von Nicolas Bouliane
<nib(at)cookinglinux!org>

Über den Autor:
Nicolas ist ein junger Kämpfer in der Gemeinschaft freier Software. Er ist ein GNU/Linux−Abhängiger, seit dem er es im Jahre 1998 auf seinem Rechner installierte. Er verbringt seine Zeit damit, den Linux Netzwerkstack zu untersuchen, freie Software zu schreiben und Linux−bezogene Konferenzen wie OLS zu besuchen. Wenn er nicht vor seinem Computer sitzt, schaut er SciFi−Filme an, spielt Schach und hört Richard Stallmans Reden.

Übersetzt ins Deutsche von:
Hermann J. Beckers
<hj.beckers/at/onlinehome.de>

Schreiben Ihres eigenen netfilter−Tests

Zusammenfassung:

Das iptables/netfilter−Rahmenwerk gibt uns die Möglichkeit, Eigenschaften hinzuzufügen. Dazu schreibt man Kernel−Module, die sich bei diesem Rahmenwerk registrieren. Abhängig von der Kategorie dieser neuen Eigenschaft schreiben wir auch ein iptables−Modul. Durch das Schreiben Ihrer neuen Erweiterung können Sie ein bestimmtes Paket testen, verändern, akzeptieren und verfolgen. Tatsächlich können Sie im Bereich des Filterns fast alles tun, was Sie möchten. Beachten Sie, dass ein kleiner Fehler in einem Kernel−Modul Ihren Computer abstürzen lassen kann.

Aus Gründen der Einfachheit werde ich einen Mustertest erläutern, den ich geschrieben habe. Auf diese Weise hoffe ich, dass die Interaktion mit dem Rahmenwerk leichter zu verstehen ist. Ich setze hier voraus, dass Sie bereits etwas über iptables wissen und auch die C−Programmierung kennen.

Dieses Beispiel wird Ihnen zeigen, wie man ein Paket entsprechend der Quell− und/oder Zieladresse verfolgt.

Beschreibung

Die allgemeinen Schritte zum Erstellen eines iptables/netfilter−Moduls sind:

- Sie möchten eine spezielle Situation abfangen.
- Schreiben des Teils für den Benutzerbereich, der die Argumente behandelt.
- Schreiben des Teils für den Kernel−Bereich, der die Pakete analysiert und erklärt, ob eine Übereinstimmung vorliegt oder nicht.

1.0 Das iptables−Modul

Zweck einer iptables−Bibliothek ist einfach die Interaktion mit dem Anwender. Sie behandelt die Argumente, die der Anwender an den Kernel−Teil weiterleiten will.

1.1 verfügbare Strukturen und Funktionen

Zunächst einige grundlegende Strukturen <*iptables/include/iptables.h*>
Weiter im Text werden wir sehen, was der Zweck eines jeden Feldes ist.

```
/* Include file for additions: new matches and targets. */
struct iptables_match
{
   struct iptables_match *next;

   ipt_chainlabel name;

   const char *version;

   /* Size of match data. */
   size_t size;

   /* Size of match data relevent for userspace comparison purposes */
   size_t userspacesize;

   /* Function which prints out usage message. */
   void (*help)(void);

   /* Initialize the match. */
   void (*init)(struct ipt_entry_match *m, unsigned int *nfcache);

   /* Function which parses command options; returns true if it
           ate an option */
   int (*parse)(int c, char **argv, int invert, unsigned int *flags,
           const struct ipt_entry *entry,
           unsigned int *nfcache,
           struct ipt_entry_match **match);

   /* Final check; exit if not ok. */
   void (*final_check)(unsigned int flags);

   /* Prints out the match iff non-NULL: put space at end */
   void (*print)(const struct ipt_ip *ip,
            const struct ipt_entry_match *match, int numeric);

   /* Saves the match info in parsable form to stdout. */
   void (*save)(const struct ipt_ip *ip,
           const struct ipt_entry_match *match);

   /* Pointer to list of extra command-line options */
   const struct option *extra_opts;

   /* Ignore these men behind the curtain: */
   unsigned int option_offset;
   struct ipt_entry_match *m;
   unsigned int mflags;
#ifdef NO_SHARED_LIBS
   unsigned int loaded; /* simulate loading so options are merged properly */
#endif
};
```

1.2 Innerhalb des Programm−Skeletts

1.2.1 Initialisierung

Wir initialisieren die allgemeinen Felder in der Struktur 'iptables_match'.

```
static struct iptables_match ipaddr
= {
```

'Name' ist die Zeichenkette mit dem Namen Ihrer Bibliothek (z. B. libipt_ipaddr).
Sie können keinen anderen Namen angeben, er wird für das automatische Laden Ihrer Bibliothek benutzt.

```
    .name           = "ipaddr",
```

Das nächste Feld 'version' ist die Version von iptables. Die beiden nächsten Felder werden benutzt, um eine Korelation zwischen der Größe der zwischen dem Benutzerbereich und dem Kernelbereich gemeinsam genutzten Struktur zu erhalten.

```
    .version        = IPTABLES_VERSION,
    .size           = IPT_ALIGN(sizeof(struct ipt_ipaddr_info)),
    .userspacesize  = IPT_ALIGN(sizeof(struct ipt_ipaddr_info)),
```

'Help' wird aufgerufen, wenn der Benutzer 'iptables −m module −h' eingibt. 'Parse' wird aufgerufen, wenn Sie eine neue Regel eingeben, es dient zur Überprüfung der Argumente. 'print' wird von 'iptables −L' aufgerufen, um die vorher eingegebenen Regeln anzuzeigen.

```
    .help           = &help,
    .init           = &init,
    .parse          = &parse,
    .final_check    = &final_check,
    .print          = &print,
    .save           = &save,
    .extra_opts     = opts
};
```

Die iptables−Infrastruktur kann mehrere gemeinsam genutzte Bibliotheken unterstützen. Jede Bibliothek muss sich bei iptables durch Aufruf von 'register_match()' registrieren, welche in <*iptables/iptables.c*> definiert ist. Diese Funktion wird aufgerufen, wenn das Modul von iptables geladen wird. Zu weiteren Informationen hierzu siehe 'man dlopen'.

```
void _init(void)
{
   register_match(&ipaddr);
}
```

1.2.2 Speicher−Funktion

Wenn wir eine Regelmenge sichern wollen, bietet iptables das Werkzeug 'iptables−save', das alle Ihre Regeln ausgibt. Offensichtlich benötigt es die Hilfe Ihrer Erweiterung, um die richtigen Regeln auszugeben. Dies

wird durch den Aufruf dieser Funktion erreicht.

```
static void save(const struct ipt_ip *ip, const struct ipt_entry_match *match)
{
   const struct ipt_ipaddr_info *info = (const struct ipt_ipaddr_info *)match->data;
```

Wir geben die Quell−Adresse aus, wenn sie Teil der Regel ist.

```
   if (info->flags & IPADDR_SRC) {
      if (info->flags & IPADDR_SRC_INV)
         printf("! ");
      printf("--ipsrc ");
      print_ipaddr((u_int32_t *)&info->ipaddr.src);
   }
```

Wir geben die Ziel−Adresse aus, wenn sie Teil der Regel ist.

```
   if (info->flags & IPADDR_DST) {
      if (info->flags & IPADDR_DST_INV)
         printf("! ");
      printf("--ipdst ");
      print_ipaddr((u_int32_t *)&info->ipaddr.dst);
   }
}
```

1.2.3 Druckfunktion

Im gleichen Sinne wie die vorherige, versucht diese Funktion, Information über die Regel auszugeben. Sie wird von 'iptables −L' aufgerufen. Später im Text werden wir den Zweck von 'ipt_entry_match *match' sehen, aber Sie wissen bestimmt schon etwas darüber.

```
static void print(const struct ipt_ip *ip,
                  const struct ipt_entry_match *match,
                  int numeric)
{
   const struct ipt_ipaddr_info *info = (const struct ipt_ipaddr_info *)match->data;

   if (info->flags & IPADDR_SRC) {
         printf("src IP ");
      if (info->flags & IPADDR_SRC_INV)
         printf("! ");
      print_ipaddr((u_int32_t *)&info->ipaddr.src);
   }

   if (info->flags & IPADDR_DST) {
      printf("dst IP ");
      if (info->flags & IPADDR_DST_INV)
         printf("! ");
      print_ipaddr((u_int32_t *)&info->ipaddr.dst);
   }
}
```

1.2.4 Funktion letzter Test

Diese Funktion ist eine letzte Gelegenheit für Sicherheitsprüfungen. Sie wird direkt nach der Argumentauswertung aufgerufen, wenn der Anwender eine neue Regel eingibt.

```
static void final_check(unsigned int flags)
{
   if (!flags)
      exit_error(PARAMETER_PROBLEM, "ipt_ipaddr: Invalid parameters.");
}
```

1.2.5 Auswerte−Funktion

Dies ist die wichtigste Funktion, weil wir hier verifizieren, dass die Argumente korrekt benutzt werden und wir Informationen eintragen, die wir mit dem Kernel−Teil gemeinsam verwenden. Sie wird jedesmal aufgerufen, wenn ein Argument gefunden wird, d. h. wenn der Anwender zwei Argumente eingibt, wird sie zweimal mit dem Argument−Code in der Variablen 'c' aufgerufen.

```
static int parse(int c, char **argv, int invert, unsigned int *flags,
                 const struct ipt_entry *entry,
                 unsigned int *nfcache,
                 struct ipt_entry_match **match)
{
```

Wir benutzten diese spezielle Struktur, um Informationen zu erhalten, die wir mit dem Kernel−Teil teilen. Der 'Match'−Zeiger wird an einige Funktionen weitergereicht, damit wir mit der gleichen Datenstruktur arbeiten können. Sobald die Regel geladen ist, wird dieser Zeiger in den Kernel−Bereich geladen. Auf diese Weise weiss das Kernel−Modul, um welche Analyse der Anwender gebeten hat (und darum geht es doch, oder?).

```
   struct ipt_ipaddr_info *info = (struct ipt_ipaddr_info *)(*match)->data;
```

Jedes Argument korrespondiert mit einem einzelnen Wert, so dass wir spezielle Aktionen entsprechend den eingegebenen Argumenten durchführen können . Wir werden später im Text sehen, wie wir Argumente auf Werte abbilden.

```
   switch(c) {
```

Zunächst testen wir, ob das Argument mehr als einmal benutzt wurde. Wenn dies der Fall zu sein scheint, rufen wir das in <*iptables/iptables.c*> definierte 'exit_error()', das unmittelbar mit dem Status−Flag 'PARAMETER_PROBLEM' (definiert in <*iptables/include/iptables_common.h*>), zurückkehrt. Ansonsten setzen wir 'flags' und 'info−>flags' auf den in unserer Header−Datei definierten Wert 'IPADDR_SRC'. Diese Header−Datei werden wir später sehen.

Obwohl beide Flags offensichtlich den gleichen Zweck haben, ist dies wirklich nicht der Fall. Der Bereich von 'flags' ist nur diese Funktion und 'info−>flags' ist ein Teilfeld unserer Struktur, die mit dem Kernel−Teil gemeinsam benutzt wird.

```
      case '1':
         if (*flags & IPADDR_SRC)
            exit_error(PARAMETER_PROBLEM, "ipt_ipaddr: Only use --ipsrc once!");
         *flags |= IPADDR_SRC;
         info->flags |= IPADDR_SRC;
```

Wir verifizieren, ob das Invert−Flag, '!', eingegeben wurde und setzen entsprechende Informationen in 'info−>flags'.
Als nächstes rufen wir 'parse_ipaddr', eine interne Funktion, die für dieses Programm−Skelett geschrieben wurde, um eine Zeichenkette mit der IP−Adresse in einen 32−Bit−Wert zu wandeln.

```
         if (invert)
            info->flags |= IPADDR_SRC_INV;

         parse_ipaddr(argv[optind-1], &info->ipaddr.src);
         break;
```

Auf die gleiche Art testen wir auf mehrfache Verwendung und setzen entsprechende Flags.

```
      case '2':
         if (*flags & IPADDR_DST)
            exit_error(PARAMETER_PROBLEM, "ipt_ipaddr: Only use --ipdst once!");
         *flags |= IPADDR_DST;
         info->flags |= IPADDR_DST;
         if (invert)
            info->flags |= IPADDR_DST_INV;

         parse_ipaddr(argv[optind-1], &info->ipaddr.dst);
         break;

      default:
         return 0;
   }

   return 1;
}
```

1.2.6 Optionen−Struktur

Wir haben bereits diskutiert, das jedes Argument auf einen einzelnen Wert abgebildet wird. Die 'struct option' ist der bessere Weg, dies zu erreichen. Zu weiteren Informationen über diese Struktur, empfehle ich Ihnen sehr, 'man 3 getopt' zu lesen.

```
static struct option opts[] = {
   { .name = "ipsrc",   .has_arg = 1,   .flag = 0,   .val = '1' },
   { .name = "ipdst",   .has_arg = 1,   .flag = 0,   .val = '2' },
   { .name = 0 }
};
```

1.2.7 Initialisierungs−Funktion

Diese Initialisierungsfunktion wird benutzt, um einige spezielle Sachen wie das netfilter−Cache−System einzurichten. Es ist jetzt nicht sehr wichtig zu wissen, wie das genau funktioniert.

```
static void init(struct ipt_entry_match *m, unsigned int *nfcache)
{
   /* Can't cache this */
   *nfcache |= NFC_UNKNOWN;
}
```

1.2.7 Hilfe−Funktion

Diese Funktion wird von 'iptables −m match_name −h' aufgerufen, um die verfügbaren Argumente anzuzeigen.

```
static void help(void)
{
   printf (
             "IPADDR v%s options:\n"
             "[!] --ipsrc <ip>\t\t The incoming ip addr matches.\n"
             "[!] --ipdst <ip>\t\t The outgoing ip addr matches.\n"
             "\n", IPTABLES_VERSION
          );
}
```

1.2.8 Die Header−Datei 'ipt_ipaddr.h'

In dieser Datei definieren wir die von uns benötigten Sachen.

```
#ifndef _IPT_IPADDR_H
#define _IPT_IPADDR_H
```

Wir haben bereits gesehen, dass wir Flags auf bestimmte Werte setzen.

```
#define IPADDR_SRC   0x01     /* Match source IP addr */
#define IPADDR_DST   0x02     /* Match destination IP addr */

#define IPADDR_SRC_INV  0x10  /* Negate the condition */
#define IPADDR_DST_INV  0x20  /* Negate the condition */
```

Die Struktur 'ipt_ipaddr_info' ist diejenige, welche in den Kernel−Teil kopiert wird.

```
struct ipt_ipaddr {
   u_int32_t src, dst;
};

struct ipt_ipaddr_info {

   struct ipt_ipaddr ipaddr;

   /* Flags from above */
   u_int8_t flags;

};

#endif
```

1.3 Zusammenfassung Kapitel 1

Im ersten Teil haben wir den Zweck der iptables−Bibliothek diskutiert. Wir haben die Interna jeder Funktion besprochen und wie die Struktur 'ipt_ipaddr_info' benutzt wird, um Informationen zu speichern, die zur weiteren Verwendung in den Kernel−Teil kopiert werden. Wir schauten ausserdem auf die iptables−Struktur und wie wir unsere neue Bibliothek registrieren. Sie sollten bedenken, dass dies nur ein Programm−Beispiel ist, das mir hilft, Ihnen zu zeigen, wie dieses Rahmenwerk arbeitet. Weiterhin sind 'ipt_ipaddr_info' und ähnliche Sachen nicht Teil von iptables/netfilter, sondern Bestandteil dieses Beispiels.

2.0 Das netfilter−Modul

Zweck eines Match−Moduls ist es, jedes empfangene Paket zu inspizieren und zu entscheiden, ob es entsprechend unserer Kriterien übereinstimmt oder nicht. Das Modul hat die folgenden Möglichkeiten, dieses zu tun:

- Empfange jedes Paket, welches die mit dem Match−Modul verbundene Tabelle trifft
- Teile netfilter mit, ob unser Modul auf das Paket zutrifft

2.1 verfügbare Strukturen und Funktionen

Zunächst einige grundlegende Strukturen. Diese Struktr ist definiert in *<linux/netfilter_ipv4/ip_tables.h>*. Wenn Sie daran interessiert sind, mehr über diese und die vorher für iptables vorgestellte Struktur zu lernen, sollten Sie sich netfilter hacking howto ansehen, das von Rusty Russell und Harald Welte geschrieben wurde.

```
struct ipt_match
{
   struct list_head list;

   const char name[IPT_FUNCTION_MAXNAMELEN];

   /* Return true or false: return FALSE and set *hotdrop = 1 to
           force immediate packet drop. */
   /* Arguments changed since 2.4, as this must now handle
           non-linear skbs, using skb_copy_bits and
           skb_ip_make_writable. */
   int (*match)(const struct sk_buff *skb,
           const struct net_device *in,
           const struct net_device *out,
           const void *matchinfo,
           int offset,
           int *hotdrop);

   /* Called when user tries to insert an entry of this type. */
   /* Should return true or false. */
   int (*checkentry)(const char *tablename,
           const struct ipt_ip *ip,
           void *matchinfo,
           unsigned int matchinfosize,
           unsigned int hook_mask);

   /* Called when entry of this type deleted. */
   void (*destroy)(void *matchinfo, unsigned int matchinfosize);

   /* Set this to THIS_MODULE. */
   struct module *me;
```

```
};
```

2.2 Innerhalb des Programm−Skeletts

2.2.1 Initialisierung

Wir initialisieren die allgemeinen Felder in der Struktur 'ipt_match'.

```
static struct ipt_match ipaddr_match
= {
```

'Name' ist die Zeichenkette mit dem Dateinamen Ihres Moduls (z. B. ipt_ipaddr).

```
        .name       = "ipaddr",
```

Die nächsten Felder enthalten Rücksprungfunktionen, die das Rahmenwerk benutzen wird. 'Match' wird aufgerufen, wenn ein Paket an Ihr Modul weitergereicht wird.

```
        .match      = match,
        .checkentry = checkentry,
        .me         = THIS_MODULE,
};
```

Die Init−Funktion Ihres Kernelmoduls muss 'ipt_register_match()' aufrufen mit einem Zeiger auf eine Struktur 'struct ipt_match', um sich beim netfiler−Rahmenwerk zu registrieren. Diese Funktion wird beim Laden des Moduls aufgerufen.

```
static int __init init(void)
{
        printk(KERN_INFO "ipt_ipaddr: init!\n");
        return ipt_register_match(&ipaddr_match);
}
```

Beim Entladen des Moduls wird diese Funktion aufgerufen. Hier deregistrieren wir unser Match−Modul.

```
static void __exit fini(void)
{
        printk(KERN_INFO "ipt_ipaddr: exit!\n");
        ipt_unregister_match(&ipaddr_match);
}
```

Wir übergeben ihnen Funktionen, die beim Laden und Entladen des Moduls aufgerufen werden.

```
module_init(init);
module_exit(fini);
```

2.2.2 'match'−Funktion

Der Linux−TCP/IP−Stack verfügt über 5 netfilter−Einsprungstellen. Wenn ein Paket hereinkommt, übergibt der Stack das Paket an die entsprechende Einsprungstelle, welche dann jede Tabelle durchläuft, die dann ihrerseits jede Regel testet. Wenn dann Ihr Modul mit dem Paket an die Reihe kommt, kann es endlich seine Aufgabe erfüllen.

```
static int match(const struct sk_buff *skb,
                 const struct net_device *in,
                 const struct net_device *out,
                 const void *matchinfo,
                 int offset,
                 const void *hdr,
                 u_int16_t datalen,
                 int *hotdrop)
{
```

Sie erinnern sich hoffentlich daran, was wir im Benutzerbereich getan haben! :). Nun übertragen wir die vom Benutzerbereich kopierte Struktur in unsere eigene.

```
        const struct ipt_skeleton_info *info = matchinfo;
```

'skb' enthält das Paket, das wir untersuchen wollen. Zu weiteren Informationen über diese mächtige Struktur, die überall im LInux−TCP/IP−Stack benutzt wird, hat Harald Welte einen exzellenten Artikel (ftp://ftp.gnumonks.org/pub/doc/skb−doc.html) geschrieben.

```
   struct iphdr *iph = skb->nh.iph;
```

Hier geben wir nur einige lustige Sachen aus, um zu sehen, wie sie aussehen. Das Makro 'NIPQUAD' wird benutzt, um eine IP−Adresse in lesbarer Form auszugeben, definiert in *<linux/include/linux/kernel.h>*.

```
   printk(KERN_INFO "ipt_ipaddr: IN=%s OUT=%s TOS=0x%02X "
                    "TTL=%x SRC=%u.%u.%u.%u DST=%u.%u.%u.%u "
                    "ID=%u IPSRC=%u.%u.%u.%u IPDST=%u.%u.%u.%u\n",

                    in ? (char *)in : "", out ? (char *)out : "", iph->tos,
                    iph->ttl, NIPQUAD(iph->saddr), NIPQUAD(iph->daddr),
                    ntohs(iph->id), NIPQUAD(info->ipaddr.src), NIPQUAD(info->ipaddr.dst)
          );
```

Wenn das Argument '−−ipsrc' übergeben wurde, schauen wir, ob die Quelladresse mit der in der Regel angegebenen übereinstimmt. Wir vergessen nicht, das Invert−Flag '!' zu berücksichtigen. Wenn es keine Übereinstimmung gibt, geben wir das Urteil 0 zurück.

```
   if (info->flags & IPADDR_SRC) {
      if ( (ntohl(iph->saddr) != ntohl(info->ipaddr.src)) ^ !!(info->flags & IPADDR_SRC_INV) ) {

         printk(KERN_NOTICE "src IP %u.%u.%u.%u is not matching %s.\n",
                            NIPQUAD(info->ipaddr.src),
                            info->flags & IPADDR_SRC_INV ? " (INV)" : "");
         return 0;
      }
   }
```

Hier machen wir das gleiche mit der Ausnahme, das wir auf die Zieladresse schauen, wenn '−−ipdst' eingegeben wurde.

```
   if (info->flags & IPADDR_DST) {
```

```
      if ( (ntohl(iph->daddr) != ntohl(info->ipaddr.dst)) ^ !!(info->flags & IPADDR_DST_INV) )  {

         printk(KERN_NOTICE "dst IP %u.%u.%u.%u is not matching%s.\n",
                            NIPQUAD(info->ipaddr.dst),
                            info->flags & IPADDR_DST_INV ? " (INV)" : "");
         return 0;
      }
   }
```

Wenn beides nicht zutrifft, geben wir das Urteil 1 zurück, was bedeutet, dass das Paket übereinstimmt.

```
   return 1;
}
```

2.2.3 Funktion 'checkentry'

Checkentry wird überwiegend benutzt als letzte Möglichkeit für Sicherheitstest. Es ist etwas schwer zu verstehen, wenn es aufgerufen wird. Zur Erläuterung siehe dieses Posting (http://www.mail−archive.com/netfilter−devel@lists.samba.org/msg00625.html). Dies wird auch im 'netfilter hacking howto' erläutert.

```
static int checkentry(const char *tablename,
                           const struct ipt_ip *ip,
                           void *matchinfo,
                           unsigned int matchsize,
                           unsigned int hook_mask)
{
   const struct ipt_skeleton_info *info = matchinfo;

   if (matchsize != IPT_ALIGN(sizeof(struct ipt_skeleton_info))) {
      printk(KERN_ERR "ipt_skeleton: matchsize differ, you may have forgotten to recompile me.\n"
      return 0;
   }

   printk(KERN_INFO "ipt_skeleton: Registered in the %s table, hook=%x, proto=%u\n",
                    tablename, hook_mask, ip->proto);

   return 1;
}
```

2.3 Zusammenfassung Kapitel 2

In diesem zweiten Teil behandelten wir das netfilter−Modul und wie es mittels einer speziellen Struktur registriert wird. Zusätzlich diskutierten wir, wie man eine spezifische Situation entsprechend der vom Benutzer−Teil bereitgestellten Kriterien auf Übereinstimmung testet.

3.0 mit iptables/netfilter spielen

Wir haben gesehen, wie man ein neues iptables/netfilter−Match−Modul schreibt. Nun wollen wir es unserem Kernel hinzufügen, um damit zu spielen. Hier setzte ich voraus, dass Sie einen Kernel erstellen bzw kompilieren können. Zunächst holen Sie sich die match−Dateien für dieses Programm−Skelett von der

Download−Seite für diesen Artikel.

3.1 iptables

Wenn Sie nicht über die iptables−Quellen verfügen, können Sie sie von ftp://ftp.netfilter.org/pub/iptables/ abrufen. Dann müssen Sie 'libipt_ipaddr.c' nach <*iptables/extensions/*> kopieren.

Dies ist eine Zeile von <*iptables/extensions/Makefile*> in der Sie 'ipaddr' hinzufügen müssen.

```
PF_EXT_SLIB:=ah addrtype comment connlimit connmark conntrack dscp ecn
esp hashlimit helper icmp iprange length limit ipaddr mac mark
multiport owner physdev pkttype realm rpc sctp standard state tcp tcpmss
tos ttl udp unclean CLASSIFY CONNMARK DNAT DSCP ECN LOG MARK MASQUERADE
MIRROR NETMAP NOTRACK REDIRECT REJECT SAME SNAT TARPIT TCPMSS TOS TRACE
TTL ULOG
```

3.2 Kernel

Zunächst müssen Sie 'ipt_ipaddr.c' in <*linux/net/ipv4/netfilter/*> und 'ipt_ipaddr.h' nach <*linux/include/linux/netfilter_ipv4/*> kopieren. Einige von Ihnen benutzen immer noch Linux 2.4, daher zeige ich die zu editierenden Dateien für 2.4 und 2.6.

Für 2.4 editieren Sie <*linux/net/ipv4/netfilter/Config.in*> und fügen die fett dargestellte Zeile hinzu.

```
# The simple matches.
  dep_tristate '  limit match support' CONFIG_IP_NF_MATCH_LIMIT $CONFIG_IP_NF_IPTABLES
  dep_tristate '  ipaddr match support' CONFIG_IP_NF_MATCH_IPADDR $CONFIG_IP_NF_IPTABLES
```

Dann editieren Sie <*linux/Documentation/Configure.help*> und fügen den fett dargestellten Text hinzu. Ich habe etwas Text kopiert, um Ihnen dabei zu helfen, wo Sie Ihren hinzufügen können.

```
limit match support
CONFIG_IP_NF_MATCH_LIMIT
  limit matching allows you to control the rate at which a rule can be
  ...
ipaddr match support
CONFIG_IP_NF_MATCH_IPADDR
  ipaddr matching. etc etc.
```

Zum Schluss müssen Sie diese fett dargestellte Zeile in <*linux/net/ipv4/netfilter/Makefile*> einfügen.

```
# matches
obj-$(CONFIG_IP_NF_MATCH_HELPER) += ipt_helper.o
obj-$(CONFIG_IP_NF_MATCH_LIMIT) += ipt_limit.o
obj-$(CONFIG_IP_NF_MATCH_IPADDR) += ipt_ipaddr.o
```

Für 2.6 sind die zu editierenden Dateien <*linux/net/ipv4/netfilter/Kconfig*> und <*linux/net/ipv4/netfilter/Makefile*>.

Zusammenfassung

Nun müssen Sie nur noch rekompilieren und das hinzufügen, was ich vergessen habe, Ihnen zu erzählen. Fröhliches Hacken!!

Danke an Samuel Jean.

2005−07−23, generated by lfparser_pdf version 2.51