

Inhaltsverzeichnis

1	Netfilter	1
1.1	Einführung	1
1.2	Der „Linux Socket Buffer“ sk_buff	1
1.3	Die Netfilter Hooks	2
1.4	Reise eines IPv4 Paketes durch den Linux Kernel	4
1.5	iptables	5
1.5.1	Das iptables Programm	5
1.5.2	Die <i>filter</i> Tabelle	7
1.5.3	Network Address Translation <i>NAT</i>	8
1.5.4	Paket „mangling“	9
1.6	Connection Tracking	9
1.6.1	Stateful Firewall	11
1.6.2	Conntrack Helper	12
1.6.3	NAT im Kernel	13
1.6.4	NAT Helper	14
1.7	Pakete einreihen	15
1.8	Ausgewählte iptables Module	15
1.8.1	Das <i>owner</i> Modul	15
1.8.2	Das <i>length</i> Modul	15
1.8.3	Das DSCP Modul	15
2	Danksagung	17

1 Netfilter

1.1 Einführung

Netfilter[2] bietet eine Infrastruktur für Paketbehandlung im Linux Kernel[4] und unterstützt die Protokolle¹ IPv4, IPv6 und DECnet. Das Netfilter Paket wurde entwickelt um den komplexen und sehr fehleranfälligen Firewall Code der Kernelversion 2.2 zu ersetzen. Das Projekt wurde von Paul „Rusty“ Russel[5] gestartet und wird jetzt von *Netfilter Core Team*[2] weiterentwickelt.

1.2 Der „Linux Socket Buffer“ *sk_buff*

Jedes Netzwerkpaket im Linux Kernel wird durch die *sk_buff*-Struktur repräsentiert, auf welcher die Funktionen des Linux Netzwerkstacks angewendet werden können. Diese Struktur bietet zum einen eine flexible Handhabung der Daten wenn diese zwischen zwei Netzwerkschichten übergeben werden müssen. Hierzu besitzt diese Struktur einen *head* und einen *data* Zeiger, sowie einen *tail* und einen *end* Zeiger, die jeweils Anfang und Ende des Pufferbereichs bzw. des Datenbereichs markieren. So wird durch Verschieben des *data* Pointers, mit wenig Aufwand den Protokoll-Header des darunterliegenden Protokolls für das darüberliegende entfernt, ohne Speicher kopieren zu müssen. Zum anderen bietet die *sk_buff* Struktur alle wichtigen Metadaten der einzelnen Netzwerkschichten in aufbereiteter Form.

Eine kurze Auflistung der wichtigsten Felder des *sk_buff*:

- *next/prev* - Listenzeiger
- *list* - Zeiger auf die aktuelle Liste
- *sock* - Zeiger auf den erzeugenden Socket
- *dev* - Zeiger auf ein *struct net_device*, von dem das Paket kommt oder für das das Paket bestimmt ist
- *real_dev* - Erweiterung zu dev (wird für VLAN gebraucht)
- *h* - Transport Layer Header; *h.th* für TCP, *h.uh* für UDP ...

¹Stand Linux Kernel 2.6.0-test

- *nh* - Network Layer Header; *nh.iph* für IPv4
- *mac* - Link Layer Header
- *dst* - Zeiger auf ein *struct dst_entry*; Informationen über den weiteren Weg durch den Stack. Wird beim Routing erzeugt.
- *cb* - Control Buffer (privat)
- *len, data_len* - Informationen über die Größe der Daten
- *csum* - Checksummen Informationen
- *protocol* - Ethernet Protokoll ID (0x0800 für IP)
- *data, tail* - Zeiger auf Beginn und Ende des Datenbereichs
- *head, end* - Zeiger auf Beginn und Ende des Pufferbereichs
- *nfct* - Netfilter Conntrack. Enthält einen Zeiger auf das *infos* Array einer *ip conntrack* Struktur
- *nfmark* - Feld zur Markierung der Pakete
- *nfcache* - Netfilter Caching Informationen

1.3 Die Netfilter Hooks

Die Netfilter Infrastruktur im Linux Kernel besteht zum einen aus *Netfilter-Hooks* und zum anderen aus Funktionen zur Registrierung und Verwaltung dieser. *Netfilter-Hooks* sind „Kontrollpunkte“ im Linux Netzwerkcode, für welche sich Kernelmodule registrieren können. Diese Kontrollpunkte sind so verteilt, dass alle Netzwerkpakete mindestens einen passieren müssen.

Im Folgendem werden die Netfilter Hooks für das Protokoll IPv4 beschrieben, da dies zur Zeit das am meisten genutzte ist.

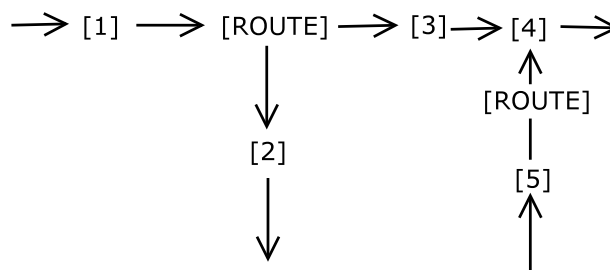


Abb. 1.1: Netfilter Hooks IPv4

Abbildung 1.1 zeigt schematisch den Paketfluss durch den IPv4 Netzwerkstack. Links kommen Pakete vom Netzwerk, rechts gehen sie zurück ins Netzwerk. Unten ist der lokale Rechner bzw. die lokalen Prozesse.

Die fünf Hooks in Abb. 1.1 heißen:

1. NF_IP_PRE_ROUTING
2. NF_IP_LOCAL_IN
3. NF_IP_FORWARD
4. NF_IP_POST_ROUTING
5. NF_IP_LOCAL_OUT

Pakete, die von diesem Rechner nur weitergeleitet werden sollen, passieren Hook #1. Entscheidet das Routing, dass das Paket nicht für den lokalen Rechner bestimmt ist, passiert es Hook #3 und #4 und verlässt den Rechner wieder.

Für die lokale Zustellung bestimmte Pakete passieren Hook #1 und #2 und werden dann an den lokalen Prozess übergeben.

Lokal erstellte Pakete passieren zuerst Hook #5, darauf folgend wird eine Routingentscheidung getroffen und die Pakete passieren Hook #4. Anschließend verlässt das Paket den Rechner.

Zur Registrierung der Hooks stehen die Funktionen *nf_register_hook()* und *nf_unregister_hook()* zur Verfügung. Um einen Netfilter-Hook zu registrieren, erzeugt man zunächst eine *nf_hook_ops* Struktur, die einen Funktionszeiger, das verwendete Protokoll, den Hook und eine Priorität enthält. Die durch den Funktionszeiger spezifizierte Funktion, wird dann für jedes, den registrierten Hook passierende Paket, aufgerufen und erhält einen Zeiger auf das Paket. Sie kann das Paket analysieren und muss abschließend ein Urteil über das Paket zurückgeben. Dabei sind folgende Rückgabewerte möglich:

- NF_ACCEPT = das Paket kann den Hook passieren
- NF_DROP = verwirft das Paket
- NF_STOLEN = das Paket darf nicht weitergereicht werden, das Modul ist jetzt für das Paket verantwortlich
- NF_QUEUE = das Paket wird an registrierte Queue-Handler übergeben
- NF_REPEAT = diesen Hook nochmal aufrufen

1.4 Reise eines IPv4 Paketes durch den Linux Kernel

Empfängt die Netzwerkkarte einen *Ethernet Frame* mit passender MAC Adresse², so schreibt sie den Inhalt in einen Puffer und löst einen Interrupt aus. Eine vom Kartentreiber registrierte Funktion wird vom Kernel Interrupt-Handler aufgerufen und führt dann folgende Aktionen mit abgeschalteten Interrupts durch:

- reserviert Speicher für eine neue *sk_buff* Struktur
- schreibt die Daten aus dem Kartenspeicher in die *sk_buff* Struktur
- ruft *netif_rx()* auf
- wenn *netif_rx()* fertig ist, werden die Interrupts wieder eingeschaltet und die Funktion endet

Die *netif_rx()* Funktion bereitet den Kernel auf die weitere Verarbeitung des Pakets vor, indem die *sk_buff* Struktur in die Liste³ eingehender Pakete hinzugefügt wird und ein NET_RX *Softirq* gesetzt wird.

Softirqs werden bearbeitet, wenn entweder ein Hardware-Interrupt bearbeitet wurde, ein System-Call aufgerufen wurde oder der Kernel-Scheduler neu Prozessorzeit vergibt.

War ein NET_RX *Softirq* gesetzt, so wird die Funktion *net_rx_action()* aufgerufen. Diese Funktion entnimmt einen *sk_buff* aus der Paketliste und prüft in den zwei Listen, *ptype_all* und *ptype_base*, ob sie dem Paket entsprechende Protokoll-Handler enthalten.

Protokoll Handler registrieren sich zusammen mit einer neuen Adressfamilie beim Systemstart oder durch Hinzufügen von Kernelmodulen. Sie spezifizieren einen Zeiger auf eine Funktion, die das ankommende Paket weiter behandeln soll. Im Falle von IPv4 ist das *ip_rcv()*.

Die Funktion *ip_rcv()* überprüft zuerst, ob das Paket tatsächlich für diesen Host bestimmt ist. Dies ist nur dann der Fall, sich die Netzwerkkarte im „promiscuous Mode“ befindet. Sind Paket-Header und Checksumme in Ordnung, durchläuft das Paket den ersten Netfilter Hook (NF_IP_PRE_ROUTING) und wird dann in *ip_rcv_finish()* weiter untersucht. Hier wird zunächst die Routingfunktion *ip_route_input()* aufgerufen um den weiteren Weg des Pakets zu bestimmen. Soll das Paket weitergeleitet werden, so geht es an *ip_forward()*, sonst an *ip_local_deliver()*. *ip_mr_input()* behandelt Multicast Paketen. Im Falle eines Fehlers, versendet *ip_error()* die entsprechende ICMP Fehlermeldung und zerstört den *sk_buff*.

Ist das Paket zur lokalen Zustellung bestimmt worden, werden, falls notwendig, in *ip_local_deliver()* Fragmente zusammengesetzt. Bevor *ip_local_deliver_finish()* aufgerufen wird, passiert das Paket einen weiteren Netfilter Hook (NF_IP_LOCAL_IN). Vor der Weitergabe des Paketes an den nächsten Netzwerklayer, wird noch nach einen wartenden „RAW IP Socket“ gesucht. Falls eine passender gefunden wird, bekommt die *raw_v4_input()* Funktion einen *Clone* des Pakets.

Anschließend muss, ähnlich wie in *net_rx_action()*, das richtige Protokoll gefunden werden. Für IP Pakete können das TCP, UDP, ICMP und IGMP sein. Für jedes der Protokolle steht eine

²oder ein Broadcast Paket, oder Pakete im Promiscuous Mode

³CPU gebunden

entsprechende Funktion zur Verfügung (zBsp. *tcp_v4_rcv()*, *udp_rcv()*, ...), die das Paket weiter verarbeitet. Netfilter und *iptables* arbeiten jedoch nur mit Paketen des IP Layers⁴.

Wurde ein Paket an *ip_forward()* übergeben, wird zuerst die TTL überprüft. Ist sie ≤ 1 so wird das Paket verworfen, sonst um 1 dekrementiert. Bevor die Funktion *ip_forward_finish()* aufgerufen wird, passiert das Paket den Netfilter Hook NF_IP_FORWARD. *ip_forward_finish()* setzt eventuell noch fehlende IP Optionen und übergibt das Paket an *ip_send()*, die eventuell Fragmentierung veranlasst. Die nächste Station, *ip_finish_output()*, setzt im *sk_buff* das Netzdevice gemäß der Routingentscheidung und, bevor *ip_finish_output2()* aufgerufen wird, muss das Paket noch den letzten Netfilter Hook (NF_IP_POST_ROUTING) passieren. *ip_finish_output2()* übergibt dann schließlich die Paketdaten an die darunterliegende Schicht.

Lokal erzeugte Pakete, können durch vier verschiedene Funktionen den IP Netzwerklayer Layer betreten (*ip_build_and_send_pkt()*, *ip_queue_xmit()*, *ip_build_xmit()*, *ip_queue_xmit2()*). Im IP Layer muss zunächst ein IP Header dem *sk_buff* hinzugefügt werden, bevor der Netfilter Hook (NF_IP_LOCAL_OUT) passiert wird. Nach dem Passieren des Hooks wird nochmal überprüft, ob durch den Hook die Route verändert wurde. Anschließend verlässt das Paket über *ip_finish_output()* und *ip_finish_output2()* den IP Layer.

1.5 iptables

iptables sind Tabellen zur Paketauswahl im Linux Kernel und stellen, ähnlich zu Netfilter, Infrastruktur im Linux Kernel bereit. Kernelmodule können Tabellen registrieren, die dann von Paketen durchlaufen werden müssen. Eine solche Tabelle besteht aus Ketten (*chains*) von Regeln, die einzeln abgearbeitet werden. Eine solche Kette repräsentiert dabei den Regelsatz für jeweils einen Netfilter-Hook. Jede dieser Regeln besteht aus *match(es)*, welche Pakete, die auf diese Regel passen, beschreiben und höchstens einem *target*, das das Ziel dieser Regel beschreibt. Eine Regel kann, für spezielle Zwecke, auch entweder kein *match* oder kein *target* enthalten.

Drei solcher Tabellen werden dabei automatisch erzeugt. Eine zur Paketauswahl „*filter*“, eine zur Network Address Translation „*nat*“ und eine zur allgemeinen Behandlung von Paketen „*mangle*“.

1.5.1 Das iptables Programm

Um Regeln einer Tabelle hinzuzufügen oder eine Regel zu ändern benötigt man das *iptables* „userspace“ Programm. Eine *iptables* Regel in einer *chain* hat folgende Eigenschaften:

- *match(es)* - bestimmt Ziel-/Quelladresse, Ziel-/Quellport ...
- *target* - Ziel oder Aktion wenn das Paket passt

⁴Netfilter unterstützt noch weitere Layer 3 Protokolle

Ein *iptables* Kommando muss folgende Informationen enthalten:

- welche Tabelle soll bearbeitet werden
- welche *chain* soll bearbeitet werden
- eine mögliche Operation *anfügen, löschen, ersetzen ...*
- *match(es)*
- höchstens ein *target*

Eine Regel immer zumindest ein *target* oder ein *match* enthalten.

Mögliche *targets*:

- ACCEPT = erlaube das Paket
- DROP = verwerfe das Paket (silently)
- QUEUE = Paket in eine Liste einfügen (s.u.)
- RETURN = Rücksprung zur aufrufenden *chain*
- foobar = springe zu einem selbstdefinierten *target*
- REJECT = verwerfe das Paket und informiere den Sender
- LOG = Logge den Vorgang mit *syslog* und lasse das Paket passieren
- ULOG = sende das Paket an einen „userspace“ Log-Prozess
- MIRROR = ändere Quell- und Zieladresse und verschicke das Paket erneut (zu Testzwecken)

Eine Auswahl von möglichen *matches*:

- -p protocol (tcp/udp/icmp/...)
- -s source address
- -d destination address
- -i incoming interface
- -o outgoing interface
- -dport destination port
- -sport source port
- -state (NEW,ESTABLISHED,RELATED,INVALID)
- -mac-source source MAC address

- `-mark nfmark value`
- `-tos TOS value of the packet`
- `-ttl ttl value of the packet`
- `-limit (limit the rate of this packet to a certain amount of pkts/timeframe)`

Die Syntax eines *iptables* Befehls sieht wie folgt aus:

```
iptables -t table -operation chain -j target match(es)
```

Ein paar einfache Beispiele:

```
iptables -t filter -A INPUT -j ACCEPT -p tcp --dport ssh
iptables -t filter -A INPUT -j DROP
iptables -t filter -A OUTPUT -m owner --pid 741
iptables -L -v
```

Der erste Befehl fügt eine neue Regel (`-A` (append)) an die `INPUT` *chain* in die *filter* Tabelle ein. Die Regel erlaubt alle eingehenden TCP Pakete mit Zielport `ssh` (22).

Die zweite Regel stellt ein Beispiel einer Regel ohne *match* dar. Hier werden alle Pakete, die lokal zugestellt werden sollen verworfen.

Die dritte Regel ist ein Beispiel für eine Regel ohne ein *target*. Diese Regel ist sinnvoll um den ausgehenden Netzwerkverkehr der Anwendung mit der Prozess ID 741 zu zählen.

Die vierte Regel listet alle Tabellen, deren Regeln und die Anzahl der Pakete (bzw. deren Größe), die auf die jeweilige Regel gepasst haben.

1.5.2 Die *filter* Tabelle

Die *filter* Tabelle ist zur allgemeinen Paketauswahl gedacht. Diese Tabelle nutzt die Netfilter Hooks `NF_IP_LOCAL_IN` (#2), `NF_IP_FORWARD` (#3) und `NF_IP_LOCAL_OUT` (#5). Das heißt, dass jedes Paket, je nach Ursprung oder Ziel, nur an einer Stelle gefiltert werden kann. Lokal generierte Pakete passieren nur Hook #5, lokal bestimmte nur Hook #2 und weitergeleitete nur Hook #3.

Diese Tabelle enthält folgende Regelketten:

- `NF_IP_LOCAL_IN` = `INPUT` chain
- `NF_IP_LOCAL_OUT` = `OUTPUT` chain
- `NF_IP_FORWARD` = `FORWARD` chain

1.5.3 Network Address Translation NAT

Man kann zwei verschiedene Arten von NAT unterscheiden:

1. SNAT (source address NAT), MASQUERADE als spezielles Beispiel
2. DNAT (destination address NAT), REDIRECT als spezielles Beispiel

Hierzu werden für Adressänderungen von nicht-lokalen Paketen die `NF_IP_PRE_ROUTING` und `NF_IP_POST_ROUTING` Hooks genutzt und für das Ändern der Zieladresse von lokal generierten Paketen der `NF_IP_LOCAL_OUT` Hook.

Im Gegensatz zu der *filter* Tabelle, muss hier nur das erste Paket einer Verbindung die Tabelle durchlaufen. Das Ergebnis wird dann auf alle Pakete der gleichen Verbindung angewandt.

Ein Beispiel zu SNAT:

```
iptables -t nat -A POSTROUTING -j SNAT --to-source 10.0.0.1 -o eth0
```

Hierdurch werden alle Pakete, die über die Netzwerkschnittstelle `eth0` geschickt werden sollen, die Quelladresse `10.0.0.1` erhalten.

Das *target* SNAT benutzt man bei statischen IP Adressen. Hat man eine dynamische IP Adresse, sollte man das *target* MASQUERADE nutzen.

```
iptables -t nat -A POSTROUTING -j MASQUERADE -o ppp0
```

In diesem Beispiel wird der Verkehr über der Schnittstelle `ppp0` maskiert. Dies ist der gleiche Vorgang wie beim SNAT Target, mit dem Unterschied, dass die Quelladresse selbst von der Schnittstelle `ppp0` ermittelt wird.

Ein Beispiel zu DNAT:

```
iptables -t nat -A PREROUTING -j DNAT --to-destination 10.0.0.1:8080 -p tcp \
--dport 80 -i eth0
```

Hier werden alle Pakete, welche von der Netzwerkschnittstelle `eth0` kommen und für Port 80 bestimmt sind, zu der Adresse `10.0.0.1:8080` umgeleitet.

Für das DNAT *target* kann man das spezielle *target* REDIRECT nutzen.

```
iptables -t nat -A PREROUTING -j REDIRECT --to-port 8080 \
-i eth0 -p tcp --dport 80
```

In diesem Beispiel werden alle Webserveranfragen über `eth0` zum lokalen Port 8080 umgeleitet.

1.5.4 Paket „mangling“

Mit der *mangle* Tabelle erlaubt die Manipulation von Paketdaten. Zur Zeit stehen unter anderem folgende *targets* zur Verfügung:

- TOS - erlaubt das Ändern der TOS Bits im Header (veraltet)
- DSCP - Nachfolger von TOS; kann frei benutzt werden
- TCPMMS - max Segment Size
- ECN - ECN-capable Transport RFC 2481
- TTL - erlaubt das Ändern des TTL Felds im Header
- MARK - setzt das skb->nfmark Feld

```
iptables -t mangle -A OUTPUT -j DSCP 1 -p tcp --dport 22
```

1.6 Connection Tracking

Das Connection Tracking verfolgt alle ein- und ausgehenden Verbindungen und speichert diese in einer Hashtabelle. Conntrack benutzt, mit jeweils höchster Priorität, die beiden Hooks `NF_IP_LOCAL_OUT` und `NF_IP_PRE_ROUTING` um alle Pakete zu bewerten, bevor sie anderweitig manipuliert werden können. Connection Tracking ist einerseits essentieller Bestandteil von NAT aber andererseits auch als eigenständiges Modul (State-Modul) implementiert. Des Weiteren ist Connection Tracking aber auch ein konkretes Beispiel für die Nutzung der Netfilter Infrastruktur.

Jedes eingehende Paket (`PRE_ROUTING`, für ausgehende Pakete ist der Vorgang analog) wird zunächst in eine *ip_conntrack_tuple* umgewandelt und anschließend wird nach einer, schon bestehenden Verbindung, gesucht, die auf dieses Tupel passt. Ein solches Tupel enthält alle Merkmale einer Verbindung. Bei TCP sind das Quell- und Zieladresse und die jeweiligen Portnummern. Je nach Ergebnis der Suche wird das Paket mit einer der folgenden Konstanten bewertet:

- `IP_CT_NEW` - das ankommende Paket erzeugt eine neue Verbindung
- `IP_CT_ESTABLISHED` - das Paket gehört zu einer schon bestehenden Verbindung
- `IP_CT_ESTABLISHED + IP_CT_IS_REPLY` - bestehende Verbindung in die andere Richtung
- `IP_CT_RELATED` - das Paket steht in Beziehung mit einer schon bestehenden Verbindung
- `IP_CT_RELATED + IP_CT_IS_REPLY` - Rückrichtung der Verbindung
- `IP_CT_INVALID` - unbekannt/ungültig

Gehört das ankommende Paket einer schon bestehenden Verbindung an, so erhält man eine *tuple_hash* Struktur. Diese enthält zum einen das Tupel der Verbindung und zum anderen einen Zeiger auf eine *ip_conntrack* Struktur. Aufgrund der schon bestehenden Verbindung muss nur noch die Richtung des Pakets bestimmt werden. Für die Bewertung des Pakets ergeben sich nur die Möglichkeiten ESTABLISHED oder RELATED, falls es sich um eine erwartete Verbindung handelt, beide mit Angabe der Richtung.

Wurde keine bestehende Verbindung in der Hash-Tabelle gefunden, so muss zunächst eine neue *ip_conntrack* Struktur angelegt werden. Hierzu werden zunächst die beiden *tuple_hash* (eine für jede Richtung) Strukturen erzeugt und das Protokoll der Verbindung in die Struktur eingetragen. Anschließend muss überprüft werden ob es sich bei dieser Verbindung um eine bereits erwartete (*expectation*) handelt.

Falls es eine bereits erwartete Verbindung ist, werden *Master*-Verbindung und die neue Verbindung verlinkt, und die Erwartung (*expectation*) wird, da sie nun erfüllt ist, ausgetragen. Um die Verbindung später als RELATED wieder zu erkennen, wird noch das EXPECTED-Bit gesetzt.

Erfüllt das Paket keine *expectation*, wird nach einem, auf dieses Tupel⁵ passenden, registrierten *Helper* gesucht. Falls einer gefunden wurde, wird er ebenfalls in die *ip_conntrack* Struktur eingetragen.

Für ein Paket, das eine neue Verbindung erzeugt, wird die Bewertung NEW vergeben.

Nachdem die Bewertung des Pakets feststeht, wird diese Information in den *sk_buff* eingetragen.

Das Feld *nfct* der *sk_buff* Struktur enthält jetzt einen Zeiger auf das *infos*-Array des zugehörigen Conntrack. Mit Hilfe der Funktion *ip_conntrack_get()* erhält man aus dem *nfct*-Feld einmal die Bewertung des Pakets und auch einen Zeiger auf die zugehörige *ip_conntrack* Struktur. Diese Informationen werden später noch von NAT gebraucht.

Anschließend wird der Protokollabhängige Teil des Conntrack Codes ausgeführt. Es sind Funktionen für, die über IP liegenden Protokolle, UDP, TCP und ICMP implementiert. Hier wird, protokollabhängig, die Gültigkeit der Verbindung überprüft und eventuell noch Verbindungstimeouts aktualisiert. Bei TCP, zum Beispiel, wird, bei schon erhaltenem SYN Paket, auf ein entsprechendes ACK gewartet. Erst nach eingegangenem ACK wird der Verbindungsstatus auf „ASSURED“ gesetzt. Verbindungen die nicht als „ASSURED“ klassifiziert sind, werden, falls die Hash-Tabelle voll ist, zuerst entfernt.

Zuletzt wird, sofern ein *Helper* für diese Verbindung gefunden wurde, die entsprechende *help()* Funktion aufgerufen. (Vgl. Conntrack Helper / Nat Helper)

Verbindungen werden erst dann in die Hash-Tabelle eingetragen, wenn sie bestätigt wurden, d.h. Conntrack wartet mit dem Eintrag in die Hashtabelle, bis das Paket die Netfilter Hooks NF_IP_LOCAL_IN oder NF_IP_POST_ROUTING passiert hat. In diesem Fall kann man sicher sein, dass die Verbindung nicht durch eine Firewall Regel verworfen wurde.

Jede Verbindung wird zweimal in die Tabelle eingetragen, und zwar einmal für jede Richtung. Das heißt es werden zwei *tuple_hash* Strukturen, die jeweils auf den gleichen Conntrack verweisen, eingetragen.

struct ip_conntrack_tuple:

⁵dem inversen Tupel

- *src* - vom Typ *struct ip_conntrack_manip*. Repräsentiert den manipulierbaren Teil vom Tupel. Besteht aus den Feldern *ip*, *tcp.port / udp.port / icmp.id*.
- *dst* - Repräsentiert den nicht manipulierbaren Teil. Besteht aus den Feldern *ip*, *tcp.port / udp.port / icmp.type, icmp.code*.
- *protonum* - verwendete Protokoll

struct ip_conntrack:

- *ct_general* - Usage Counter
- *tupelhash[]* - Array von Tupelhashes. Eines für jede Richtung
- *status* - Status Flags
- *expected_list* - List aller *expectations*
- *timeout* - Timeout der Verbindung
- *sibling_list* - Listenkopf für weitere erwartete Verbindungen
- *expecting* - Anzahl der erwarteten Verbindungen
- *master* - *ip_conntrack_expect* falls *expectation*
- *helper* - *Helper* Struktur
- *infos[]* - Array vom Typ *nf_ct_info*. Ein Feld für jede mögliche Bewertung.
- *proto* - Protokoll
- *help* - private Daten für *Helper*
- *nat* - Daten für NAT; nur enthalten, falls NAT aktiviert ist.

Auch lässt sich das Connection Tracking flexibel erweitern durch:

- Application Helper Module (*ip_conntrack_ftp*, *ip_conntrack_irc*, ...)
- Protokoll Helper Module (PPTP, ...)

1.6.1 Stateful Firewall

Eine wichtige Anwendung des Connection Trackings ist die sogenannte „Stateful Firewall“. Diese Art von Firewalls betrachtet nicht mehr jedes Paket einzeln, sondern arbeitet Verbindungsorientiert. Hierzu nutzt sie die Ergebnisse die das Connection Tracking bereits ausgewertet hat. Dadurch kann sich die Performance der Firewall, gerade bei komplexen Regelsätzen, deutlich verbessern. Ein Beispiel einer solchen Firewall wäre ein Regelsatz, der neue Verbindungen von „innen“ nach „ausen“ erlaubt, aber nicht anders herum. Weiterhin werden alle Pakete, die einer schon bestehenden Verbindung zugeordnet werden können, ebenfalls durchgelassen.

```
iptables -A FORWARD -j ACCEPT -m state --state ESTABLISHED,RELATED
```

Diese iptables-Regel erlaubt die Weiterleitung aller Pakete schon bestehender Verbindungen.

1.6.2 Conntrack Helper

Conntrack Helper erweitern Conntracking um neue Fähigkeiten, Verbindungen zu klassifizieren. So können neue Verbindungen zu bestehenden zugeordnet werden (RELATED). Die Aufgabe eines *Conntrack Helper* ist es, diese Assoziationen zwischen Verbindungen herzustellen.

Conntrack Helper lassen sich durch eine Struktur beschreiben, *ip_conntrack_helper*, deren wichtigsten Merkmale ein Tupel mit Maske und ein Funktionszeiger sind. Das Tupel und Maske vom Typ *ip_conntrack_tuple* beschreiben die passende Verbindung. Die Maske spezifiziert dabei die gültigen Felder des Tupels. Der Funktionszeiger spezifiziert die Funktion, die für das auf Tupel/Maske passende Paket aufgerufen werden soll. Diese kann, neben Paketmanipulation, Erwartungen (*expectations*) registrieren.

Ein solche Erwartung wird ebenfalls durch Initialisierung einer Struktur erzeugt. In diesem Fall ist das die *ip_conntrack_expect* Struktur. Diese Struktur enthält alle Informationen um eine eintreffende erwartete Verbindung zu erkennen und zuzuordnen. Desweiteren ist noch ein Funktionszeiger enthalten, der für das erste Paket der erwarteten Verbindung aufgerufen wird.

ip_conntrack_helper:

- *list* - Listenzeiger
- *name* - Name des Helpers
- *flags* - reuse. falls *max_expectation* erreicht noch verwenden
- *me* - Zeiger auf sich (Modul)
- *max_expected* - max. Anzahl gleichzeitig erwarteter Verbindungen
- *timeout* - timeout für eine Erwartung
- *tupel* - *ip_conntrack_tuple* beschreibt die Pakete, die der Helper möchte
- *mask* - *ip_conntrack_tuple* markiert gültige Felder in *tupel*.
- *help* - Funktionszeiger. Wird für alle auf *tupel-mask* passenden Pakete aufgerufen

ip_conntrack_expect:

- *list* - Listenzeiger
- *use* - Referenz-Zähler

- *expected_list* - Liste aller Erwartungen des Masters⁶
- *expectant* - Master
- *sibling* - Zeiger auf die zu erwartende Verbindung, falls sie zustande kommt
- *ct_tupel* - saved for conntrack
- *timeout* - Timeout der Erwartung
- *tupel* - beschreibt die erwartete Verbindung
- *mask* - gültige Felder in *tupel*
- *expectfn* - Funktionszeiger; wird aufgerufen für das erste Paket der erwarteten Verbindung
- *seq* - Sequenznummer, bei der diese Erwartung erzeugt wurde
- *proto* - Protokoll
- *help* - Helper Struktur (zBsp *ip_ct_ftp_expect*)

Ein Beispiel eines Conntrack Helpers ist das Modul *ip_conntrack_ftp*. Zum Datentransfer nutzt das FTP Protokoll, neben der Kontroll-Verbindung auf Port 21, eine zusätzliche Verbindung zum Datentransfer. Diese Verbindung kann zwischen Client und Server ausgehandelt. Durch senden eines „PORT“ Befehls durch den Client, weiss der Server auf welchen Port der Client auf die Datenverbindung wartet.

Sieht der FTP-Conntrack-Helper ein Paket, das einen solchen Befehl enthält, so registriert dieser eine Erwartung auf diesem Port.

1.6.3 NAT im Kernel

Network Adress Translation (NAT) ist ein weiteres Beispiel der Nutzung der Netfilter Infrastruktur. NAT nutzt aber auch die Fähigkeiten des Connection Trackings. Das NAT Kernel-Modul nutzt die gleichen Netfilter Hooks wie das Connection Tracking, ist aber von der Priorität nachgeordnet, da es auf dessen Ergebnissen aufbaut.

Erreicht ein Paket den NAT-Code, so werden zunächst die Informationen des Connection Trackings ausgewertet. Die Hilfsfunktion *ip_conntrack_get()* liefert hierzu sowohl den Status des Pakets als auch die *ip_conntrack* Struktur.

In der *ip_conntrack* Struktur ist ein gleichnamiges Feld für NAT reserviert, welches auch einen Zeiger auf eine *ip_nat_info* Struktur enthält. Diese Struktur enthält Informationen über die evtl. notwendigen Adressmanipulationen für Pakete dieser Verbindung, als auch einen Eintrag für eine NAT-Helper Struktur. Handelt es sich bei dem ankommenden Paket um ein Paket, das mit der Bewertung „NEW,, vom Connection Tracking ausgezeichnet wurde, so muss für dieses in der

⁶Verbindung zu der diese Erwartung gehört

iptables NAT Tabelle nach einer Regel gesucht werden. Wird eine solche gefunden, wird sie in die *ip_nat_info* Struktur eingetragen.

Gehört das Paket zu einer bereits bestehenden Verbindung, so sind die Regeln der Adressmanipulation schon eingetragen und falls welche existieren, können diese angewendet werden. Dies geschieht für alle Pakete in der Funktion *do_bindings()*. Zuletzt wird noch, falls ein Helper eingetragen worden ist, die *help()* Funktion aufgerufen.

1.6.4 NAT Helper

Genügen einem über IP liegendem Netzwerkprotokoll die Möglichkeiten der Adressmanipulation nicht mehr, so kann man für dieses die Funktionalität von NAT durch *NAT Helper* erweitern. *NAT Helper* werden mit Hilfe der *ip_nat_helper* Struktur registriert. Diese beschreibt die zu untersuchenden Pakete mit einem Tupel, einer Maske und spezifiziert dazu noch zwei Funktionszeiger. Die *help* Funktion wird für alle auf das Tupel passende Pakete aufgerufen. Sie sammelt unter anderem Informationen für zukünftige „RELATED“ Verbindungen und kann auch das Paket verändern. Die andere Funktion (*expect*) wird für das erste Paket einer erwarteten Verbindung (*expectation*) aufgerufen. Hier können Adressmanipulationen vorgenommen werden.

Die *ip_nat_helper* Struktur sieht folgendermaßen aus:

- *list* - Listenzeiger
- *name* - Name des Helpers
- *flags* - Standalone oder nicht
- *me* - Zeiger auf sich (Modul)
- *tupel* - *ip_conntrack_tupel* beschreibt die Pakete die der Helper möchte
- *mask* - *ip_conntrack_tupel* markiert gültige Felder in *tupel*.
- *help* - Funktionszeiger. Wird für alle auf *tupel-mask* passenden Pakete aufgerufen
- *expect* - Funktionszeiger. Wird für Pakete einer *expected* Verbindung aufgerufen

Ein Beispiel dafür ist das FTP Protokoll[6]. So ist es notwendig den PORT Befehl eines FTP-Clients nachträglich zu verändern, weil dieser sich in einem Netz befindet, dessen Adressen verändert werden. In diesem Fall muss die Adresse in dem Paket durch die des Gateways getauscht werden. Dazu kann noch die Schwierigkeit kommen, dass sich die Paketgröße ändern kann und es zu einer Verschiebung der erwarteten Paketsequenznummer kommt, die für alle folgenden Pakete ausgeglichen werden muss.

Auch müssen alle als „RELATED“ markierten Pakete speziell behandelt werden. So müssen IP Adresse und Portnummer, dem PORT Befehl entsprechend, aller eingehenden FTP-Daten Pakete umgeschrieben werden.

Ein Codeskeleton für Helper Module und weitere Details zu den einzelnen Strukturen gibt es hier [7].

1.7 Pakete einreihen

Der Netfilter Code erlaubt es, sogenannte Queuehandler zu registrieren. Zur Zeit nutzt *ip_queue.c* diese Schnittstelle. Ist der Queuehandler mit *nf_register_queue_handler()* registriert, so werden alle Pakete, für die an einem Hook NF_QUEUE zurückgegeben worden ist, in die Warteschlange aufgenommen. Ist kein Queuehandler registriert, so sind NF_QUEUE und NF_DROP äquivalent. Während sich die Pakete in der Queue befinden, lassen sie sich beliebig manipulieren. Mit *nf_reinject()* können sie an das System zurückgegeben werden. Mit dem Paket übergibt man weiterhin ein Urteil zurück, welches das weitere Schicksal des Paketes bestimmt. Hier sind wieder alle fünf Rückgabewerte möglich.

Das Einreihen von Paketen zusammen mit dem Queuehandler *ip_queue* erlauben es, fast alles, was zur Zeit im Kernel an Filtern möglich ist, auch im sogenannten „userspace“ zu realisieren. *Ip_queue* bietet dem Programmierer einen Unix Socket, auf dem er die Pakete bearbeiten kann.

1.8 Ausgewählte iptables Module

1.8.1 Das *owner* Modul

Das Owner-Modul bietet die Möglichkeit, Regeln für lokal generierte Pakete auf *matches* mit Unix GID, UID, SID, PID zu generieren. Dadurch hat man zum Beispiel die Möglichkeit, einzelnen User oder Usergruppen den Zugang zum Netz teilweise zu sperren oder ganz zu verwehren. Auch der vom User verursachte Netzwerkverkehr ist so zählbar.

Dabei arbeitet dieses Modul nicht auf den Paketdaten selbst, sondern auf den Metadaten des Pakets. In diesem Fall *skb->sk->socket->file*. Die *file* Struktur enthält alle benötigten Informationen über Benutzer und Prozess, die zu diesem Socket gehört.

Dieser *match* funktioniert nur in der OUTPUT *chain*, da im IP Layer nur lokal generierte ausgehende Pakete einen Socket zugeordnet haben. Durch eine Modifikation des Moduls ist es möglich, auch eingehende Pakete einem Socket zuzuordnen. Diese Zuordnung passiert normalerweise erst in der nächsthöheren Netzwerkschicht (z.B. bei TCP `__tcp_v4_lookup()`). Zieht man diese Operation vor, so hat man ebenfalls eine *sk*-Struktur zur Verfügung.

1.8.2 Das *length* Modul

Das *length* Modul ist eines der einfachsten *iptables*-Module. Es wird lediglich der Parameter der Regel mit dem Header des Paketes verglichen. In diesem Modul werden, im Gegensatz zum *owner*-Modul, echte Paketdaten verglichen.

1.8.3 Das DSCP Modul

Es gibt zwei DSCP Module, ein *match* und ein *target*.

Differentiated Services Code Point[8] (DSCP) ist der Nachfolger von TOS (Type of Service). Dabei können die ersten sechs Bit, des ehemaligen TOS Felds im IP Header, zur Klassifizierung der Netzwerkpakete genutzt werden.

Ein Beispiel eines Anwendungsszenarios ist eine volle Internetleitung. Ohne spezielle Eingriffe kommt es zum Beispiel bei interaktiven Sessions (ssh) zu größeren Verzögerungen bei der Übertragung der Tastatureingaben, da alle Datenpakete gleichbehandelt werden.

Durch Einteilung des Netzwerkverkehrs in Klassen wie „interaktiv“, „downloads“ oder „ACK“ kann eine volle Leitung wieder benutzbar gemacht werden, indem auf dem Router oder Internetgateway bestimmte Klassen vorrangig behandelt werden.

Das DSCP Modul, welches das *target* enthält ist für das Setzen der Klassifizierung zuständig, das das den *match* enthält und zum Finden von klassifizierten Paketen. Das letztere ist nur innerhalb der *mangle* Tabelle nutzbar.

2 Danksagung

Dank geht an Sandra Zipfel für zigfaches Korrekturlesen.

Ebenfalls vielen Dank an Patrick „Kaber“ McHardy für seine Hilfe und fachlichen Rat.

Literaturverzeichnis

- [1] DFN/CERT *<http://www.cert.dfn.de/team/ue/fw/workshop/node2.html>*
- [2] Netfilter *<http://www.netfilter.org>*
- [3] FWTK *<http://www.fwtk.org/main.html>*
- [4] Linux Kernel *<http://www.kernel.org>*
- [5] Paul „Rusty“ Russel . . .
- [6] FTP Protokoll RFC 959 *<http://www.ietf.org/rfc/rfc0959.txt>*
- [7] Harald Welte: Netfilter connection tracking and nat helper modules
<http://gnumonks.org/ftp/pub/doc/conntrack+nat.html>
- [8] DSCP *<http://www.ietf.org/rfc/rfc2474.txt>*