

# Netfilter / IP-Tables

Sicherheitstage SS 2007

Hergen Harnisch

`harnisch@rrzn.uni-hannover.de`

20.06.2007 (mit Nachträgen 14.01.2010)



## Firewall:

- Paket-Filter
- stateful inspection
- Policy

## IP-Tables:

- Grundsätzliches
- Aufbau
- Regeln
- Handling

## Einbindung

- Suse
- Debian/Ubuntu

## Tool: FWBuilder

## Advanced:

- SSH-Bruteforce
- Portscan-Abwehr
- Benutzerdefinierte Chains

- Abblocken ungewollten Datenverkehrs

- möglichst frühzeitig:

- vor Erreichen des Rechners:  
Hardware-Firewall, vgl. Netzschutz
- im Netzwerk-Code des Kernels:  
Personal-Firewall

← z.B. mit IP-Tables

- teilweise mehr:

- Verbindungs-Logging
- Analyse Datenverkehr
- Datenweiterleitung

- auf verschiedenen Ebenen:

- nur Header-Daten:  
Verbindungsebene (OSI-Layer 2–4)
- inhaltlich:  
Application-Level (OSI-Layer 7)

← Paket-Filter

## Header-Informationen

## Ethernet

Identifikation durch Hardware-Adressen (MAC)

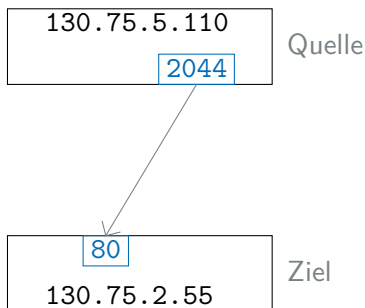
## TCP/IP

IP-Pakete haben:

- Quell-IP-Adresse
- Ziel-IP-Adresse

TCP & UDP zudem:

- Quell-Port
- Ziel-Port



## Protokolle

### TCP — Transmission Control Protocol

- verbindungsorientiert: Verbindungsauf- und abbau
- Bsp.: http, ssh

### UDP — User Datagram Protocol

- verbindungslos  
trotzdem evt. Zugehörigkeit, z.B. Antwort auf Anfrage
- Bsp.: dns, syslog

### ICMP — Internet Control Message Protocol

- verbindungslos, nicht für Daten
- Bsp.: ping (echo-request)

## stateless

- jedes Paket wird einzeln betrachtet; ohne Vorwissen, Folgerung
- mögliche Antworten sind zu beachten
- schnell, z.B. in Routern, „veraltet“

## stateful

- Pakete werden im Zusammenhang gesehen; Folgepakete werden wie 1. Paket behandelt, zugehörige Verbindungen (z.B. ftp) auch
- Regeln nur für Verbindungsaufbau anzugeben
- z.B. bei Netzschutz-FW, bei Personal-FWs, heute üblich

IP-Tables wird stateful durch spez. Regeln

```
-m state --state RELATED,ESTABLISHED -j ACCEPT
```

## Grundsatz

- gut bewusst Gewolltes erlauben, Rest verbieten  
theoretische „Diensteinventur“
- q&d Schlechtes verbieten, Rest zulassen  
nachträgliches Aufsetzen, Analyse Datenverkehr

## Erlaubtes

- Außen nach Innen (auf Rechner):
  - Wartungszugang, z.B. ssh aus Institutsnetz
  - für Server: je nach Aufgabe, z.B. http & https
- Innen nach Außen (von Rechner):
  - dns an bestimmte DNS-Server
  - http/ftp für System-/Virensignatur-Updates
  - smtp für System-Meldungen
  - *generell*: diese Richtung eher unkritisch

## Netfilter / IP-Tables

- im Linux-Kernel 2.4 & 2.6 der Standard, ersetzt altes `ipchains` bzw. `ipfwadm`
- im Kernel implementiert, z.T. Kernel-Module

## Vortrag

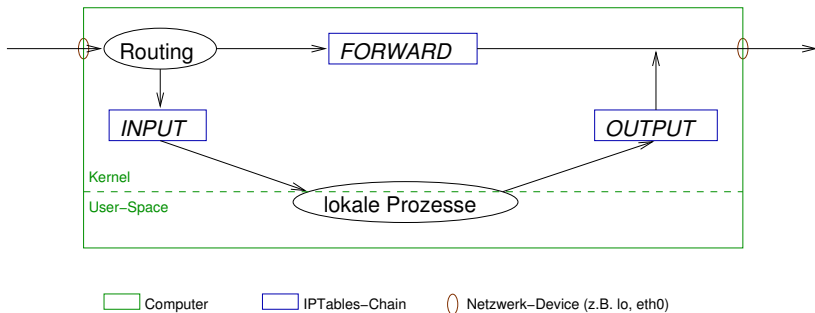
**Ziel** Einsatz als Personal-Firewall

**nicht** Natting/Masquerading, Port-Forwarding, Routing/Forwarding, Accounting, Logging

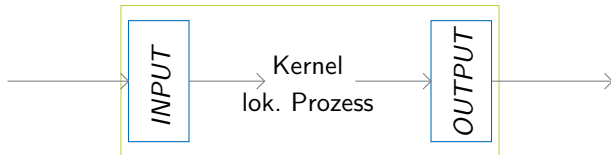


## Schema

## Aufbau IPTables



- kein Routing: FORWARD uninteressant
- jedes reinkommende Paket geht durch INPUT
- jedes rausgehende Paket geht durch OUTPUT
- nach Eintrag der „Stateful-Regel“ in INPUT & OUTPUT:
  - nach Innen / zum Rechner in INPUT
  - nach Außen / vom Rechner in OUTPUT



## Flankierendes

Über proc-Dateisystem (de-) aktivieren:

### Forwarding

- `/proc/sys/net/ipv6/conf/all/forwarding`
- `/proc/sys/net/ipv4/ip_forward`

### IP-Spoofing

- `/proc/sys/net/ipv4/conf/*/rp_filter`

→ beides bei Debian die Standard-Einstellung,  
meist mittels `echo 0|1>...` in `/etc/init.d/networking`

*Nachtrag 2011:* jetzt über `sysctl`, vgl. `/etc/sysctl.conf`

## Grund-Setup

### Standard-Policy

```
iptables -P INPUT DROP
# erstmal raus alles, spaeter ggf. restriktiver
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
# ggf. alle chains komplett leeren
iptables -F
```

### Stateful

```
iptables -A INPUT -m state \  
    --state RELATED,ESTABLISHED -j ACCEPT
```

→ bereits gute Absicherung eines Clients, aber noch zulassen:

```
iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT1
```

---

<sup>1</sup>Nachtrag 2011: lieber nur `iptables -A INPUT -i lo -j ACCEPT`

## Prinzip

## Einzelregel

Jede Regel besteht aus einer *Bedingung* und einem *Ziel*:

<code>-s 127.0.0.1 -i lo</code>	<code>-j ACCEPT</code>
<u>wenn lokale IP vom Loopback-Device</u>	
<code>-m tcp -p tcp --dport 22</code>	<code>-j ACCEPT</code>
<u>wenn TCP auf den SSH-Port</u>	<u>dann akzeptieren</u>

Das Ziel bestimmt das weitere Schicksal eines passenden Paketes.

## Prinzip

### Abfolge

Jede *chain* besteht aus einer Abfolge von Regeln, Reihenfolge ist entscheidend: Abarbeitung bis *first match*, d.h. nach einem „Match“ wird Regelverarbeitung abgebrochen

### Beispiel

```
1 -p tcp -d 130.75.2.0/24 --dport 25 -j ACCEPT
2 -p tcp --dport 25 -j DROP
3 -j ACCEPT
```

akzeptiert Mail an Mailserver des RRZN,  
Zustellung an andere Mailserver nicht erlaubt;  
restlicher Verkehr wieder erlaubt  
→ fast alles erlaubt, aber direktes Spamming unterdrückt

## Abfolge: Umsetzung

- Regeln können ans Listenende angefügt werden:  
`iptables -A chain ...`
- Regeln können vor bestimmten Eintrag eingefügt werden:  
`iptables -I chain rulenum ...`
- Bestimmte Regeln können gelöscht werden:  
`iptables -D chain rulenum`
- Bestimmte Regeln können ersetzt werden:  
`iptables -R chain rulenum ...`

Anzeige mit Zeilennummern:

```
iptables --line-numbers -L [chain]
```

## Ziele

### ACCEPT

Das Paket wird durchgelassen.

### DROP

Das Paket wird ohne Benachrichtigung/Rückmeldung verworfen.

### REJECT

Das Paket wird verworfen, aber der Absender wird benachrichtigt:

- standardmäßig per ICMP `port-unreachable`
- alternativ auch mit Beendigung einer TCP-Verbindung, sinnvoll für Ident-Port (Vermeidung von Timeouts):

```
-p tcp -dport 113 -j REJECT --reject-with tcp-reset
```

REJECT kann nicht Chain-Policy sein und benötigt Zusatzmodul.



## Ziele: Drop (vs. Reject)

Bei **DROP** erhält Anfragender keine Rückmeldung, ähnlich wie Unterdrückung der Pingbarkeit einzustufen.<sup>2</sup>

- IP-Adresse scheint (derzeit) ungenutzt
  - erschwert das Abscannen des Rechners:  
Timeouts sind abzuwarten
  - Anfragender probiert wie bei Paketverlust nochmal, z.B. P2P bei Dial-Up-Zugängen ebbt nur langsam ab, wenn man selbst kein P2P macht aber IP von einem P2P-Teilnehmer erbt
  - Angreifer lassen sich kaum noch abschrecken, häufiger wird sowieso gezielt ein Port abgescannt
  - man selbst muss auch auf Timeouts warten
- *Zumindest ping im LAN / LUH-Netz zulassen*

---

<sup>2</sup>Wird häufig als *stealth*-Funktionalität beworben.

## Matches

## IP-Adresse

**Quelle:** `-s [!] address[/mask]` Untersuchung der Absender-IP-Adresse (! negiert)

z.B. nicht aus LUH-Netz: `-s ! 130.75.0.0/255.255.0.0`

**Ziel:** `-d [!] address[/mask]`

## TCP/UDP-Ports

**Quelle:** `-p tcp|udp --sport [!] port`

**Ziel:** `-p tcp|udp --dport [!] port`

z.B. alle TCP-Verbindungen außer ssh: `-p tcp --dport ! 22`

## Matches

### ICMP

**Typ:** `-p icmp --icmp-type [!] type`

z.B. ping-Anfrage: `-p icmp --icmp-type echo-request`

### state-Modul

**Status:** `-m state --state state`

wobei *state* eine Kombination aus

- **ESTABLISHED** Teil einer existierenden Verbindung
- **RELATED** zwar neue Verbindung, aber zu existierender gehörend (z.B. FTP-Data zu FTP-Control)
- **NEW** gehört zu nichts Bekanntem
- **INVALID** ungültig (oder auch Speicher-Problem des Moduls)

## Connection-Tracking

Status RELATED hängt sehr vom Protokoll ab, daher sind protokoll-spezifische Erweiterungen nötig.

- `ip_conntrack_ftp`:  
FTP-Datenport (active & passive) fällt unter RELATED
- `ip_conntrack_netbios_ns`:  
damit wird das „Browsen“ nach Samba-Shares möglich

*meist nötig*: Module mit `modprobe` explizit laden!

## Beispielregeln

```
01 iptables -P INPUT DROP
02 iptables -P OUTPUT ACCEPT
03 iptables -P FORWARD DROP; iptables -F
04 iptables -A INPUT -m state --state RELATED,ESTABLISHED
\
                                                                    -j ACCEPT

# rein: lokal, ping, ssh aus LUH
05 iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT
06 iptables -A INPUT -p icmp -icmp-type 8 -j ACCEPT
07 iptables -A INPUT -s 130.75.0.0/255.255.0.0 -p tcp \
                                                                    --dport 22 -j ACCEPT

# raus: alles, aber Mail/DNS nur LUH
08 iptables -A OUTPUT -d 127.0.0.1 -o lo -j ACCEPT
09 iptables -A OUTPUT -d ! 130.75.0.0/16 -p tcp \
                                                                    --dport 25,465 -j DROP
10 iptables -A OUTPUT -d 130.75.1.32/32 -j ACCEPT
11 iptables -A OUTPUT -d 130.75.1.40/32 -j ACCEPT
12 iptables -A OUTPUT --dport 53 -j DROP
```

## „Design-Kriterien“

- DROP als Standard-Policy
- lieber REJECT als DROP:  
leichter zur Fehlersuche allgemein, DROP nur gegen DoS gut
- komplizierte Regelpassagen in User-Chains abtrennen (s.u.)
- Übersichtlichkeit, Verständlichkeit, Wartbarkeit  
wichtiger als 100%-iger Schutz mit Raffinessen
- Aufbewahrung eines *kommentierten* Erstellungsskriptes
  
- *ersetzt nie die normale Dienste-Absicherung*

## Speicherung von Regelsätzen

### Befehle

- Abspeichern mit `iptables-save >Datei`
- Wiederherstellen mit `iptables-restore <Datei`

### Beispiel-Datei

```
# Generated by iptables-save v1.3.3 on Tue Jun 6 19:21:25 2006
*filter
:INPUT DROP [1:60]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [72:6224]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -s 127.0.0.1 -i lo -j ACCEPT
COMMIT
# Completed on Tue Jun 6 19:21:25 2006
```

## Remote-Bearbeitung

**Problem:** falsche Regeln führen zur Aussperrung

**Lösung:** automatische Wiederherstellung / Löschung

## Skript

```
# ggf. Abspeicherung
# iptables-save >/root/iptables-fallback.conf
# neue Regeln:
...
# Timeout und Wiederherstellung:
if read -p "Annahme neuer Regeln mit Return" -t 30;
  then echo neue Regeln gelten;
  else
    iptables-restore </root/iptables-fallback.conf;
    echo Regeln zurueckgesetzt;
fi
```



## Init-Prozess

### Init-Skript

Init-Skript `/etc/init.d/iptables` erstellen. Parameter:

**start** aktiviert die Firewall

**stop** deaktiviert die Firewall

Quelle: [http://www.rrzn.uni-hannover.de/fw\\_linux.html](http://www.rrzn.uni-hannover.de/fw_linux.html)

Meist wird mit `iptables-save` erzeugte Datei  
`/etc/iptables.conf` eingelesen.

### Aktivierung

Setzen von Links in `/etc/rc?.d`,  
unbedingt distributionsspezifische Tools verwenden (s.u.)

## Module

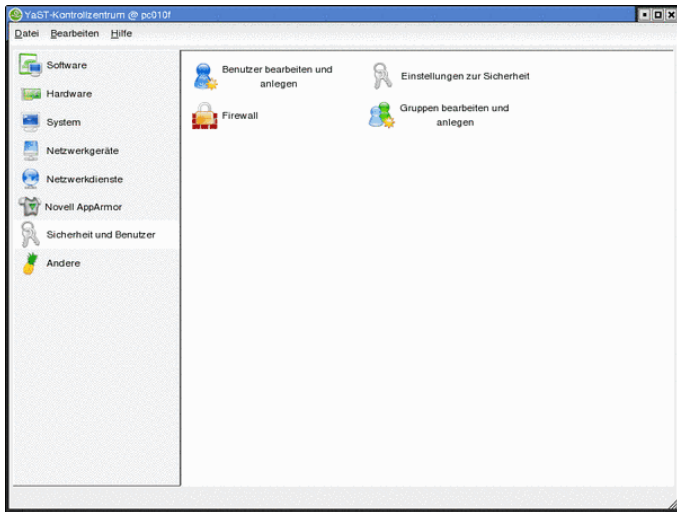
Einige IP-Tables-Erweiterungen liegen in Modulen, z.B.

- `ipt_state` (für `established`, `related`)
- `ip_conntrack_ftp` (für Datenport-Freischaltung)
- `ipt_REJECT` (für `Reject` statt `Drop`)

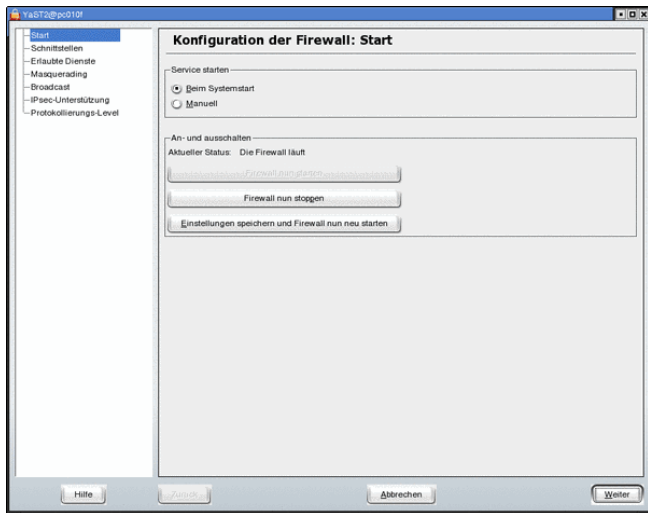
Diese Module sind ggf. beim Systemstart zu laden, durch

- Auflisten in `/etc/modules`
- `modprobe`-Aufruf im Init-Skript
- Abhängigkeiten von Geräten in `/etc/modules.conf` meist nicht möglich

## Yast: Aufruf Firewall



## Yast: Unterpunkt Firewall



## Yast

### Vorgehen

- SuseFirewall2 nach der Installation aktiv
- dann über Yast Ports öffnen („erlaubte Dienste“)

### Eigenschaften Yast/SuseFirewall

- Zonen-Konzept
  - nur 3 Zonen: intern, extern, DMZ
  - geht konzeptionell von sichererem LAN aus
  - Öffnen nur für gewisse Clients nur begrenzt möglich
- ermöglicht NAT, DMZ, Port-Redirect, VPN, dadurch nicht unbedingt einfacher als Skript

## Init-Skript

Zur Verwendung des oben genannten Init-Skripts:

1. in Yast die FW-Konfiguration & Start auf „manuell“ schalten, vgl. [http://www.rrzn.uni-hannover.de/fw\\_suse.html](http://www.rrzn.uni-hannover.de/fw_suse.html)

### ab Suse 9 beachten

2. INIT INFO-Block ins Skript einfügen (vgl. `man insserv`)

### Suse 8-10

3. statt Links manuell zu setzen:
  - Yast Control Center → System → Run level editor & Runlevel properties
  - oder `chkconfig iptables on` aufrufen

## Init-Skript

- altes Init-Skript bei Debian-Sarge (oldstable) noch gepackt in `/usr/share/doc/iptables/example/oldinitdscript.gz`, (oder [http://www.rrzn.uni-hannover.de/fw\\_debian.html](http://www.rrzn.uni-hannover.de/fw_debian.html)) verwendet zwei gespeicherte Regelsätze:
  - `start` aktiviert `/var/lib/iptables/active`, erzeugen durch Setzen der Regeln und Abspeichern mit `/etc/init.d/iptables save active`
  - `stop` aktiviert `/var/lib/iptables/deactive`, abspeichern mit `/etc/init.d/iptables save inactive`
- *alternativ*: generisches Init-Skript von [http://www.rrzn.uni-hannover.de/fw\\_linux.html](http://www.rrzn.uni-hannover.de/fw_linux.html)
- Links setzen mit: `update-rc.d iptables defaults`

## ifupdown

- ifup/ifdown (de)-aktivieren Netzwerkschnittstellen und zugehörige Konfigurationen, Einstellungen dazu in `/etc/network/interfaces`
- z.B. Flankierendes (Spoof-Protection, Forwarding) auch in `/etc/network/options`
- z.B. über Skripte in `/etc/network/if-[pre-up|up|down|post-down].d/`
- Vorteil gegenüber Init-Skript:
  - Umkonfigurationen werden berücksichtigt
  - Beachtung von DHCP, Namen von Interfaces leichter
  - dynamische Änderungen wie z.B. WLAN abdeckbar
- Beispiel-Skript & Erklärung unter [http://www.rrzn.uni-hannover.de/fw\\_debian.html](http://www.rrzn.uni-hannover.de/fw_debian.html)



- GUI zur Regeleingabe & -bearbeitung, viel Drag&Drop
- System-Einstellungen (z.B. Forward, Spoofing-Protection)
- Standard-Setups zur Auswahl
- benannte Objekte, Kommentierung
- interne Datenbank der Einstellungen (XML-Dateien)
- Export, u.a. als IPTables-Skript (policy-compiler)
- zentrale Verwaltung mehrere Firewalls, Regelverteilung auf die Systeme
- unterstützt Bridging, Routing, NAT
- FWBuilder selbst Cross-Plattform (insb. Linux, Windows)
- Quelle: <http://www.fwbuilder.org/>

## Tool: FWBuilder

## Screenshot

Firewall Builder: test.fwb

File Edit Object Rules Help

User

Firewalls: test2

Policy	outside	inside	loopback	dmz	NAT			
	Source	Destination				Service	Action	Time
0	net-192.168.1.0	test2				ssh	Accept	
1	test2	internal server				DNS	Accept	
2	Any	test2				Any	Deny	
3	Any	Any				auth	Reject	
4	Any	server on dmz				smtp	Accept	
5	server on dmz	internal server				smtp	Accept	
6	server on dmz	net-192.168.1.0				DNS	Accept	
7	net-192.168.2.0	net-192.168.1.0				Any	Deny	
8	net-192.168.1.0	Any				Any	Accept	
9	Any	Any				Any	Deny	

Object Type: Firewall  
 Object Name: test2  
 Platform: iptables  
 Version:  
 Host OS: linux24

This firewall has three interfaces: eth0 faces outside and has a static routable address; eth1 faces inside; eth2 is connected to DMZ subnet. Policy includes basic rules to permit unrestricted outbound access and anti-spoofing rules. Access to the firewall is restricted only from internal

Firewall

General Templates SNMP

Name: test2

Library: User

Platform: iptables

Version: - any -

Firewall Settings ...

Host OS: Linux 2.4/2.6

Host OS Settings ...

Comment:

This firewall has three interfaces: eth0 faces outside and has a static routable address; eth1 faces inside; eth2 is connected to DMZ subnet. Policy includes basic rules to permit unrestricted outbound

Apply Changes

Tool: FWBuilder

Demo Host: GUI

Firewall Builder: sita.fwb

File Edit Object Rules Help

Standard

pc0815

Firewalls: pc0815

Policy outside loopback NAT

	Source	Destination	Service	Action	Time
0	Any	pc0815	ssh Useful_ICMP ping request	Accept	Any
1	Any	RRZN-Server	smtp smtps	Accept	Any
2	Any	Any	smtp smtps	Accept	Any
3	pc0815	Any	Any	Deny	Any
4	Any	Any	Any	Reject	Any

Object Type: Network  
Object Name: RRZN-Server  
130.75.0.0/255.255.248

rrwhois  
securidprop  
SMB  
smtp  
smtps  
socks  
sqlnet1  
squid  
ssh

Accept  
Deny  
Reject  
Accounting

## Demo Host: Ergebnis

## aus Datei pc0815.fw

```
# Rule 1 (global)
#
echo "Rule 1 (global)"
#
#
#
$IPTABLES -N Cid466E5765.0
$IPTABLES -A OUTPUT -d ! 130.75.0.0/21 -j Cid466E5765.0
$IPTABLES -N RULE_1
$IPTABLES -A Cid466E5765.0 -p tcp -m tcp -m multiport --dports 25,465 -j RULE_1
$IPTABLES -N Cid466E5765.1
$IPTABLES -A INPUT -d ! 130.75.0.0/21 -j Cid466E5765.1
$IPTABLES -A Cid466E5765.1 -p tcp -m tcp -m multiport --dports 25,465 -j RULE_1
$IPTABLES -N Cid466E5765.2
$IPTABLES -A FORWARD -d ! 130.75.0.0/21 -j Cid466E5765.2
$IPTABLES -A Cid466E5765.2 -p tcp -m tcp -m multiport --dports 25,465 -j RULE_1
$IPTABLES -A RULE_1 -j LOG --log-level info --log-prefix "RULE 1 -- REJECT "
$IPTABLES -A RULE_1 -j REJECT

#
```

## recent-Modul

- Modul laden mit `modprobe ipt_recent`
- ermöglicht eine adaptive Firewall:
  - Quell-IP kann in Liste protokolliert werden
  - mit Liste können bisherige Zugriffe in Bedingungen geprüft werden, dabei auch die Anzahl und der Zeitraum prüfbar
- Listen:
  - mehrere Listen möglich
  - standardmäßig bis zu 100 Einträge
  - angelegte Listen in `/proc/net/iptables/`
  - Liste manipulierbar durch Schreiben auf `procfs`-Datei:  
„xx.xx.xx.xx“ fügt IP zu, „-xx.xx.xx.xx“ löscht IP, „clear“ leert

```
1 modprobe ipt_recent
2 # state NEW nicht noetig, da stateful-Regel davor
3 iptables -A INPUT -p tcp --dport 22 \
4   -m recent --update --seconds 300 --hitcount 2
5                                     --name SSH \
6   -j DROP
7 iptables -A INPUT -p tcp --dport 22 \
8   -m recent --set --name SSH \
9   -j ACCEPT
```

in 4: SSH-Verbindungsaufbauten werden analysiert:

- a) Kombination `hitcount&seconds`: wenn Zähler für die letzten 5 min  $\geq 2$ , dann die Verbindung „dropen“
- b) `update`: unter Bed. b) Zugriff in Liste SSH protokollieren

in 7: wenn Limit noch nicht erreicht (d.h. 4 nicht erfüllt):

- c) wenn neue IP: IP wird der Liste SSH zugefügt (Zähler 0)
- d) wenn IP bereits in Liste SSH: Zugriff protokollieren & Zähler+1

erlaubt 2 neue SSH-Verbindungen je 5 min, egal ob erfolgreich oder nicht

```
1 iptables -A INPUT -m recent --update \  
2   --seconds 300 --hitcount 4 --name BANME -j DROP  
3 iptables -A INPUT -m tcp -p tcp ! --syn -j DROP  
4 iptables -A INPUT -m tcp -p tcp --dport 22 -j ACCEPT  
5 iptables -A INPUT -m recent --set --name BANME -j REJECT
```

- Prinzip: häufig genug auf verbotene Ports, dann wohl ein Scan  
genauer: 4 verbotene Zugriffe in 5 min gelten als Scan
- nach 3 bleiben von TCP nur Verbindungsaufbauten
- 4 ist die letzte Stelle für Unverdächtiges (Bsp. ssh)
- in 5 Ankommendes ist verdächtig, also notieren & zählen
- 1,2: ggf. verbieten (s.o.) und auch weiterzählen,  
dann definiert auch 4 keine Ausnahme mehr

- obiges Abwehr-Verfahren sehr brutal  
Problem: Zugriffsversuch mit Retry könnte schon als Scan gelten  
Einsatzbereich: Workstations mit SSH-Zugang, nicht für Server
- allgemein ist Port-Scan-Abwehr schwierig, da mehrere Scan-Verfahren möglich (vgl. `man nmap`)
- Ausführliche Konfiguration, die auch andere Scan-Typen abdeckt, im `rc.firewall`-Skript (URL s.u., unterhalb von „#portscan detector“).
- extra Modul `psd` für Port-Scan-Detection war verfügbar, aber nicht mehr supported

## Empfehlung

Da Port-Scans eher unkritisch sind, einfach hinnehmen. Port-Scans (meint auch: ein Port auf mehreren IPs) sind nur interessant, da häufig der Urheber infiziert ist.



Man kann selbst Chains erzeugen, die als Ziel dienen können:

- Übersichtlichkeit durch Kapselung von Regeln
- Wiederverwendung von Regelsätzen (mehrfach als Sprungziel)
- statt Transaktionen: Regelsatz als Ganzes zufügen, gut auch für Tests

## Beispiel

```
# Chain erzeugen und Regeln einfügen
1 iptables -N SSHIN
2 iptables -A SSHIN -m recent --update --seconds 300 \
    --hitcount 2 --name SSH -j DROP
3 iptables -A SSHIN -m recent --set --name SSH -j ACCEPT
# aus user-def. Chains wird normalerweise zurueckgekehrt
# daher hier explizit Policy: restlichen Verkehr verwerfen
4 iptables -A SSHIN -j DROP
# SSH-Verkehr in SSHIN-Chain umlenken
5 iptables -A INPUT -p tcp --dport 22 -j SSHIN
```

# Schlussbemerkungen

## IP-Tables

- Konfiguration ohne Spezialitäten wie NAT ist überschaubar
- Möglichkeiten mächtig, dann aber kompliziert
- Spezialitäten evt. in extra Chains abtrennbar
- dokumentiertes Erstellungsskript aufbewahren

→ lieber eine überschaubare Grundabsicherung

## Sonstiges

Trotz IP-Tables-Sicherung immer Applikation traditionell sichern:

- applikationsspezifisch (z.B. `httpd.conf`, `sshd_config`)
- TCP-Wrapper (`hosts.allow`, `hosts.deny`)

→ Firewall ist Zusatz, nicht Ersatz

## Literatur

- Linux 2.4 Packet Filtering HOWTO  
<http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html>  
knapp, aber auch NAT
- IPTables-Tutorial  
<http://iptables-tutorial.frozentux.net/>  
umfassend, auch TCP/IP-Grundzüge, Zusatzmodule etc.
- RRZN-Webseite zu IP-Tables  
[http://www.rrzn.uni-hannover.de/fw\\_linux.html](http://www.rrzn.uni-hannover.de/fw_linux.html)  
bezogen auf Host-Absicherung (nicht NAT), Skripte zum Download