Netfilter / IP-Tables

Sicherheitstage SS 2006

Hergen Harnisch

harnisch@rrzn.uni-hannover.de

15.06.2006

- Firewall:
  - Paket-Filter
  - stateful inspection
  - Policy
- IP-Tables:
  - Grundsätzliches
  - Aufbau
  - Regeln
  - Handling
- Einbindung
  - Suse
  - Debian
- Advanced:
  - SSH-Bruteforce
  - Portscan-Abwehr
  - Benutzerdefinierte Chains
- Tools

Firewall:

- 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) ⟵ Paket-Filter
  - inhaltlich:
    Application-Level (OSI-Layer 7)

R|R|Z|N

Header-Informationen

Ethernet

Identifikation durch Hardware-Adressen (MAC)

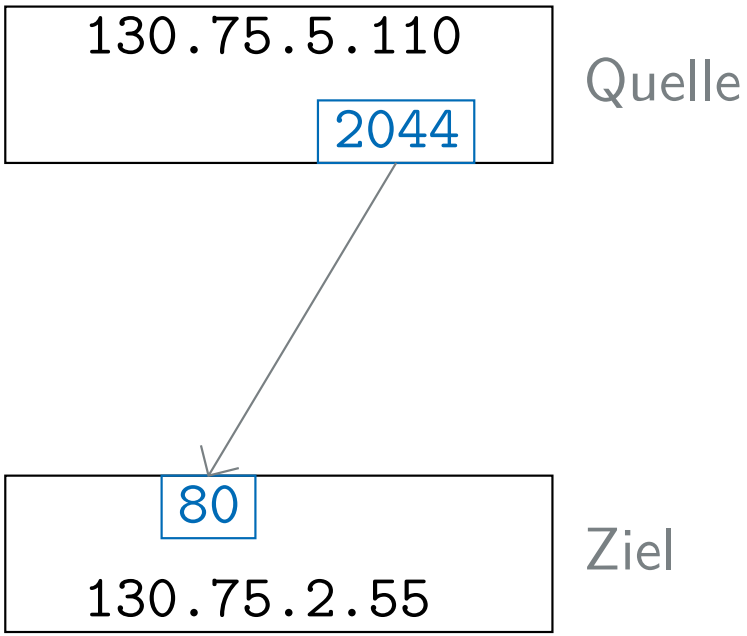
TCP/IP

IP-Pakete haben:

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

TCP & UDP zudem:

- Quell-Port
- Ziel-Port



R|R|Z|N

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
```

R|R|Z|N

Grundsatz

gut bewusst Gewolltes erlauben, Rest verbieten
theoretische „Diensteinventur“

q&d Schlechtes verbieten, Rest zulassen
nachträgliches Aufsetzen, Analyse Datenverkehr

Erlaubtes

- Außen nach Innnen (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

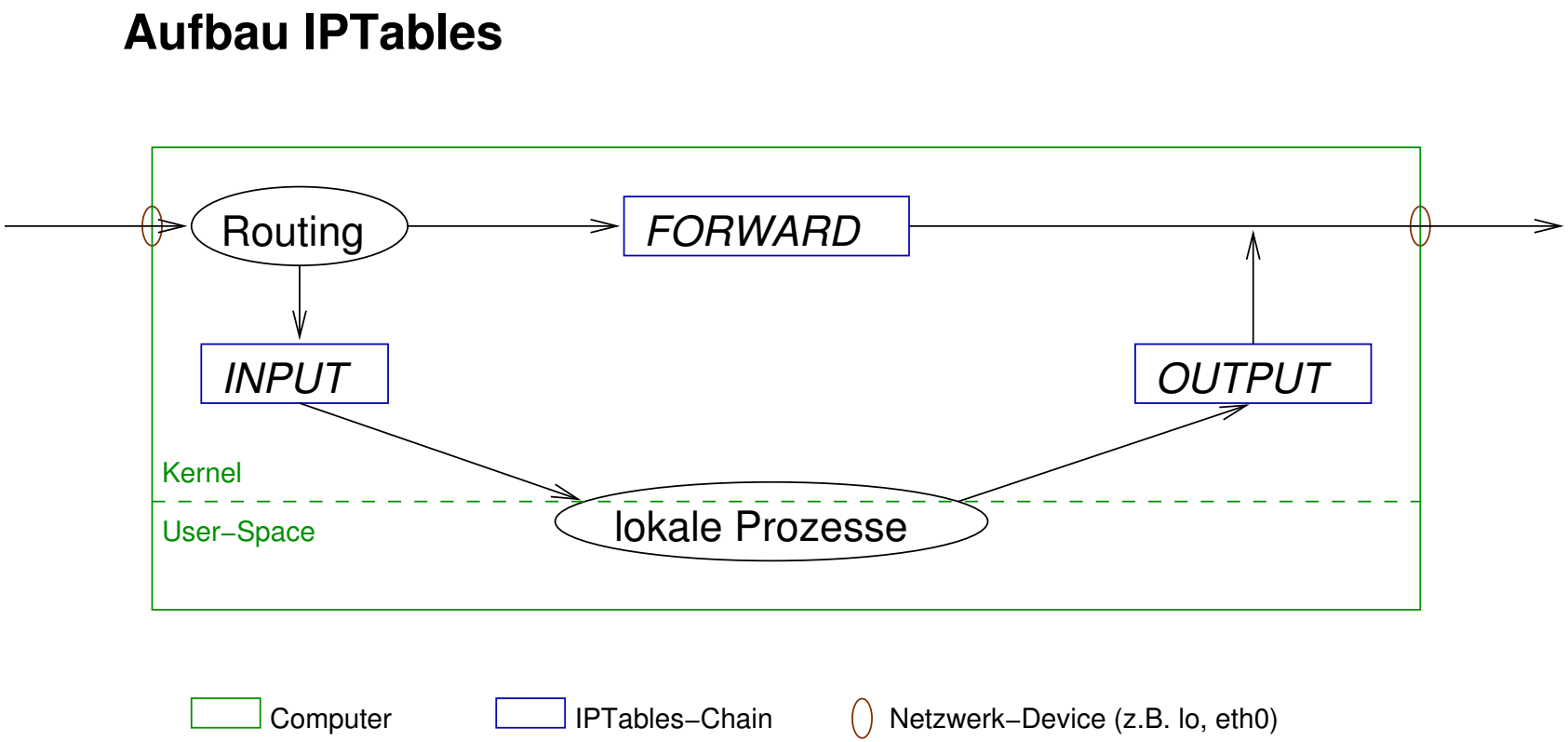
Vortrag

Ziel Einsatz als Personal-Firewall

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

R|R|Z|N

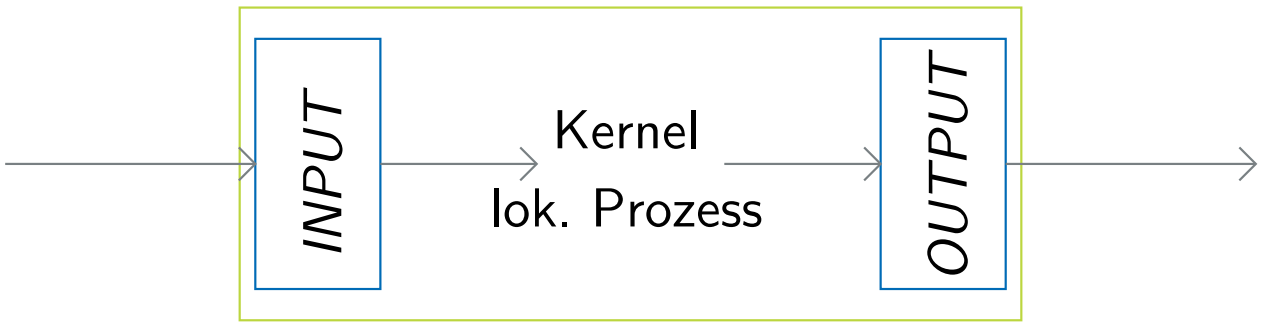
Schema


Aufbau IPTables
Routing
FORWARD
INPUT
OUTPUT
Kernel
User–Space
lokale Prozesse
Computer
IPTables–Chain
Netzwerk–Device (z.B. lo, eth0)

R|R|Z|N

für Server / Clients

- kein Routing: `FORWARD` uninteressant
- jedes reinkommende Paket geht durch `INPUT`
- jedes rausgehende Paket geht durch `OUTPUT`
- nach Eintrag der „Stateful-Regel“ in `INPUT` & `OUTPUT`: nur noch „Verbindungs“-Aufbauten
  - 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`

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

R|R|Z|N

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
```

$\longrightarrow$ bereits gute Absicherung eines Clients, aber noch zulassen:

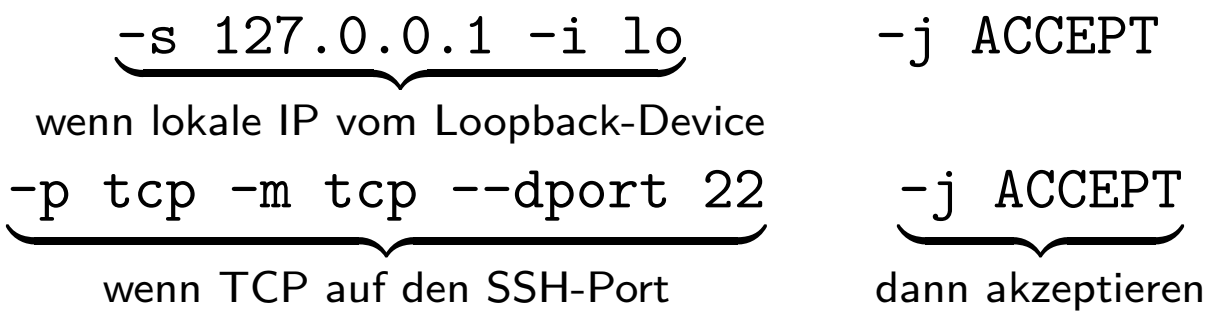
```
iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT
```

R|R|Z|N

Prinzip

Einzelregel

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



Das Ziel bestimmt das weiter 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
$\longrightarrow$ 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` [*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.

R|R|Z|N

Matches

IP-Adresse

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

z.B. nicht aus UH-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`

R|R|Z|N

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)

R|R|Z|N

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 UH
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 UH
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
```

R|R|Z|N

„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
```

R|R|Z|N

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`
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.)

R|R|Z|N

Module

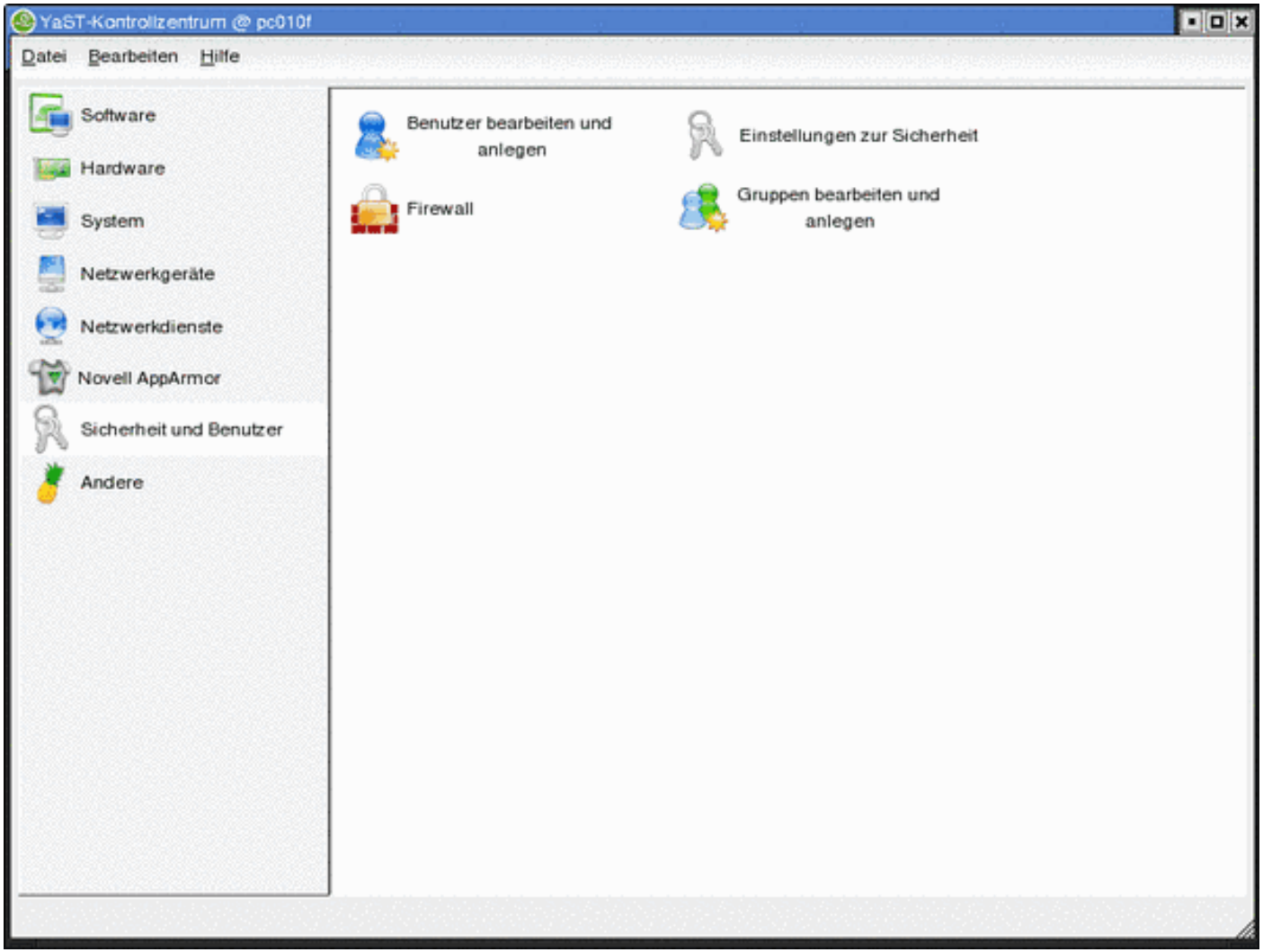
Einige IP-Tables-Erweiterungen liegen in Modulen, z.B.

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

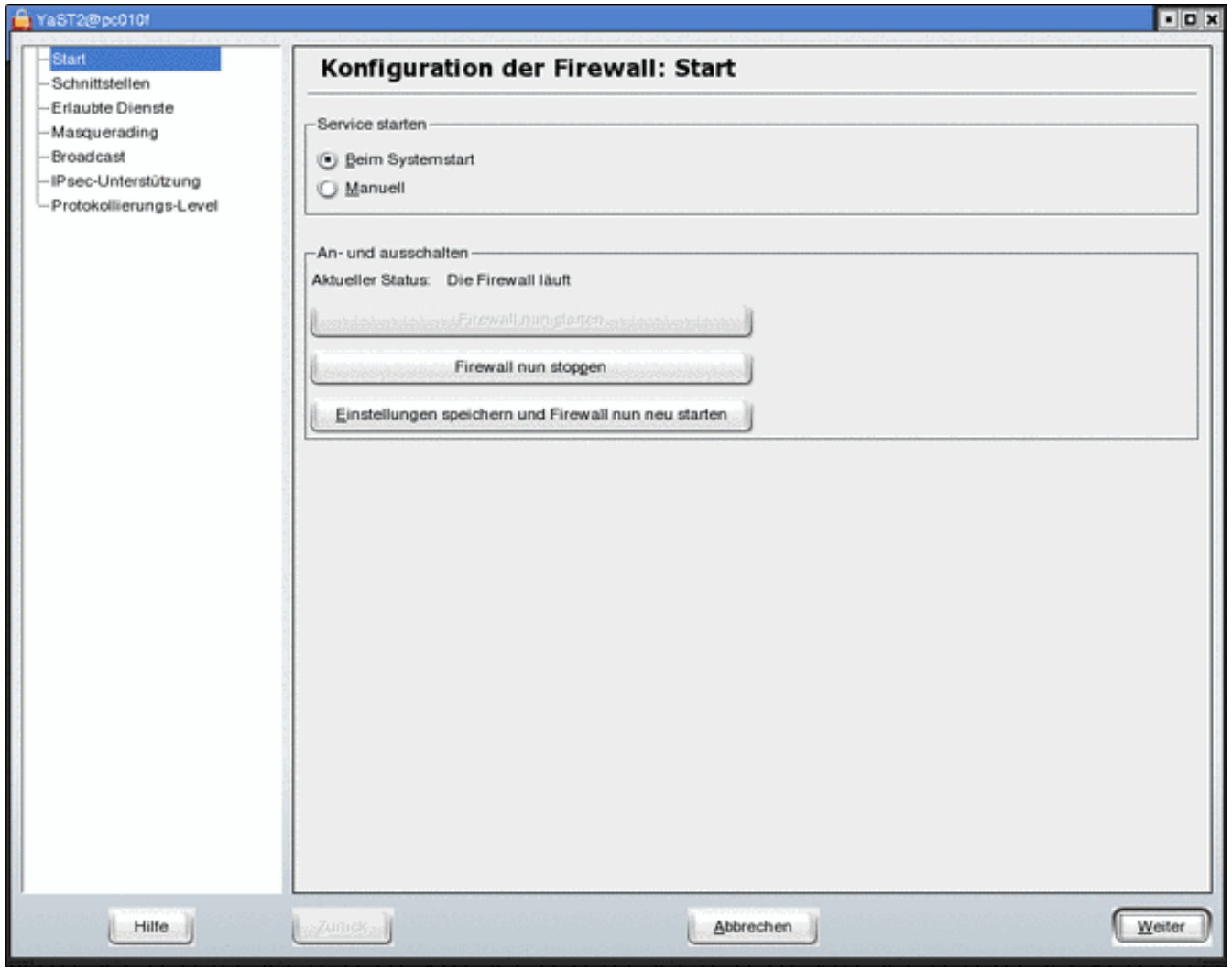
Diese Module sind ggf. beim Systemstart zu laden, durch

- 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



R|R|Z|N

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

R|R|Z|N

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`

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 gepackt in `/usr/share/doc/iptables/example/oldinitdscript.gz` 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`

- Links setzen mit: `update-rc.d iptables defaults`

R|R|Z|N

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|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`

recent-Modul

- Modul laden mit `modprobe ipt_recent`
- ermöglicht eine adaptive Firwall:
  - 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/ipt_recent/`
  - Liste manipulierbar durch Schreiben auf procfs-Datei: „xx.xx.xx.xx“ fügt IP zu, „-xx.xx.xx.xx“ löscht IP, „clear“ leert

Advanced: SSH-Bruteforce


```
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
                                          --name SSH \
5   -j DROP
6 iptables -A INPUT -p tcp --dport 22 \
7   -m recent --set --name SSH \
8   -j ACCEPT
```

in 4: SSH-Verbindungsaufbauten werden analysiert:

a) Kombination `hitcount&seconds`: wenn Zähler für die letzten 5 min $\geq$2, dann die Verbindung „droppen“

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

Advanced: Portscan-Abwehr


```
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`“).
- inzwischen extra Modul `psd` für Port-Scan-Detection verfügbar, aber nicht Standardumfang

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.

Advanced: Benutzerdefinierte Chains

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
```

R|R|Z|N

Übersicht

zu IP-Tables gibt es

- Skripterzeuger
- grafische Frontends
- Pakete für Konfiguration & Einbindung
- Stand-Alone-Firwalldistributionen
- Simulation, Paketverfolgung
- Modul-Erweiterungen
- Erweiterungen um Application-Level-FW

hier exemplarisch: Konfigurationshilfen

R|R|Z|N

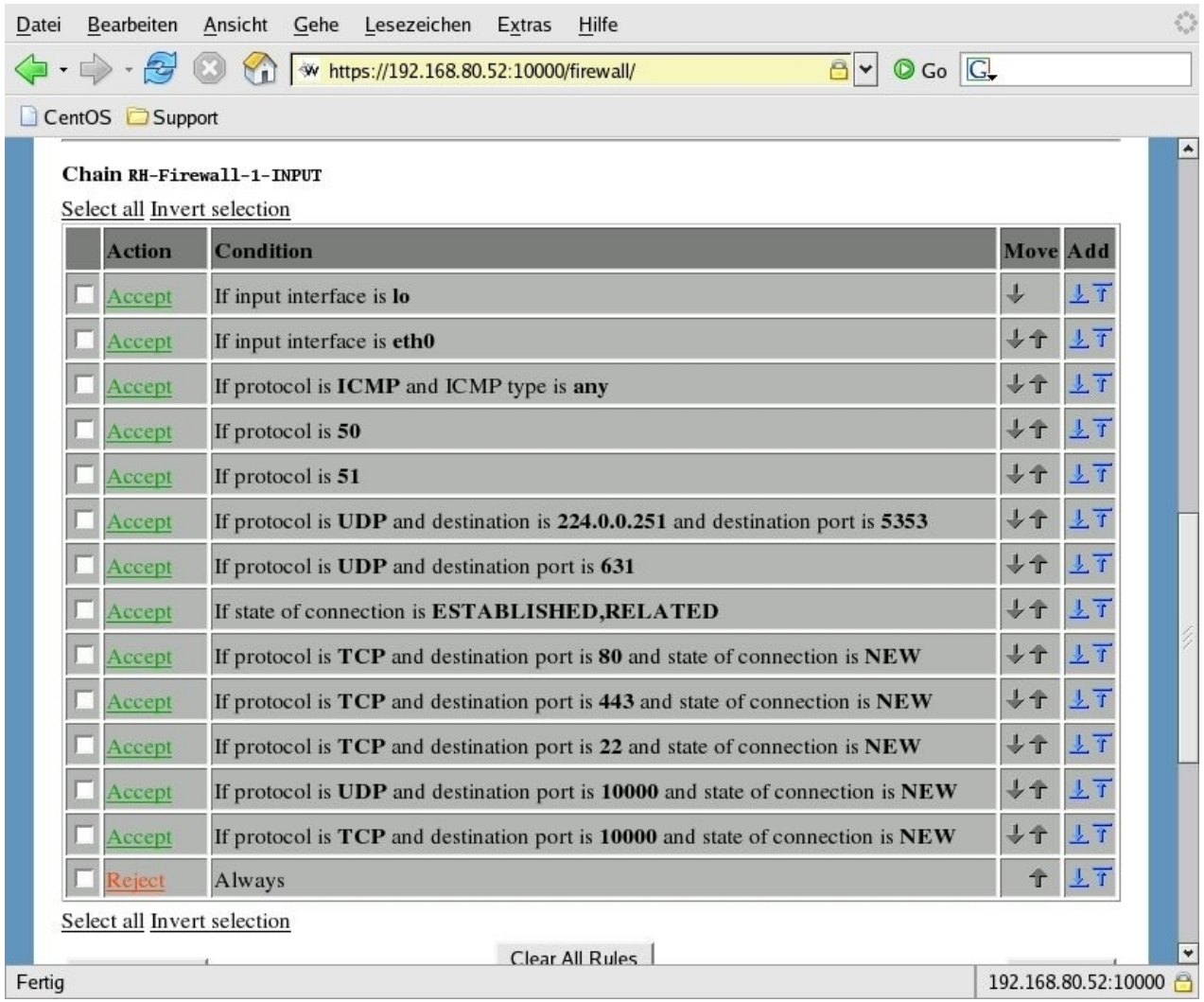
rc.firewall

- ein großes Init-Skript
- Konfiguration über Setzen von Variablen am Skriptanfang
- unterstützt auch NAT, Port-Forwarding, Routing
- Quelle: `http://projectfiles.com/firewall/`

Webmin

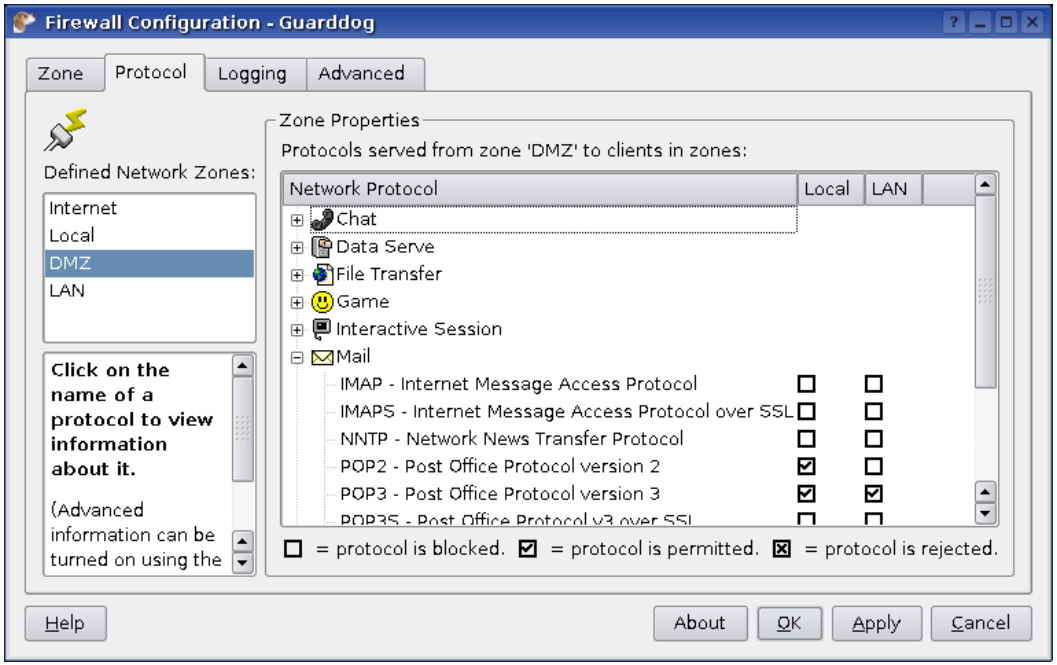
- IPTables-Modul vorhanden
- Konfiguration über Browser
- Auswahl aus vorgefertigten Scenarien
- detailierte Konfiguration möglich
- *aber* Webmin ist abzusichern, am Besten:
  nur lokaler Zugriff und Tunnelung über SSH
- Quelle:
  `http://www.niemueller.de/webmin/modules/iptables/`

Webmin: Screenshot



guarddog

- Zonen-Konzept
- Kapselung von TCP/UDP & Ports in „Application-Protokollen“, aber Port-Einstellungen möglich

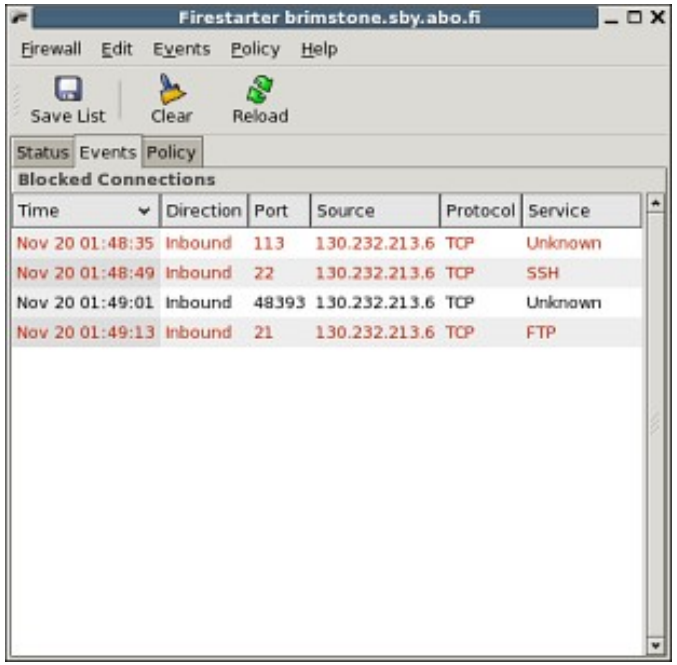


- Quelle: `http://www.simonzone.com/software/guarddog/`

R|R|Z|N

firestarter

- Regeln (policy) als Auflistung
- zeigt Verbindungsversuche (events) an, diese können zugelassen & als Regel eingetragen werden



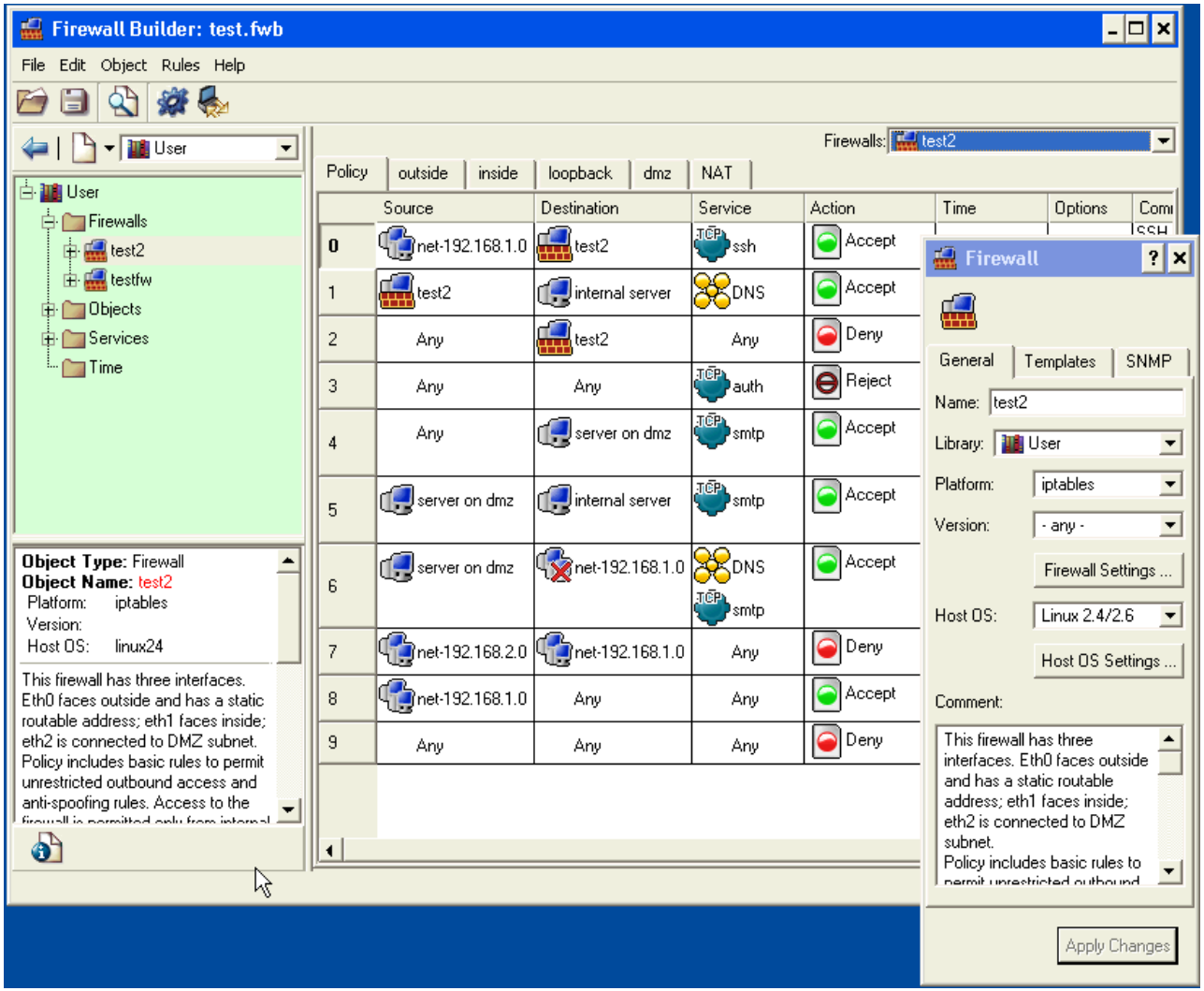
- Quelle: `http://www.fs-security.com/`

R|R|Z|N

FWBuilder

- GUI zur Regeleingabe & -bearbeitung
- 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
- Quelle: `http://www.fwbuilder.org/`

FWBuilder: Screenshot



R|R|Z|N

shorewall

- bindet Firewall in den Startup-Prozess ein
- Konfiguration aus mehreren Config-Dateien in `/etc/shorewall/`
  - Zonen-Konzept (Definition in `zones`
  - zonenabhängige Standard-Policies in `policy`
  - spezielle Regeln in `rules`
- eher für dedizierte Firewall, benötigt z.B. `iproute`
- unterstützt NAT, Port-Forwarding, Routing, Bridging, VPN, Traffic-Shaping, Accounting
- Quelle: `http://www.shorewall.net/`

Schlussbemerkungen

IP-Tables

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

$\longrightarrow$ 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`)

$\longrightarrow$ 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`
  bezogen auf Host-Absicherung (nicht NAT), Skripte zum Download
- rc.firewall, Port-Security explained
  `http://projectfiles.com/firewall/newbie.html`
  warum ICMP-Antworten & REJECT nicht schlechter sind als DROP