return 0;
}
```

```
static int
deinit_module(void)
{
    struct pfil_head *pfh_inet;

    if (!hisar_hooked)
        return (0);
    pfh_inet = pfil_head_get(PFIL_TYPE_AF, AF_INET);
    if (pfh_inet == NULL)
        return ESRCH;
    pfil_remove_hook(hisar_chkinput, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet);
    pfil_remove_hook(hisar_chkoutput, NULL, PFIL_OUT | PFIL_WAITOK, pfh_inet);
    hisar_hooked = 0;
    free(flwbuf, M_TEMP);
    destroy_dev(sdev);
    uprintf("Unloaded %s %s\n", MODNAME, MODVERSION);
    return 0;
}

/* Module event handler */
static int
mod_evhandler(struct module *m, int what, void *arg)
{
    int err = 0;

    switch(what) {
    case MOD_LOAD:
        err = init_module();
        break;
    case MOD_UNLOAD:
        err = deinit_module();
        break;
    default:
        err = EINVAL;
        break;
    }
    return err;
}

int
dev_open(struct cdev *dev, int oflags, int devtype, struct thread *td)
{
    int err = 0;

    if (count > 0)
        return EBUSY;
    count = 1;
    return (err);
}

int
dev_close(struct cdev *dev, int fflag, int devtype, struct thread *td)
{

```

```
    int err = 0;
    count = 0;
    return (err);
}

int
dev_read(struct cdev *dev, struct uio *uio, int ioflag)
{
    int rv = 0;

    sprintf(flwbuf, "%016d,%016d\n", in_bytes, out_bytes);
    rv = uiomove(flwbuf, MIN(uio->uio_resid, strlen(flwbuf)), uio);
    return rv;
}

DEV_MODULE(hisarmodule, mod_evhandler, NULL);
MODULE_VERSION(hisarmodule, 1);
```

Kaynak kodu derlemek için şöyle yazın:

```
efe@~/freebsd-kernel/pfil/lkm# make
Warning: Object directory not changed from original /root/freebsd-kernel/pfil/lkm
@ -> /usr/src/sys
machine -> /usr/src/sys/i386/include
cc -O -pipe -D_KERNEL -DKLD_MODULE -nostdinc -I- \
-I. -I@ -I@/contrib/altq -I@/../include -I/usr/include -finline-limit=8000 \
-fno-common -mno-align-long-strings -mpreferred-stack-boundary=2 \
-mno-mmx -mno-3dnow -mno-sse -mno-sse2 -ffreestanding -Wall -Wredundant-decls \
-Wnested-externs -Wstrict-prototypes -Wmissing-prototypes -Wpointer-arith \
-Winline -Wcast-qual -fformat-extensions -std=c99 -c hisar.c
ld -d -warn-common -r -d -o hisar.kld hisar.o
touch /root/freebsd-kernel/pfil/lkm/export_syms
awk -f /sys/conf/kmod_syms.awk hisar.kld /root/freebsd-kernel/pfil/lkm/export_syms
| xargs -J% objcopy % hisar.kld
ld -Bshareable -d -warn-common -o hisar.ko hisar.kld
objcopy --strip-debug hisar.ko
efe@~/freebsd-kernel/pfil/lkm
```

Derlenen kernel modülünü yüklemek için `kldload` komutunu kullanın `kldstat` yüklenen güncel kernel modülünü listeleyecektir:

```
efe@~/freebsd-kernel/pfil/lkm# kldload ./hisar.ko
Loaded EnderUNIX HISAR - A simple network flow accountant 1.0
efe@~/freebsd-kernel/pfil/lkm# kldstat
Id Refs Address      Size      Name
 1     8 0xc0400000 5e5c34   kernel
 2     1 0xc09e6000 59c4     snd_ich.ko
 3     2 0xc09ec000 1d4fc    sound.ko
 4    15 0xc0a0a000 58034    acpi.ko
 5     1 0xc227e000 2000     fire_saver.ko
```

```
6      1 0xc25d7000 2000      hisar.ko
efe@~/freebsd-kernel/pfil/lkm#
```

Hisar şimdi kernel adres alanına yüklenmiştir, ve gelen giden byteleri saymaktadır. Modulümüzün karakter araç kaydı olduğunu unutmayalım, read(2) üzerine çağrıldığında IP kümeleri üzerinden gönderilen ve alınan güncel bytes sayılarını bize verir. Aşağıdaki kod /dev/flwacct yi açar ve oradan okunur.

```
/* flwcnt.c read from /dev/flwacct */

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

int
main(void)
{
    char buf[1024];
    int fd;

    if ((fd = open("/dev/flwacct", O_RDONLY)) == -1) {
        perror("open");
        exit(1);
    }
    while(read(fd, buf, sizeof(buf) - 1) > 0) {
        printf("IN: %.16s bytes, OUT: %.16s bytes\n", buf, buf + 17);
        sleep(1);
    }
    close(fd);
    return 0;
}
```

Derleyin ve çalıştırın:

```
efe@~# make flwcnt
cc -O -pipe flwcnt.c -o flwcnt
efe@~# ./flwcnt
IN: 0000000004621389 bytes, OUT: 000000000900378 bytes
IN: 0000000004621429 bytes, OUT: 000000000900514 bytes
IN: 0000000004621469 bytes, OUT: 000000000900650 bytes
IN: 0000000004621509 bytes, OUT: 000000000900786 bytes
^C
efe@~#
```

6. Referanslar

1. <http://cvsweb.netbsd.org/bsdweb.cgi/~checkout~/src/sys/net/pfil.c?rev=1.1&content-type=text/plain>

2. <http://marc.theaimsgroup.com/?l=netbsd-tech-net&m=97356923225193&w=2>
3. <http://cvswb.netbsd.org/bsdweb.cgi/src/sys/net/pfil.c?rev=1.16&content-type=text/x-cvswb-markup>
4. <http://cvswb.freebsd.org/cgi/cvswb.cgi/src/sys/net/pfil.c?rev=1.1&content-type=text/x-cvswb-markup>
5. <http://marc.theaimsgroup.com/?l=openbsd-tech&m=95465467513182&w=2>
6. http://cvswb.freebsd.org/cgi/cvswb.cgi/src/sys/net/ip_input.c?rev=1.283.2.3&content-type=text/x-cvswb-markup
7. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/netinet/ip_input.c?rev=1.305&content-type=text/x-cvswb-markup
8. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/netinet6/ip6_input.c?rev=1.81.2.3&content-type=text/x-cvswb-markup
9. <http://www.freebsd.org/cgi/man.cgi?query=pfil&apropos=0&sektion=0&manpath=FreeBSD+7.0-current&format=html>
10. <http://cvswb.netbsd.org/bsdweb.cgi/src/sys/net/pfil.c?rev=1.7&content-type=text/x-cvswb-markup>
11. Marshall Kirk McKusick, George V. Neville-Neil *The Design and Implementation of the FreeBSD Operating System*. Boston, Mass. : Addison-Wesley, 2004. ISBN 0-201-70245-2
12. <http://www.freebsd.org/cgi/cvswb.cgi/src/sys/netinet/ipprotosw.h?rev=1.5.2.1&content-type=text/x-cvswb-markup>
()
13. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/netinet/ip_fw_pfil.c?rev=1.19&content-type=text/x-cvswb-markup
14. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/netinet/ip_output.c?rev=1.248&content-type=text/x-cvswb-markup
15. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/contrib/pf/net/pf_ioctl.c?rev=1.12.2.8&content-type=text/x-cvswb-markup
16. <ftp://ftp.tr.freebsd.org/pub/FreeBSD/doc/en/books/arch-handbook/>
17. http://www.freebsd.org/cgi/cvswb.cgi/src/sys/contrib/ipfilter/netinet/ip_fil.c